Fixed Virtual Platforms

Version 1.4

VE and MPS FVP Reference Guide

Fixed Virtual Platforms
VE and MPS FVP Reference Guide

Release Information

Change History

Date	Issue	Confidentiality	Change
May 2011	A	Non-Confidential	First release for Fast Models v6.1.
July 2011	B	Confidential	Update for Fast Models v6.2.
November 2011	C	Non-Confidential	Update for Fast Models v7.0.
May 2012	D	Non-Confidential	Update for Fast Models v7.1.
December 2012	E	Non-Confidential	Update for Fast Models v8.0.
May 2013	F	Non-Confidential	Update for Fast Models v8.1.

Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

`http://www.arm.com`

Contents

Fixed Virtual Platforms VE and MPS FVP Reference Guide

Chapter 1
Conventions and feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
<u>`mono`</u>`space`	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- your name and company

- the serial number of the product
- details of the release you are using
- details of the platform you are using, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tools, including the version number and build numbers.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0575F
- if viewing online, the topic names to which your comments apply
- if viewing a PDF version of a document, the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- *ARM Information Center*, `http://infocenter.arm.com/help/index.jsp`
- *ARM Technical Support Knowledge Articles*, `http://infocenter.arm.com/help/topic/com.arm.doc.faqs`
- *Support and Maintenance*, `http://www.arm.com/support/services/support-maintenance.php`
- *ARM Glossary*, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Chapter 2
Introduction

The following topics introduce you to the Fixed Virtual Platforms:

Concepts

2.1 About system models

Fixed Virtual Platforms (FVPs) enable development of software without the requirement for actual hardware. The software models provide *Programmer's View* (PV) models of processors and devices. The functional behavior of a model is equivalent to real hardware.

Absolute timing accuracy is sacrificed to achieve fast simulated execution speed. This means that you can use the PV models for confirming software functionality, but you must not rely on the accuracy of cycle counts, low-level component interactions, or other hardware-specific behavior.

System models are supplied as a *Component Architecture Debug Interface* (CADI) shared library, and are loaded by any environment compatible with the CADI API. Such environments include:

- Model Debugger
- Model Shell.

2.1.1 See also

Concepts

- *About the VE FVP* on page 2-3
- *About the MPS FVP* on page 2-6
- *MPS hardware* on page 2-7
- *MPS FVP* on page 2-8.

Reference

- *Model Debugger for Fast Models User Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0314-/index.html`
- *Model Shell for Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0457-/index.html`.

2.2 About the VE FVP

Versatile™ *Express* (VE) is a hardware development platform produced by ARM. The Motherboard Express *µAdvanced Technology Extended* (ATX) V2M-P1 is the basis for a highly-integrated software and hardware development system based on the ARM *Symmetric Multiprocessor System* (SMP) architecture.

The motherboard provides the following features:

- Peripherals for multimedia or networking environments.
- All motherboard peripherals and functions are accessed through a static memory bus to simplify access from daughterboards.
- High-performance PCI-Express slots for expansion cards.
- Consistent memory maps with different processor daughterboards simplify software development and porting.
- Automatic detection and configuration of attached CoreTile Express and LogicTile Express daughterboards.
- Automatic shutdown for over-temperature or power supply failure.
- System is unable to power-on if the daughterboards cannot be configured.
- Power sequencing of system.
- Supports drag and drop file update of configuration files.
- Uses either a 12V power-supply unit or an external ATX power supply.
- Supports FPGA and processor daughterboards to provide custom peripherals, or early access to processor designs, or production test chips. Supports test chips with an IO voltage range of 0.8 to 3.3 volts.

The VE FVP is a system model implemented in software. The model contains:
- virtual implementations of a motherboard
- a single daughterboard containing a specific ARM processor
- associated interconnections.

——— **Note** ———
The model is based on the VE platform memory map, but is not intended to be an accurate representation of a specific VE hardware revision. The VE FVP supports selected peripherals. For more information about these peripherals, see the reference information at the end of this topic. The supplied model is sufficiently complete and accurate to boot the same operating system images as for the VE hardware.

———————————

The model has been developed using the ARM Fast Models™ Portfolio product.

VE FVPs provide a functionally-accurate model for software execution. However, the model sacrifices timing accuracy to increase simulation speed. Key deviations from actual hardware are:
- timing is approximate
- buses are simplified
- caches for the processors and the related write buffers are not implemented.

The VE FVPs provided in this release are:

- FVP_VE_AEMv7A
- FVP_VE_AEMv8A
- FVP_VE_Cortex-A9x1
- FVP_VE_Cortex-A9x2
- FVP_VE_Cortex-A9x4
- FVP_VE_Cortex-A15x1
- FVP_VE_Cortex-A15x1-A7x1
- FVP_VE_Cortex-A15x2
- FVP_VE_Cortex-A15x4
- FVP_VE_Cortex-A15x4-A7x4
- FVP_VE_Cortex-R5x1
- FVP_VE_Cortex-R5x2
- FVP_VE_Cortex-R7x1
- FVP_VE_Cortex-R7x2

The following figure shows a block diagram of a top-level VE model with a Cortex-A15 cluster:

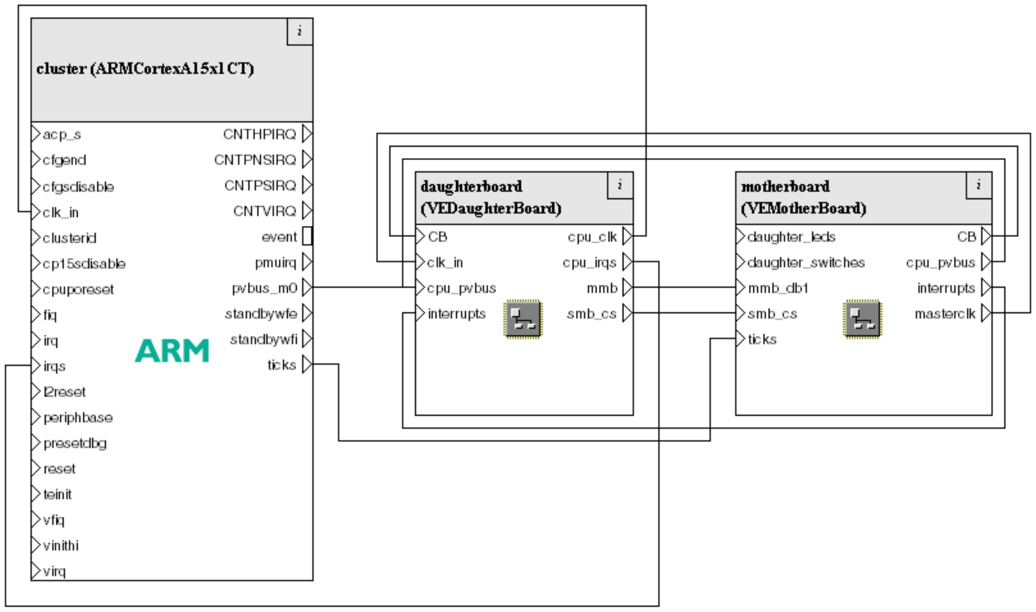


Figure 2-1 Block diagram of top-level VE model

2.2.1 See also

Concepts

- *About system models* on page 2-2
- *About the MPS FVP* on page 2-6

- *MPS hardware* on page 2-7
- *MPS FVP* on page 2-8
- *VE model parameters* on page 4-6
- *Differences between the VE and CoreTile hardware and the models* on page 4-58.

Reference

- Chapter 4 *Programmer's Reference for the VE FVPs*
- *FVP_VE_Cortex-A15MPxn CoreTile parameters* on page 4-21
- *FVP_VE_Cortex-A9 CoreTile parameters* on page 4-24
- *FVP_VE_Cortex-R5_MPxn CoreTile parameters* on page 4-25
- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41
- *Motherboard Express µATX V2M-P1 Technical Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0447-/index.html`.

2.3 About the MPS FVP

The *Microcontroller Prototyping System* (MPS) is a hardware development platform produced by Gleichmann Electronics Research. The ARM Hpe® module extends the hardware to support an ARM Cortex-M3 or Cortex-M4 processor implemented in a FPGA.

The *Microcontroller Prototyping System Fixed Virtual Platforms* (MPS FVPs) are system models implemented in software. They are developed using the ARM Fast Models library product.

Note

The MPS FVPs are provided as example platform implementations, and are not intended to be accurate representations of a specific hardware revision. The MPS FVP supports selected peripherals. For more information about these peripherals, see the reference information at the end of this topic. The supplied FVPs are sufficiently complete and accurate to boot the same application images as the MPS hardware.

2.3.1 See also

Concepts

- *About system models* on page 2-2
- *About the VE FVP* on page 2-3
- *MPS hardware* on page 2-7
- *MPS FVP* on page 2-8
- *VE model parameters* on page 4-6
- *Differences between the VE and CoreTile hardware and the models* on page 4-58.

Reference

- Chapter 5 *Programmer's Reference for the MPS FVPs*.

2.4 MPS hardware

The MPS hardware contains two FPGAs that implement the system:

CPU This FPGA contains:
- one instance of the Cortex-M3 or Cortex-M4 processor with ETM
- two memory controllers for RAM and FLASH on the board
- touchscreen interface
- pushbutton and DIP switch interfaces
- I2C interface
- an RS232 interface
- a configuration register block.

DUT This FPGA contains an example system that includes:
- timers
- display drivers (CLCD, character LCD, and seven-segment LED)
- audio interface
- pushbutton and DIP switch interfaces
- two RS232 interfaces
- an Hpe module interface
- MCI/SD card interface
- a USB interface.

The MPS FVPs provide a functionally-accurate model for software execution. However, the model sacrifices timing accuracy to increase simulation speed. Key deviations from actual hardware are:
- timing is approximate
- buses are simplified
- caches for the processors and the related write buffers are not implemented
- ETM is not modeled.

2.4.1 See also

Concepts
- *About system models* on page 2-2
- *About the VE FVP* on page 2-3
- *About the MPS FVP* on page 2-6
- *MPS FVP* on page 2-8
- *VE model parameters* on page 4-6
- *Differences between the VE and CoreTile hardware and the models* on page 4-58.

2.5 MPS FVP

The MPS FVP models in software some of the functionality of the MPS hardware.

A complete model implementation of the MPS platform includes both MPS-specific components and generic ones such as buses and timers. The following figure shows a block diagram of a MPS FVP:

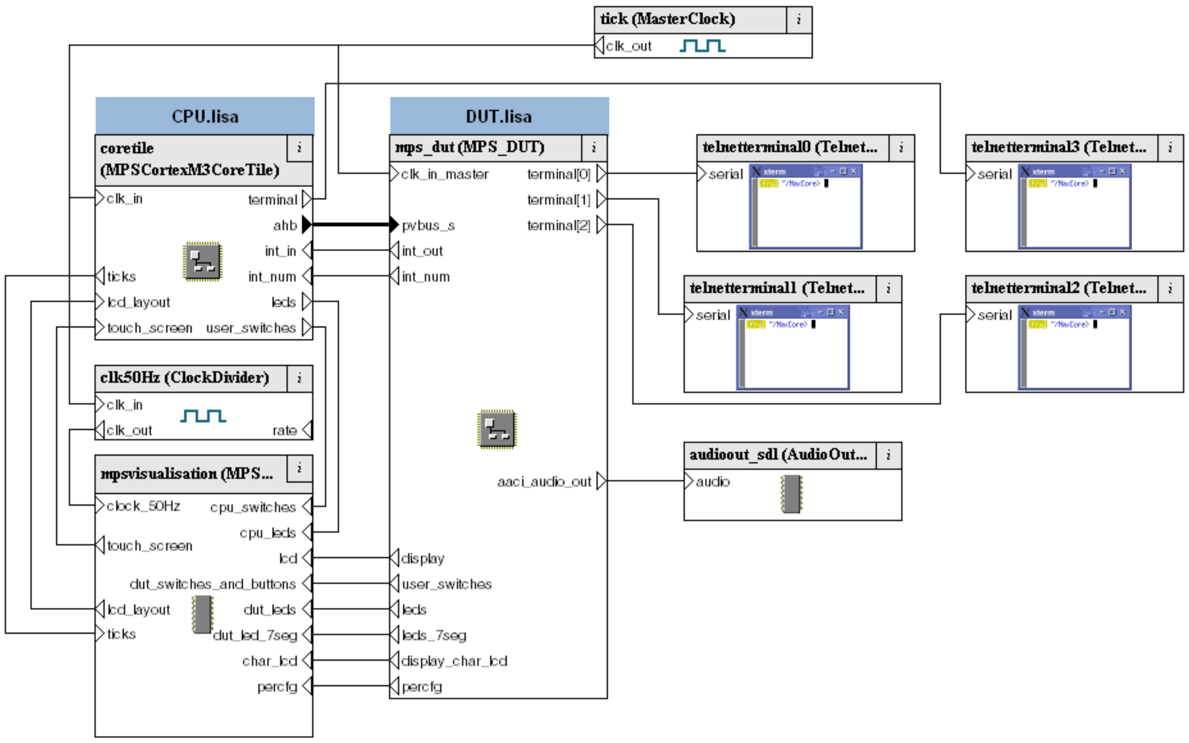


Figure 2-2 MPS FVP block diagram

2.5.1 See also

Concepts

- *About system models* on page 2-2
- *About the VE FVP* on page 2-3
- *About the MPS FVP* on page 2-6
- *MPS hardware* on page 2-7
- *VE model parameters* on page 4-6
- *Differences between the VE and CoreTile hardware and the models* on page 4-58.

Chapter 3
Getting Started with Fixed Virtual Platforms

The following topics describe the procedures for starting and configuring FVPs, and running a software application on the model. The procedures differ, depending on the ARM software tools that you are using.

Tasks

Reference

3.1 Debugging a FVP

To debug a FVP, you can either:

- start the FVP from within a debugger
- connect a debugger to a model that is already running.

You can use your own debugger if it has a CADI interface to connect to a FVP. For information about using your debugger in this way, see your debugger documentation.

3.1.1 Semihosting support

Semi-hosting 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. You can find more information on semihosting in the *ARM Compiler toolchain Developing Software for ARM Processors*.

The simulator handles semihosting by intercepting HLT `0xF000`, or SVC `0x123456` or `0xAB`, depending on whether the processor is in A64, A32 or T32 state. All other HLTs and SVCs are handled as normal.

If the operating system does not use HLT `0xF000`, SVC `0x123456` or `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 the documentation that accompanies your debugger.

3.1.2 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Configuring VE and MPS FVPs* on page 3-5
- *Using a configuration file* on page 3-5
- *Using the command line* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using the MPS Visualization window* on page 3-12
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *Using the VFS with a prebuilt FVP* on page 3-21
- *ARM Compiler toolchain Developing Software for ARM Processors*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0471-/index.html`.

3.2 Starting the FVP using Model Shell

You can use the Model Shell application to start VE and MPS FVPs. You can start the FVP with its own CADI debug server, enabling the model to run independently of a debugger. However, it means that you must configure your model using arguments that are passed to the model at start time.

To start the FVP using Model Shell:

1. Change to the directory where your model file is located.
2. Enter the following at the command prompt:

   `model_shell --cadi-server --model` *`model_name`* `[--config-file` *`filename`*`] [--parameter` *`instance.parameter=value`*`] [--application` *`app_filename`*`]`

   where:

   `model_name`
   : is the name of the model file. By default this file name is typically `FVP_VE_`*`processor`*`.dll` or `FVP_MPS_`*`processor`*`.dll` on Microsoft Windows or `FVP_VE_`*`processor`*`.so` or `FVP_MPS_`*`processor`*`.so` on Linux.

   `filename`
   : is the name of your optional plain-text configuration file. Configuration files simplify managing multiple parameters.

   `instance.parameter=value`
   : is the optional direct setting of a configuration parameter.

   `app_filename`
   : is the file name of an image to load to your model at startup.

The following example shows the format for using Model Shell to load and run an image from an ELF file:

Example 3-1 Load and run an image from an ELF file

```
# Load and run from an ELF image file
model_shell \
    --parameter "motherboard.vis.rate_limit-enable=0" \
    --application test_image.axf \
    FVP_VE_Cortex-A15x1.so
```

Note

On Microsoft Windows, it might be necessary to add to your `PATH` the directory in which the Model Shell executable is found. This location is typically:

```
install_directory\..\bin\model_shell
```

You can use `*` to load the same image into all the processors in one multiprocessor cluster, for example:

```
model_shell $MODEL -a "cluster0.*=image.axf"
```

Note

You must quote the argument as shown if you are using csh, or if you have spaces in the filename, otherwise the shell tries to expand the `*` instead of passing it to the application.

Starting the model opens the FVP CLCD display. After the FVP starts, you can use your debugger if it has a CADI interface to connect to the FVP.

3.2.1 See also

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Using a configuration file* on page 3-5
- *Using the command line* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using the MPS Visualization window* on page 3-12
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *Using the VFS with a prebuilt FVP* on page 3-21
- *Model Shell for Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0457-/index.html`.

3.3 Configuring VE and MPS FVPs

You can configure VE and MPS FVPs either by using a configuration GUI in your debugger or by setting model configuration options from Model Shell.

3.3.1 Using a configuration GUI in your debugger

In your debugger, you might be able to configure FVP parameters before you connect to the model and start it. See the documentation that accompanies your debugger.

Note

To connect to a FVP, your debugger must have a CADI interface.

3.3.2 Setting model configuration options from Model Shell

You can control the initial state of the FVP by configuration settings provided on the command line or in the CADI properties for the model.

Using a configuration file

To configure a model that you start from the command line with Model Shell, include a reference to an optional plain text configuration file when you are starting the FVP.

Comment lines in the configuration file must begin with a `#` character.

Each non-comment line of the configuration file contains:

- the name of the component instance
- the parameter to be modified and its value.

  Boolean values can be set using either `true`/`false` or `1`/`0`. You must enclose strings in double quotation marks if they contain whitespace.

The following example shows a typical configuration file:

Example 3-2 Configuration file

```
# Disable semihosting using true/false syntax
cluster.semihosting-enable=false
#
# Enable the boot switch using 1/0 syntax
motherboard.sp810_sysctrl.use_s8=1
#
# Set the boot switch position
motherboard.ve_sysregs_0.boot_switch_value=1
```

Using the command line

You can use the `-C` switch to define model parameters when you invoke the model. You can also use `--parameter` as a synonym for the `-C` switch. Use the same syntax as for a configuration file, but each parameter must be preceded by the `-C` switch.

The following example shows how to configure a MPS FVP using Model Shell:

Example 3-3 Using Model Shell to boot a model from a flash image

```
# Boot from a flash image
model_shell \
    --parameter "coretile.core.semihosting-cmd_line="\
    --parameter "coretile.fname=flash.bin" \
    --parameter "coretile.mps_sysregs.user_switches_value=4" \
    --parameter "coretile.mps_sysregs.memcfg_value=0" \
    --parameter "mpsvisualisation.disable-visualisation=false" \
    --parameter "mpsvisualisation.rate_limit-enable=0" \
    FVP_MPS_Cortex-M3.so
```

3.3.3 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Debugging a FVP* on page 3-2
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using the MPS Visualization window* on page 3-12
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19.
- *VE model parameters* on page 4-6
- *MPS parameters* on page 5-11.

3.4 Loading and running an application on the VE FVP

Example applications are provided for use with the FVPs for the VE system board.

Note

These applications are provided for demonstration purposes only and are not supported by ARM. The number of examples or implementation details might change with different versions of the system model.

A useful example application that runs on all versions of the VE FVP is:

`brot_ve.axf` This demo application provides a simple demonstration of rendering an image to the CLCD display. Source code is supplied.

In Fast Models, the examples are in the `%PVLIB_HOME%\images` directory.

If you are using non-Fast Models software, the source code might be in the directory `%ARMROOT%\Examples\…\…\platform\mandelbrot`.

3.4.1 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Using the VE CLCD window* on page 3-8
- *Using the MPS Visualization window* on page 3-12
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19.

3.5 Using the VE CLCD window

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

The following figure shows the VE FVP CLCD in its default state, immediately after being started.

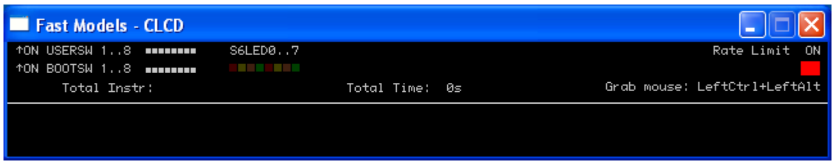


Figure 3-1 CLCD window at startup

The top section of the CLCD window displays the following status information:

USERSW Eight white boxes show the state of the VE 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 `0x10000004`.

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

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 `0x100000004`.

The switches are in the off position by default.

Note

ARM recommends that you configure the Boot DIP switches using the `boot_switch` model parameter instead of using the CLCD interface. 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 LEDs D[21:14] on the VE hardware, which are mapped to bits [7:0] of the SYS_LED register at address `0x10000008`. The boxes correspond to the red/yellow/green LEDs on the VE hardware.

Total Instr A counter showing the total number of instructions executed.

Because the FVP models provide a programmer's view of the system, the CLCD displays total instructions rather than total processor cycles. Timing might differ substantially from the hardware because:

- the 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 timing of operations within the model is not accurate.

Total Time A counter showing the total elapsed time, in seconds.

This 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 might cause timing issues.

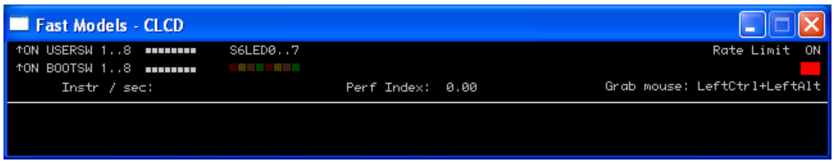
Rate Limit is enabled by default. Simulation time is restricted so that it more closely matches real time.

Click on the square button to disable or enable Rate Limit. The text changes from ON to OFF and the colored box becomes darker when Rate Limit is disabled. The figure below shows the CLCD with Rate Limit disabled.

Note

You can control whether Rate Limit is enabled by using the `rate_limit-enable` parameter, one of the visualization parameters for the MPS Visualization component, when instantiating the model.

When you click on the **Total Instr** or **Total Time** items in the CLCD, the display changes to show **Inst/sec** (instructions per second) and **Perf Index** (performance index). This is shown in the following figure:



```
Fast Models - CLCD
↑ON USERSW 1..8 ▪▪▪▪▪▪▪▪    S6LED0..7                                    Rate Limit  ON
↑ON BOOTSW 1..8 ▪▪▪▪▪▪▪▪
      Instr / sec:                      Perf Index:  0.00       Grab mouse: LeftCtrl+LeftAlt
```

Figure 3-2 CLCD window with Rate Limit ON

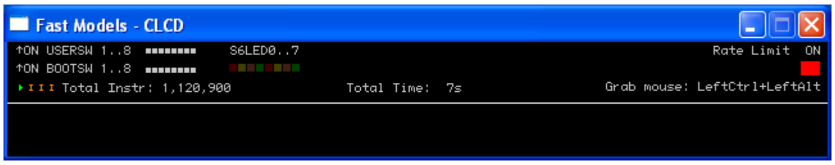
You can click on the items again to toggle between the original and alternative displays.

Instr/sec Shows the number of instructions executed 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 VE FVP CLCD displays the processor run state for each processor in the system by using one of eight colored icons. The icons are located to the left of the **Total Instr** (or **Inst/sec**) item, as shown for example in Figure 3-3:



```
Fast Models - CLCD
↑ON USERSW 1..8 ▪▪▪▪▪▪▪▪    S6LED0..7                                    Rate Limit  ON
↑ON BOOTSW 1..8 ▪▪▪▪▪▪▪▪
▸III Total Instr: 1,120,900             Total Time:  7s         Grab mouse: LeftCtrl+LeftAlt
```

Figure 3-3 Processor run state icons for a quad processor model

The description of each of the possible icons is shown in Table 3-1:

Table 3-1 Processor run state icon descriptions

Icon	State label	Description
?	`UNKNOWN`	Run status unknown, that is, simulation has not started.
▶	`RUNNING`	Processor running, is not idle, and is executing instructions.
⏸	`HALTED`	External halt signal asserted.
E	`STANDBY_WFE`	Last instruction executed was WFE and standby mode has been entered.
I	`STANDBY_WFI`	Last instruction executed was WFI and standby mode has been entered.
■	`IN_RESET`	External reset signal asserted.
🟧	`DORMANT`	Partial processor power down.
🟥	`SHUTDOWN`	Complete processor power down.

Note

The icons do not appear until you start the simulation.

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. This is then streamed to the KMI peripheral model FIFOs.

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 on the right of the keyboard does not have the same effect.

If you prefer to use a different key, use the `trap_key` configuration option, one of the visualization parameters for the MPS Visualization component.

3.5.1 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

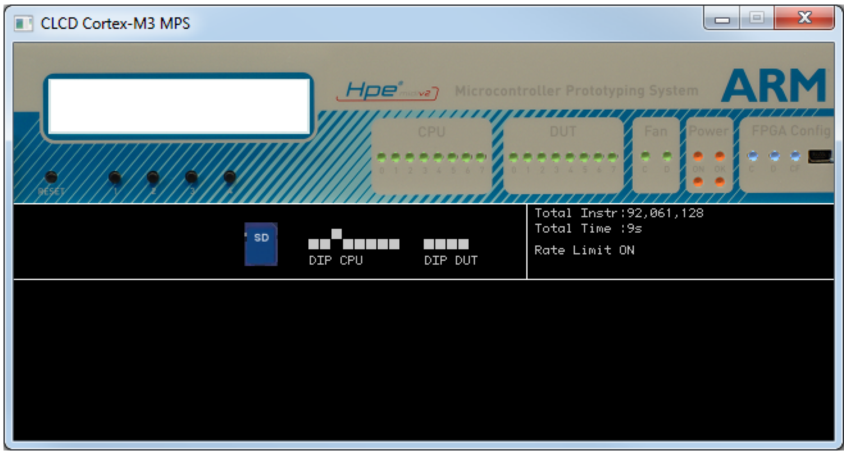
Reference

- *Debugging a FVP* on page 3-2

- *Configuring VE and MPS FVPs* on page 3-5
- *Using the MPS Visualization window* on page 3-12
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *Timing considerations* on page 4-65
- *Visualization parameters* on page 4-20
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

3.6 Using the MPS Visualization window

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



```
Total Instr:92,061,128
Total Time :9s
Rate Limit ON
```

Figure 3-4 Visualization window at startup

The top section of the CLCD window displays the following status information:

Character LCD	The large box shows the state of the character LCD.
CPU	Eight colored circles indicate the state of the processor LEDs.
DUT	Eight colored circles indicate the state of the DUT LEDs.
Fan	Two colored circles indicate the state of the fan LEDs.
Power	Four colored circles indicate the state of the power LEDs.
FPGA Config	Three colored circles indicate the state of the FPGA configuration LEDs.
SD	The box with the letters SD indicates the state of the SD memory. Click the box to enable or disable the device.
DIP CPU	Eight white boxes show the state of the processor switches.
DIP DUT	Four white boxes show the state of the DUT switches.

——— **Note** ———

ARM recommends that you configure the Boot DIP switches using the `boot_switch` model parameter instead of using the CLCD interface.

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

———————————

Total Instr	A counter showing the total number of instructions executed.

The system models provide a programmer's view of the system, so the total instructions are displayed rather than total processor cycles. Timing might differ substantially from the hardware because:

- the 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 timing of operations within the model is not accurate.

Total Time A counter showing the total elapsed time, in seconds.

This 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 might cause timing issues.

If Rate Limit is enabled, the default simulation time is restricted so that it more closely matches real time.

Click on the square button to disable or enable Rate Limit. The text changes from ON to OFF and the colored box becomes darker when Rate Limit is disabled. The figure below shows the CLCD with Rate Limit enabled.

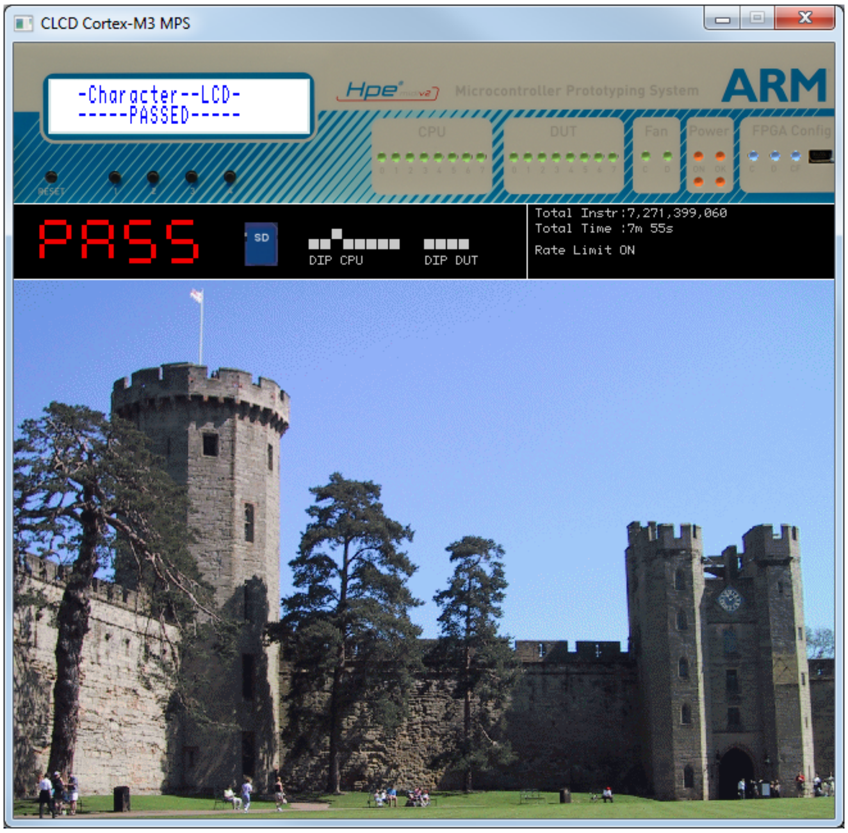
——— **Note** ———

You can control whether Rate Limit is enabled by using the `rate_limit-enable` parameter, one of the visualization parameters for the MPS Visualization component, when instantiating the model.

———————

CLCD display

The area at the bottom of the window displays the contents of the CLCD buffer.



```
Total Instr :7,271,399,060
Total Time :7m 55s
Rate Limit ON
```

Figure 3-5 Visualization window with CLCD buffer displayed

If the CLCD component is not used in the simulation, the display area is black.

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 on the right of the keyboard does not have the same effect.

If you prefer to use a different key, use the `trap_key` configuration option, one of the visualization parameters for the MPS Visualization component.

3.6.1 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Using the VE CLCD window* on page 3-8
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *MPS visualization parameters* on page 5-12
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

3.7 Using Ethernet with a VE FVP

The VE FVPs provide you with a virtual Ethernet component. This is a model of the SMSC91C111 Ethernet controller, and uses a TAP device to communicate with the network. By default, the Ethernet component is disabled.

3.7.1 Host requirements

Before you can use the Ethernet capability of the VE FVP, you must first set up your host computer. You can find more information in the *Fast Models User Guide*.

3.7.2 Target requirements

The VE FVPs include a software implementation of the SMSC91C111 Ethernet controller. Your target OS must therefore include a driver for this specific device, and you must configure the kernel to use the SMSC chip. Linux is the operating system that supports the SMSC91C111.

There are three SMSC91C111 component parameters, and when you configure these prior to starting the VE FVP:

you

- specify the TAP device name
- you set the MAC address
- you define whether promiscuous mode is enabled.

enabled

This is the default state. When the device is disabled, the kernel cannot detect the device. For more information, see the SMSC_91C111 component section in the *Fast Models Reference Manual*. The following figure shows a block diagram of the model networking structure:

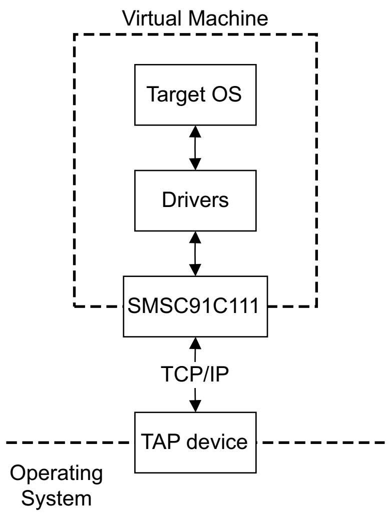


Figure 3-6 Model networking structure block diagram

You must configure a HostBridge component to perform read and write operations on the TAP device. The HostBridge component is a virtual programmer's view model, acting as a networking 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 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. This means that it receives all network traffic, even any not specifically addressed to the device. You must use this mode if you are using a single network device for multiple MAC addresses. Use this mode if, for example, you are sharing the same 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.

3.7.3 Configuring Ethernet

For information on configuring a connection to the Ethernet interface on the FVP from Microsoft Windows or Linux, see the *Fast Models User Guide*.

3.7.4 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *Using the VFS with a prebuilt FVP* on page 3-21
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`
- *Fast Models User Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0370-/index.html`.

3.8 Using a terminal with a system model

The Terminal 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.

——— **Note** ———

To use the Terminal component with a Microsoft Windows 7 client, you must first install Telnet. The Telnet application is not installed on Microsoft Windows 7 by default.

Download the application by following the instructions on the Microsoft web site. Search for "Windows 7 Telnet" to find the Telnet FAQ page. To install Telnet:

1. Select **Start** → **Control Panel** → **Programs and Features**. This opens a window that enables you to uninstall or change programs.
2. Select **Turn Windows features on or off** on the left side of the bar. This opens the Microsoft Windows Features dialog. Select the **Telnet Client** check box.
3. Click **OK**. The installation of Telnet might take several minutes to complete.

———————

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

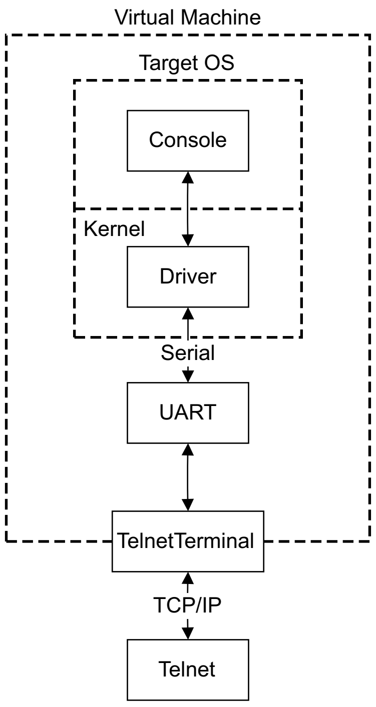


Figure 3-7 Terminal block diagram

On the target side, the console process 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, which exposes a TCP/IP port to the world outside of the FVP. This port can be connected to by, for example, a Telnet process on the host.

By default, the VE FVP or MPS FVP starts four telnet Terminals when the model is initialized. You can change the startup behavior for each of the four Terminals by modifying the corresponding component parameters.

If the Terminal connection is broken, for example by closing a client telnet session, the port is re-opened on the host. This might have a different port number if the original one is no longer available. Before the first data access, you can connect a client of your choice to the network socket. If there is no existing connection when the first data access is made, and the `start_telnet` parameter is `true`, a host telnet session is started automatically.

The port number of a particular Terminal instance can be defined when the FVP starts. The actual value of the port used by each Terminal is declared when it starts or restarts, and might not be the value 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.

You can start the Terminal component in either telnet mode or raw mode.

3.8.1 Telnet mode

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 flow control is not implemented.

3.8.2 Raw mode

Raw mode enables the byte stream to pass unmodified between the host and the target. This means that the Terminal component does not participate in initial capability negotiations between the host and client. It acts as a TCP/IP port. You can use this feature to directly connect to your target through the Terminal component.

3.8.3 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using Ethernet with a VE FVP* on page 3-15
- *Virtual filesystem* on page 3-19
- *Using the VFS with a prebuilt FVP* on page 3-21
- *Terminal parameters* on page 4-18.

3.9 Virtual filesystem

The *Virtual FileSystem* (VFS) enables your target to access parts of a host filesystem. This access is achieved through a target OS-specific driver and a memory-mapped device called the MessageBox. When using the VFS, access to the host filesystem is analogous to access to a shared network drive, and can be expected to behave in the same way.

If you want to build your own system that includes the VFS, see the reference information at the end of this topic. See also the `WritingADriver.txt` file in `%PVLIB_HOME%\VFS2\docs\`.

Note

VFS support is only provided by VE FVP models. MPS FVP models do not support VFS functionality.

The VFS supports the following filesystem operations:

getattr	retrieves metadata for the file, directory or symbolic link
mkdir	creates a new directory
remove	removes a file, directory or symbolic link
rename	renames a file, directory or symbolic link
rmdir	removes an empty directory
setattr	sets metadata for the file, directory or symbolic link.

Note

`setattr` is not currently implemented.

Symbolic links are not currently supported. Hard links cannot be created by the model but hard links created by the host operating system function correctly.

The VFS supports the following mount points:

closemounts	frees the iterator handle returned from `openmounts`
openmounts	retrieves an iterator handle for the list of available mounts
readmounts	reads one entry from the mount iterator ID.

The VFS supports the following directory iterators:

closedir	frees a directory iterator handle retrieved by `opendir`
opendir	retrieves an iterator handle for the directory specified
readdir	reads the next entry from the directory iterator.

Note

Datestamps returned are in milliseconds elapsed since the VFS epoch of January 01 1970 00:00 UTC and are host datestamps. The host datestamp might be in the future relative to the simulated OS datestamp.

The VFS supports the following file operations:

closefile	frees a handle opened with `openfile`
filesync	forces the host OS to flush all file data to persistent storage
getfilesize	returns the current size of a file, in bytes
openfile	returns a handle to the file specified

readfile	reads a block of data from a file
setfilesize	sets the current size of a file in bytes, either by truncating, or extending the file with zeroes
writefile	writes a block of data to a file.

3.9.1 See also

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Using the VFS with a prebuilt FVP* on page 3-21
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

3.10 Using the VFS with a prebuilt FVP

The supplied VE FVPs include the necessary VFS components. This permits you to run a Linux image, for example, on the VE FVP and access the filesystem running on your computer.

To use the VFS functionality of the VE FVP, use the `motherboard.vfs2.mount` configuration parameter when you start the model. The value of the parameter is the path to the host filesystem directory that is to be made accessible within the model.

3.10.1 Mount names

When the target OS is running, create a mount point, such as `/mnt/host`. For example, on a Linux target, use the `mount` command as follows:

```
mount -t vmfs A /mnt/host
```

You can then access the host filesystem from the target OS through a supported filesystem operation. See the `ReadMe.txt` file in the `%PVLIB_HOME%\VFS2\linux\` directory.

3.10.2 Path names

All path names must be fully qualified paths of the form:

```
mountpoint:/path/to/object
```

3.10.3 See also

Reference

- *Debugging a FVP* on page 3-2
- *Configuring VE and MPS FVPs* on page 3-5
- *Loading and running an application on the VE FVP* on page 3-7
- *Using the VE CLCD window* on page 3-8
- *Using Ethernet with a VE FVP* on page 3-15
- *Using a terminal with a system model* on page 3-17
- *Virtual filesystem* on page 3-19
- *VFS2 parameters* on page 4-19
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

Chapter 4
Programmer's Reference for the VE FVPs

The following topics describe the memory map and the configuration registers for the peripheral and system component models.

Note

For detailed information on the programming interface for ARM PrimeCell peripherals and controllers, see the appropriate technical reference manual.

Tasks

- *Starting the FVP using Model Shell* on page 3-3.

Reference

4.1 VE model memory map

The following table shows the global memory map for the platform model. This map is based on the Versatile Express RS1 memory map with the RS2 extensions.

Table 4-1 Memory map

Peripheral	Modeled	Address range	Size
NOR FLASH0 (CS0)	Yes	`0x00_00000000-0x00_03FFFFFF`	64MB
Reserved	-	`0x00_04000000-0x00_07FFFFFF`	64MB
NOR FLASH0 alias (CS0)	Yes	`0x00_08000000-0x00_0BFFFFFF`	64MB
NOR FLASH1 (CS4)	Yes	`0x00_0C000000-0x00_0FFFFFFF`	64MB
Unused (CS5)	-	`0x00_10000000-0x00_13FFFFFF`	-
PSRAM (CS1) - unused	No	`0x00_14000000-0x00_17FFFFFF`	-
Peripherals (CS2). See Table 4-3 on page 4-4.	Yes	`0x00_18000000-0x00_1BFFFFFF`	64MB
Peripherals (CS3). See Table 4-4 on page 4-4.	Yes	`0x00_1C000000-0x00_1FFFFFFF`	64MB
CoreSight and peripherals	No	`0x00_20000000-0x00_2CFFFFFF`[a]	-
Graphics space	No	`0x00_2D000000-0x00_2D00FFFF`	-
System SRAM	Yes	`0x00_2E000000-0x00_2EFFFFFF`	64KB
Ext AXI	No	`0x00_2F000000-0x00_7FFFFFFF`	-
4GB DRAM (in 32-bit address space)[b]	Yes	`0x00_80000000-0x00_FFFFFFFF`	2GB
Unused	-	`0x01_00000000-0x07_FFFFFFFF`	-
4GB DRAM (in 36-bit address space)[b]	Yes	`0x08_00000000-0x08_FFFFFFFF`	4GB
Unused	-	`0x09_00000000-0x7F_FFFFFFFF`	-
4GB DRAM (in 40-bit address space)[b]	Yes	`0x80_00000000-0xFF_FFFFFFFF`	4GB

a. The private peripheral region address `0x2c000000` is mapped in this region. The parameter PERIPHBASE can be used to map the peripherals to a different address.

b. The model contains only 4GB of DRAM. The DRAM memory address space is aliased across the three different regions and where the mapped address space is greater than 4GB.

The model has a `secure_memory` option. When you enable this option, the memory map is changed for a number of peripherals as shown in the following table:

Table 4-2 CS2 peripheral memory map for secure_memory option

Peripheral	Address range	Functionality with secure_memory enabled
NOR FLASH0 (CS0)	`0x00_00000000-0x00_0001FFFF`	Secure RO, aborts on non-secure accesses.
Reserved	`0x00_04000000-0x00_0401FFFF`	Secure SRAM, aborts on non-secure accesses.
NOR FLASH0 alias (CS0)	`0x00_08000000-0x00_7DFFFFFF`	Normal memory map, aborts on secure accesses.
Ext AXI	`0x00_7e000000-0x00_7FFFFFFF`	Secure DRAM, aborts on non-secure accesses.
4GB DRAM (in 32-bit address space)	`0x00_80000000-0xFF_FFFFFFFF`	Normal memory mpa, aborts on secure accesses.

The following table shows details of the memory map for peripherals in the CS2 region:

Table 4-3 CS2 peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[a]
VRAM - aliased	Yes	`0x00_18000000-0x00_19FFFFFF`	32MB	-
Ethernet (SMSC 91C111)	Yes	`0x00_1A000000-0x00_1AFFFFFF`	16MB	47
USB - unused	No	`0x00_1B000000-0x00_1BFFFFFF`	16MB	-

a. The Interrupt signal column lists the values to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.

The following table shows details of the memory map for peripherals in the CS3 region:

Table 4-4 CS3 peripheral memory map

Peripheral	Modeled	Address range	Size	GIC Int[a]
Local DAP ROM	No	`0x00_1C000000-0x00_1C00FFFF`	64KB	-
VE System Registers	Yes	`0x00_1C010000-0x00_1C01FFFF`	64KB	-
System Controller (SP810)	Yes	`0x00_1C020000-0x00_1C02FFFF`	64KB	-
TwoWire serial interface (PCIe)	No	`0x00_1C030000-0x00_1C03FFFF`	64KB	-
AACI (PL041)	Yes	`0x00_1C040000-0x00_1C04FFFF`	64KB	43
MCI (PL180)	Yes	`0x00_1C050000-0x00_1C05FFFF`	64KB	41, 42
KMI - keyboard (PL050)	Yes	`0x00_1C060000-0x00_1C06FFFF`	64KB	44
KMI - mouse (PL050)	Yes	`0x00_1C070000-0x00_1C07FFFF`	64KB	45

Table 4-4 CS3 peripheral memory map (continued)

Peripheral	Modeled	Address range	Size	GIC Int[a]
Reserved	-	`0x00_1C080000-0x00_1C08FFFF`	64KB	-
UART0 (PL011)	Yes	`0x00_1C090000-0x00_1C09FFFF`	64KB	37
UART1 (PL011)	Yes	`0x00_1C0A0000-0x00_1C0AFFFF`	64KB	38
UART2 (PL011)	Yes	`0x00_1C0B0000-0x00_1C0BFFFF`	64KB	39
UART3 (PL011)	Yes	`0x00_1C0C0000-0x00_1C0CFFFF`	64KB	40
VFS2	Yes	`0x00_1C0D0000-0x00_1C0DFFFF`	64KB	73
Reserved	-	`0x00_1C0E0000-0x00_1C0EFFFF`	64KB	-
Watchdog (SP805)	Yes	`0x00_1C0F0000-0x00_1C0FFFFF`	64KB	32
Reserved	-	`0x00_1C100000-0x00_1C10FFFF`	64KB	-
Timer-0 (SP804)	Yes	`0x00_1C110000-0x00_1C11FFFF`	64KB	34
Timer-1 (SP804)	Yes	`0x00_1C120000-0x00_1C12FFFF`	64KB	35
Reserved	-	`0x00_1C130000-0x00_1C15FFFF`	192KB	-
TwoWire serial interface (DVI) - unused	No	`0x00_1C160000-0x00_1C16FFFF`	64KB	-
Real-time Clock (PL031)	Yes	`0x00_1C170000-0x00_1C17FFFF`	64KB	36
Reserved	-	`0x00_1C180000-0x00_1C19FFFF`	128KB	-
CF Card - unused	No	`0x00_1C1A0000-0x00_1C1AFFFF`	64KB	
Reserved	-	`0x00_1C1B0000-0x00_1C1EFFFF`	256KB	-
Color LCD Controller (PL111)	Yes	`0x00_1C1F0000-0x00_1C1FFFFF`	64KB	46
Reserved	-	`0x00_1C200000-0x00_1FFFFFFF`	62KB	-

a. The Interrupt signal column lists the values to use to program your interrupt controller. The values shown are after mapping the SPI number by adding 32. The interrupt numbers from the peripherals are modified by adding 32 to form the interrupt number seen by the GIC. GIC interrupts 0-31 are for internal use.

Note

The VE FVP implementation of memory does not require programming the memory controller with the correct values. This means you must ensure that the memory controller is set up properly if you run an application on actual hardware. If this is not done, applications that run on a FVP might fail on actual hardware.

4.1.1 See also

Reference

- *VE model parameters* on page 4-6
- *Differences between the VE and CoreTile hardware and the models* on page 4-58.

4.2 VE model parameters

The Fixed Virtual Platforms for the VE reference system have configuration parameters that you can define at run time:

- *Motherboard peripheral parameters* on page 4-7
- *Motherboard virtual component parameters* on page 4-14
- *FVP_VE_Cortex-A15MPxn CoreTile parameters* on page 4-21
- *FVP_VE_Cortex-A9 CoreTile parameters* on page 4-24
- *FVP_VE_Cortex-R5_MPxn CoreTile parameters* on page 4-25
- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41.

Note

Parameters that can be modified only at model build time, or that are not normally modified by the user in the equivalent hardware system, are not discussed.

4.2.1 See also

Reference

- *VE model memory map* on page 4-3
- *Differences between the VE and CoreTile hardware and the models* on page 4-58.

4.3 Motherboard peripheral parameters

You can configure the following peripheral parameters on the motherboard:

- *Color LCD controller parameters* on page 4-8
- *Ethernet parameters* on page 4-9
- *System controller parameters* on page 4-10
- *VE system register block parameters* on page 4-11
- *UART parameters* on page 4-12
- *Watchdog parameters* on page 4-13.

4.3.1 See also

Reference

- *FLASH loader parameters* on page 4-15
- *Host bridge parameters* on page 4-16
- *Multimedia card parameters* on page 4-17
- *Terminal parameters* on page 4-18
- *Visualization parameters* on page 4-20.

4.4 Color LCD controller parameters

The table below lists the Color LCD Controller instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

`motherboard.pl111_clcd.`*`parameter`*`=`*`value`*

Table 4-5 Color LCD controller parameters

Parameter	Description	Type	Values	Default
`pixel_double_limit`	The threshold in horizontal pixels below which pixels sent to the frame-buffer are doubled in size in both dimensions.	Integer	-	`0x12C`

4.4.1 See also

Reference

- *Motherboard peripheral parameters* on page 4-7
- *Ethernet parameters* on page 4-9
- *System controller parameters* on page 4-10
- *VE system register block parameters* on page 4-11
- *UART parameters* on page 4-12
- *Watchdog parameters* on page 4-13.

4.5 Ethernet parameters

The table below lists the Ethernet instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.smsc_91c111.parameter=value
```

Table 4-6 Ethernet parameters

Parameter	Description	Type	Values	Default
`enabled`	Host interface connection enabled	Boolean	`true` or `false`	`false`
`mac_address`	Host/model MAC address	String	See *mac_address parameter*	`00:02:f7:ef:31:11`
`promiscuous`	Put host into promiscuous mode, for example when sharing the Ethernet controller with the host OS.	Boolean	`true` or `false`	`true`

4.5.1 mac_address parameter

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 and changes its bottom two bytes from `00:02` to the bottom two bytes of the MAC address of one of the adaptors on the host PC. This provides some degree of MAC address uniqueness when running models on multiple hosts on a local network.
- If you specify the MAC address as auto, this generates a completely random local MAC address each time the simulator is run. The address has bit 1 set and bit 0 clear in the first byte to indicate a locally-administered unicast MAC address.

Note

DHCP servers are used to allocate IP addresses, but because they sometimes do this based on the MAC address provided to them, then using random MAC addresses might interact with some DHCP servers.

4.5.2 See also

Reference

- *Using Ethernet with a VE FVP* on page 3-15
- *Motherboard peripheral parameters* on page 4-7
- *Color LCD controller parameters* on page 4-8
- *System controller parameters* on page 4-10
- *VE system register block parameters* on page 4-11
- *UART parameters* on page 4-12
- *Watchdog parameters* on page 4-13.

4.6 System controller parameters

The table below lists the system controller instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.sp810_sysctrl.parameter=value
```

Table 4-7 System controller parameters

Parameter	Description	Type	Values	Default
`sysid`	Value for system identification register	Integer	0, 1, 2[a]	`0x00000000`
`use_s8`	Select whether switch S8 is enabled	Boolean	`true` or `false`	`false`

a. The sysid parameter takes values 0, 1, or 2. These correspond to SYS_ID register read values of:
`sysid` parameter value = 0 => SYS_ID register value = `0x0225f500`, corresponding to REV_A
`sysid` parameter value = 1 => SYS_ID register value = `0x12257500`, corresponding to REV_B
`sysid` parameter value = 2 => SYS_ID register value = `0x22252500`, corresponding to REV_C.
Any other value for parameter `sysid` results in a SYS_ID register value of `0x0`.

4.6.1 See also

Reference

- *Motherboard peripheral parameters* on page 4-7
- *Color LCD controller parameters* on page 4-8
- *Ethernet parameters* on page 4-9
- *VE system register block parameters* on page 4-11
- *UART parameters* on page 4-12
- *Watchdog parameters* on page 4-13.

4.7 VE system register block parameters

The table below lists the VE system register instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.ve+sysregs.parameter=value
```

Table 4-8 System register parameters

Parameter	Description	Type	Values	Default
`user_switches_value`	User switch	Integer	-	`0x00`
`tilePresent`	CoreTile fitted status	Boolean	`true` or `false`	`true`

4.7.1 See also

Reference

- *Motherboard peripheral parameters* on page 4-7
- *Color LCD controller parameters* on page 4-8
- *Ethernet parameters* on page 4-9
- *System controller parameters* on page 4-10
- *UART parameters* on page 4-12
- *Watchdog parameters* on page 4-13.

4.8 UART parameters

The table below lists the UART instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.pl011_uartx.parameter=value
```

where *x* is the UART identifier 0, 1, 2 or 3.

Table 4-9 UART parameters

Parameter	Description	Type	Values	Default
`baud_rate`	Baud rate	Integer	-	0x9600
`clock_rate`	Clock rate for PL011	Integer	-	0xE10000
`in_file`	Input file	String		[empty string]
`out_file`	Output file (use "-" to send all output to stdout)	String		[empty string]
`in_file_escape_sequence`	Input file escape sequence	String		##
`shutdown_on_eot`	Shutdown simulation when an EOT (ASCII 4) char is transmitted	Boolean	`true` or `false`	`false`
`unbufferred_output`	Unbuffered output	Boolean	`true` or `false`	`false`
`untimed_fifos`	Ignore the clock rate and transmit/receive serial data immediately	Boolean	`true` or `false`	`false`
`uart_enable`	Enable the UART when the system starts	Boolean	`true` or `false`	`false`

4.8.1 See also

Reference

- *Motherboard peripheral parameters* on page 4-7
- *Color LCD controller parameters* on page 4-8
- *Ethernet parameters* on page 4-9
- *System controller parameters* on page 4-10
- *VE system register block parameters* on page 4-11
- *Watchdog parameters* on page 4-13.

4.9 Watchdog parameters

The table below lists the watchdog instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.sp805_wdog.parameter=value
```

Table 4-10 Watchdog parameters

Parameter	Description	Type	Values	Default
`simhalt`	Halt on reset	Boolean	`true` or `false`	`false`

4.9.1 See also

Reference

- *Motherboard peripheral parameters* on page 4-7
- *Color LCD controller parameters* on page 4-8
- *Ethernet parameters* on page 4-9
- *System controller parameters* on page 4-10
- *VE system register block parameters* on page 4-11
- *UART parameters* on page 4-12.

4.10 Motherboard virtual component parameters

This following topics describe the virtual component parameters that you can change on the motherboard:

- *FLASH loader parameters* on page 4-15
- *Host bridge parameters* on page 4-16
- *Multimedia card parameters* on page 4-17
- *Terminal parameters* on page 4-18
- *VFS2 parameters* on page 4-19
- *Visualization parameters* on page 4-20.

4.10.1 See also

Reference

- *VE model parameters* on page 4-6
- *Motherboard peripheral parameters* on page 4-7
- *FVP_VE_Cortex-A15MPxn CoreTile parameters* on page 4-21
- *FVP_VE_Cortex-A9 CoreTile parameters* on page 4-24
- *FVP_VE_Cortex-R5_MPxn CoreTile parameters* on page 4-25
- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41

4.11 FLASH loader parameters

The table below lists the FLASH loader instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.flashloaderx.parameter=value
```

where *x* is the FLASH identifier 0 or 1.

Table 4-11 FLASH loader parameters

Parameter	Description	Type	Values	Default
`fname`	Path to the host file used to initialize FLASH contents when the model starts. The file can be gzip compressed.	String	Valid filename	[empty string]
`fnameWrite`	Path to the host file used to save FLASH contents when the model exits.	String	Valid filename	[empty string]

4.11.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-14
- *Host bridge parameters* on page 4-16
- *Multimedia card parameters* on page 4-17
- *Terminal parameters* on page 4-18
- *VFS2 parameters* on page 4-19
- *Visualization parameters* on page 4-20.

4.12 Host bridge parameters

The table below lists the host bridge instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.hostbridge.parameter=value
```

Table 4-12 Host bridge parameters

Parameter	Description	Type	Values	Default
`interfaceName`	Host interface identifier	String	Valid string	`ARM0`

4.12.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-14
- *FLASH loader parameters* on page 4-15
- *Multimedia card parameters* on page 4-17
- *Terminal parameters* on page 4-18
- *VFS2 parameters* on page 4-19
- *Visualization parameters* on page 4-20
- *Fast Models User Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0370-/index.html`.

4.13 Multimedia card parameters

The table below lists the *MultiMedia Card* (MMC) instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

`motherboard.mmc.`*`parameter`*`=`*`value`*

Table 4-13 Multimedia card parameters

Parameter	Description	Type	Values	Default
`p_mmc_file`	File used for the MMC component backing store	String	Valid string	`mmc.dat`
`p_prodName`	Card ID product name	String	Six-character string	`ARMmmc`
`p_prodRev`	Card ID product revision	Integer	-	`0x1`
`p_manid`	Card ID manufacturer ID	Integer	-	`0x2`
`p_OEMid`	Card ID OEM ID	Integer	-	`0xCA4D0001`
`p_sernum`	Card serial number	Integer	-	`0xCA4D0001`

4.13.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-14
- *FLASH loader parameters* on page 4-15
- *Host bridge parameters* on page 4-16
- *Terminal parameters* on page 4-18
- *VFS2 parameters* on page 4-19
- *Visualization parameters* on page 4-20.

4.14 Terminal parameters

The table below lists the terminal instantiation-time parameters that you can change when the model is started.

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 0, 1, 2 or 3.

Table 4-14 Terminal parameters

Parameter	Description	Type	Values	Default
`mode`	Terminal initialization mode	String	`telnet`, `raw`	`telnet`
`start_telnet`	Enable terminal when the system starts	Boolean	`true` or `false`	`true`
`start_port`	Port used for the terminal when the system starts. If the specified port is not free, the port value is incremented by 1 until a free port is found.	Integer	Valid port number	5000

4.14.1 See also

Reference

- *Using a terminal with a system model* on page 3-17
- *Motherboard virtual component parameters* on page 4-14
- *FLASH loader parameters* on page 4-15
- *Host bridge parameters* on page 4-16
- *Multimedia card parameters* on page 4-17
- *VFS2 parameters* on page 4-19
- *Visualization parameters* on page 4-20.

4.15 VFS2 parameters

The table below lists the VFS2 instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.vfs2.parameter=value
```

Table 4-15 VFS2 parameters

Parameter	Description	Type	Values	Default
`mount`	Path to host folder to make accessible inside the model.	String	Valid path	[empty string]

4.15.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-14
- *FLASH loader parameters* on page 4-15
- *Host bridge parameters* on page 4-16
- *Multimedia card parameters* on page 4-17
- *Terminal parameters* on page 4-18
- *Visualization parameters* on page 4-20.

4.16 Visualization parameters

The table below lists the visualization instantiation-time parameters that you can change when the model is started.

The syntax to use in a configuration file or on the command line is:

```
motherboard.vis.parameter=value
```

Table 4-16 Visualization parameters

Parameter	Description	Type	Values	Default
`trap_key`	Trap key that works with **Left Ctrl** to toggle mouse display	Integer	-	`0x6B`
`rate_limit-enable`	Rate limit simulation	Boolean	`true` or `false`	`true`
`disable_visualisation`	Disable the VEVisualisation component on model startup	Boolean	`true` or `false`	`false`

4.16.1 See also

Reference

- *Motherboard virtual component parameters* on page 4-14
- *FLASH loader parameters* on page 4-15
- *Host bridge parameters* on page 4-16
- *Multimedia card parameters* on page 4-17
- *Terminal parameters* on page 4-18
- *VFS2 parameters* on page 4-19
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

4.17 FVP_VE_Cortex-A15MPx*n* CoreTile parameters

The table below lists the Cortex-A15 multiprocessor CoreTile parameters that you can change when you start any of the following models:

- FVP_VE_Cortex-A15MPx1
- FVP_VE_Cortex-A15MPx2
- FVP_VE_Cortex-A15MPx4.

All listed parameters are instantiation-time parameters. This CoreTile FVP is based on revision 2, patch 0 (r2p0) of the Cortex-A15 multiprocessor.

The syntax to use in a configuration file is:

```
cluster.parameter=value
```

Table 4-17 FVP_VE_Cortex-A15MPx*n* CoreTile parameters

Parameter	Description	Type	Allowed value	Default value
`CFGSDISABLE`	Disable some accesses to DIC registers	Boolean	`true` or `false`	`false`
`CLUSTER_ID`	Multiprocessor cluster ID value	Integer	0-15	0
`IMINLN`	Instruction cache minimum line size: false=32 bytes, true=64 bytes	Boolean	`true` or `false`	`true`
`PERIPHBASE`	Base address of peripheral memory space	Integer	-	`0x13080000`[a]
`dic-spi_count`	Number of shared peripheral interrupts implemented	Integer	0-224, in increments of 32	64
`internal_vgic`	Configures whether the model of the multiprocessor contains a *Virtual Generic Interrupt Controller* (VGIC)	Boolean	`true` or `false`	`true`
`l1_dcache-state_modelled`	Set whether L1 D-cache has stateful implementation	Boolean	`true` or `false`	`false`
`l1_icache-state_modelled`	Set whether L1 I-cache has stateful implementation	Boolean	`true` or `false`	`false`
`l2_cache-size`	Set L2 cache size in bytes	Integer	`0x080000`, `0x100000`, `0x200000`, `0x400000`.	`0x400000`
`l2_cache-state_modelled`	Set whether L2 cache has stateful implementation	Boolean	`true` or `false`	`false`
`l2-data-slice`	L2 data RAM slice	Integer	0, 1 or 2	0
`l2-tag-slice`	L2 tag RAM slice	Integer	0 or 1	0

a. If you are using the ARMCortexA15x*n*CT component on a VE model platform, this parameter is set automatically to `0x1F000000` and is not visible in the parameter list.

The FVP_VE_Cortex-A15MPx1 has the PERIPHBASE parameter set to `0x1F000000`, which is the base address of peripheral memory space on VE hardware.

The table below provides a description of the parameters for each Cortex-A15MP processor. These parameters are set individually for each Cortex-A15 processor you have in your system. Each processor has its own timer and watchdog.

The syntax to use in a configuration file is:

`cluster.cpu[n].parameter=value`

where *n* is the processor number, from 0 to 3 inclusive.

Table 4-18 FVP_VE_Cortex-A15MPx*n* CoreTile parameters - individual processors

Parameter	Description	Type	Allowed value	Default value
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true` or `false`	`false`
`CP15SDISABLE`	Initialize to disable access to some CP15 registers.	Boolean	`true` or `false`	`false`
`DBGROMADDR`	This value is used to initialize the CP15 **DBGDRAR** register. Bits[39:12] of this register specify the ROM table physical address.	Integer	`0x12000003`	`0x12000003`
`DBGROMADDRV`	If `true`, this sets bits[1:0] of the CP15 **DBGDRAR** to indicate that the address is valid.	Boolean	`true` or `false`	`true`
`DBGSELFADDR`	This value is used to initialize the CP15 **DBGDSAR** register. Bits[39:17] of this register specify the ROM table physical address.	Integer	`0x00010003`	`0x00010003`
`DBGSELFADDRV`	If `true`, this sets bits[1:0] of the CP15 **DBGDSAR** to indicate that the address is valid.	Boolean	`true` or `false`	`true`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true` or `false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true` or `false`	`false`
`ase-present`[a]	Set whether processor model has been built with NEON™ support.	Boolean	`true` or `false`	`true`
`min_sync_level`	Controls the minimum syncLevel by the CADI parameter interface.	Integer	0-3	0
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-cwd`	Virtual address of CWD.	String	-	-
`semihosting-enable`	Enable semihosting SVC traps.	Boolean	`true` or `false`	`true`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`

Table 4-18 FVP_VE_Cortex-A15MPx*n* CoreTile parameters - individual processors (continued)

Parameter	Description	Type	Allowed value	Default value
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`vfp-enable_at_reset`[b]	Enable coprocessor access and VFP at reset.	Boolean	`true` or `false`	`false`
`vfp-present`[a]	Set whether processor model has been built with VFP support.	Boolean	`true` or `false`	`true`

a. The ase-present and vfp-present parameters configure the synthesis options for the Cortex-A15 model. The options are:

vfp present and ase present
NEON and VFPv3-D32 supported.

vfp present and ase not present
VFPv3-D16 supported.

vfp not present and ase present
Illegal. Forces vfp-present to true so model has NEON and VFPv3-D32 support.

vfp not present and ase not present
Model has neither NEON nor VFPv3-D32 support.

b. This is a model specific behavior with no hardware equivalent.

4.17.1 See also

Reference

- *VE model parameters* on page 4-6
- *Motherboard peripheral parameters* on page 4-7
- *Motherboard virtual component parameters* on page 4-14
- *FVP_VE_Cortex-A9 CoreTile parameters* on page 4-24
- *FVP_VE_Cortex-R5_MPxn CoreTile parameters* on page 4-25
- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41.

4.18 FVP_VE_Cortex-A9 CoreTile parameters

The table below lists the Cortex-A9 MPCore parameters that you can change when you start the FVP_VE_Cortex-A9 model. This CoreTile FVP is based on r3p0 of the Cortex-A9 MPCore multiprocessor.

The table provides a description of the parameters for each Cortex-A9MP processor. These parameters are set individually for each Cortex-A9 processor you have in your system. Each processor has its own timer and watchdog.

The syntax to use in a configuration file is:

```
cluster.cpun.parameter=value
```

where *n* is the processor number, from 0 to 3 inclusive.

Table 4-19 FVP_VE_Cortex-A9_MPx*n* CoreTile parameters - individual processors

Parameter	Description	Type	Allowed value	Default value
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-cwd`	Virtual address of CWD.	String	-	-
`semihosting-enable`	Enable semihosting SVC traps.	Boolean	`true` or `false`	`true`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`

4.18.1 See also

Reference

- *VE model parameters* on page 4-6
- *Motherboard peripheral parameters* on page 4-7
- *Motherboard virtual component parameters* on page 4-14
- *FVP_VE_Cortex-A15MPxn CoreTile parameters* on page 4-21
- *FVP_VE_Cortex-R5_MPxn CoreTile parameters* on page 4-25
- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41.

4.19 FVP_VE_Cortex-R5_MPx*n* CoreTile parameters

The table below lists the parameters for the following models:

- FVP_VE_Cortex-R5_MPx1
- FVP_VE_Cortex-R5_MPx2.

These parameters are set once, irrespective of the number of Cortex-R5 processors in your system. If you have multiple Cortex-R5 processors, then each processor has its own parameters.

Table 4-20 FVP_VE_CortexR5_MPx*n* CoreTile parameters

Parameter	Description	Type	Allowed value	Default value
`GROUP_ID`	Value read in GROUP ID register field, bits[15:8] of the MPIDR.	Integer	0-15	0
`INST_ENDIAN`	Controls whether the model supports the instruction endianness bit. You can find more information in the *ARM Architecture Reference Manuals*.	Boolean	`true`/`false`	`true`
`LOCK_STEP`	Affects dual-processor configurations only, and ignored by single-processor configurations.	Integer	0 - Disable. Set for two independent processors. 1 - Lock Step. Appears to the system as two processors but is internally modeled as a single processor. 3 - Split Lock. Appears to the system as two processors but can be statically configured from reset either as two independent processors or two locked processors. For the model, these are equivalent to Disable and Lock Step, respectively, except for the value of build options registers. The model does not support dynamically splitting and locking the processor.	0
`MICRO_SCU`	Controls whether the effects of the MicroSCU are modeled. You can find more information in the *Cortex-R5 Technical Reference Manual*.	Boolean	`true`/`false`	`true`
`NUM_BREAKPOINTS`	Controls with how many breakpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.	Integer	2-8	3

Table 4-20 FVP_VE_CortexR5_MPx*n* CoreTile parameters (continued)

Parameter	Description	Type	Allowed value	Default value
`NUM_WATCHPOINTS`	Controls with how many watchpoint pairs the model has been configured. This only affects the build options registers, because debug is not modeled.	Integer	1-8	2
`dcache-state_modelled`	Set whether D-cache has stateful implementation.	Boolean	`true`/`false`	`false`
`icache-state_modelled`	Set whether I-cache has stateful implementation.	Boolean	`true`/`false`	`false`

The table below provides a description of the parameters for each FVP_VE_Cortex-R5_MPx1 component processor. These parameters are set individually for each processor you have in your system.

Table 4-21 FVP_VE_CortexR5_MPx*n* CoreTile parameters - individual processors

Parameter	Description	Type	Allowed value	Default value
`CFGATCMSZ`	Sets the size of the ATCM.	Integer	`0x00000000 - 0xE`	`0xE`
`CFGBTCMSZ`	Sets the size of the BTCM.	Integer	`0x00000000 - 0xE`	`0xE`
`CFGEND`	Initialize to BE8 endianness.	Boolean	`true`/`false`	`false`
`CFGIE`	Set the reset value of the instruction endian bit.	Boolean	`true`/`false`	`false`
`CFGNMFI`	Enable nonmaskable FIQ interrupts on startup.	Boolean	`true`/`false`	`false`
`DP_FLOAT`	Sets whether double-precision instructions are available. You can find more information in the *ARM Architecture Reference Manuals*.	Boolean	`true`/`false`. If `true`, then double precision VFP is supported. If `false`, then the VFP is single precision only.	`true`
`NUM_MPU_REGION`	Sets the number of MPU regions.	Integer	`0x00`, `0xC`, `0x10`. `0` = no MPU.	`0xC`
`TEINIT`	T32 exception enable. The default has exceptions including reset handled in A32 state.	Boolean	`true`/`false`	`false`
`VINITHI`	Initialize with high vectors enabled.	Boolean	`true`/`false`	`false`

Table 4-21 FVP_VE_CortexR5_MPx*n* CoreTile parameters - individual processors (continued)

Parameter	Description	Type	Allowed value	Default value
`atcm_base`[a]	Model-specific. Sets the base address of the ATCM. You can find more information on the processor configuration signals in the *Cortex-R5 Technical Reference Manual*.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x40000000`
`btcm_base`[a]	Model-specific. Sets the base address of the BTCM. You can find more information on the processor configuration signals in the *Cortex-R5 Technical Reference Manual*.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x00000000`
`dcache-size`	Set D-cache size in bytes.	Integer	`0x4000 - 0x10000`	`0x10000`
`icache-size`	Set I-cache size in bytes.	Integer	`0x4000 - 0x10000`	`0x10000`
`semihosting-ARM_SVC`	A32 SVC number for semihosting.	Integer	`0x000000 - 0xFFFFFF`	`0x123456`
`semihosting-cmd_line`	Command line available to semihosting SVC calls.	String	No limit except memory	[Empty string]
`semihosting-cwd`	Virtual address of CWD.	String	-	-
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to `false`.	Boolean	`true/false`	`true`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10000000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0F000000`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	`0x00 - 0xFF`	`0xAB`
`vfp-enable_at_reset`[a]	Enable coprocessor access and VFP at reset.	Boolean	`true/false`	`false`
`vfp-present`	Set whether model has VFP support.	Boolean	`true/false`	`true`

a. This is a model-specific behavior with no hardware equivalent.

4.19.1 See also

Reference

- *VE model parameters* on page 4-6
- *Motherboard peripheral parameters* on page 4-7
- *Motherboard virtual component parameters* on page 4-14
- *FVP_VE_Cortex-A15MPxn CoreTile parameters* on page 4-21
- *FVP_VE_Cortex-A9 CoreTile parameters* on page 4-24
- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41
- *Cortex-R5 Technical Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0460-/index.html`
- *ARM Architecture Reference Manuals*, `http://infocenter.arm.com/help/topic/com.arm.doc.set.architecture/index.html`.

4.20 ARMv7-A AEM parameters

The ARMv7-A *Architecture Envelope Model* (AEM) parameters adjust the behavior of external platform components on the *Versatile™ Express* (VE) system board.

- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv7-A AEM debug architecture parameters* on page 4-39
- *ARMv7-A AEM message parameters* on page 4-40
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.20.1 See also

Reference

- *VE model parameters* on page 4-6
- *Motherboard peripheral parameters* on page 4-7
- *Motherboard virtual component parameters* on page 4-14
- *FVP_VE_Cortex-A15MPxn CoreTile parameters* on page 4-21
- *FVP_VE_Cortex-A9 CoreTile parameters* on page 4-24
- *FVP_VE_Cortex-R5_MPxn CoreTile parameters* on page 4-25.

4.21 ARMv7-A AEM general multiprocessor parameters

You can adjust the overall behavior of the models with the multiprocessor parameters.

Table 4-22 multiprocessor parameters

Parameter	Description	Default
`auxilliary_feature_register0`	Value for AFR0 ID Register.	0
`cpuID`	Value for Main CPU ID Register.	`0x411fc081`
`dic-spi_count`	Number of shared peripheral interrupts implemented.	64
`dtcm0_base`	DTCM base address at reset.	0
`dtcm0_enable`	Enable DTCM at reset.	`false`
`dtcm0_size`	DTCM size in KB.	32
`FILTEREN`	Enable filtering of accesses between master bus ports. This is usually not used inside a VE system and should be left `false`.	`false`
`FILTEREND`	End of region filtered to `pvbus_m1`. Values must be aligned to a 1MB boundary.	0
`FILTERSTART`	Start of region filtered to `pvbus_m1`. Values must be aligned to a 1MB boundary.	0
`implements_ple_like_a8`	Add support for the PLE from a Cortex-A8 processor.	`false`
`IS_VALIDATION`[a]	Reserved. Enable A9-validation-like trickbox-coprocessor, which is only usable in validation platform model.	`false`
`itcm0_base`	ITCM base address at reset.	`0x40000000`
`itcm0_enable`	Enable ITCM at reset.	`false`
`itcm0_size`	ITCM size in KB.	32
`PERIPHBASE`[b]	Base address of MP "private" peripherals (WatchdogTimers, GIC) (bits 31:13 used).	`0x13080000`
`siliconID`	Value for Auxiliary ID Register.	`0x41000000`
`CFGSDISABLE`	Disable access to some registers in the internal interrupt controller peripheral.	`false`
`implements_lpae`	Implement the Large Physical Address Extension in this multiprocessor.	`false`
`implements_virtualization`	Implement the Virtualization extension in this multiprocessor. When set, this also enables LPAE.	`false`
`use_Cortex-A15_peripherals`	Change the layout of the internal peripheral memory map to mimic that of the Cortex-A15 multiprocessor.	`false`
`delayed_CP15_operations`	Delay the functional effect of CP15 operations.	`false`
`take_ccfail_undef`	Take undefined exceptions even if the instruction failed its condition codes check.	`false`
`low_latency_mode`	Run only a single instruction between checks for IRQ and other events. This ensures that when the platform raises an interrupt, the exception vector is taken immediately, but it involves a considerable penalty in performance.	`false`

a. IS_VALIDATION is not exposed in the VE platform model, and fixed as `false`.

b. PERIPHBASE is not exposed in the VE platform model, and fixed as `0x2C000000`.

4.21.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv7-A AEM debug architecture parameters* on page 4-39
- *ARMv7-A AEM message parameters* on page 4-40
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.22 ARMv7-A AEM multiprocessor parameters

You can configure this model as a multiprocessor, so there are separate groups of parameters for each processor in the system. In cases where fewer processors than the maximum number possible are instantiated, the parameters from `cpu0` are always used first.

Table 4-23 Multiprocessing parameters

Parameter	Description	Default
`cluster_id`	Value for Cluster ID that is available to target programs in MPIDR.	0
`multiprocessor_extensions`	Enable the instruction set changes introduced with the ARMv7 Multiprocessor Extensions.	`true`
`num_cores`	Number of processors implemented. To instantiate more than one processor, set parameter `multiprocessor_extensions`.	1
`vmsa.cachetlb_broadcast`	Enable broadcasting of cache and TLB maintenance operations that apply to the inner shared domain.	`true`

4.22.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv7-A AEM debug architecture parameters* on page 4-39
- *ARMv7-A AEM message parameters* on page 4-40
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.23 ARMv7-A AEM processor parameters

These parameters are repeated in groups `cpu0-cpu3`, one for each processor in the multiprocessor.

Table 4-24 Processor parameters

Parameter	Description	Default
`cpu[n].CFGEND0`	Starts the processor in big endian BE8 mode.	`false`
`cpu[n].CFGNMFI`	Sets the NMFI bit in the *System Control Register* (SCTLR) that prevents the FIQ interrupt from being masked in APSR.	`false`
`cpu[n].CFGTE`	Starts the processor in T32 mode.	`false`
`cpu[n].CP15SDISABLE`	Disables access to some CP15 registers.	`false`
`cpu[n].VINITHI`	Starts with high vectors enabled, the vector base address is `0xFFFF0000`.	`false`
`cpu[n].implements_neon`	Support NEON in this processor.	`true`
`cpu[n].implements_thumbEE`	Support T32EE in this processor.	`true`
`cpu[n].implements_trustzone`	Support TrustZone™ in this processor.	`true`
`cpu[n].implements_vfp`	Support VFP in this processor.	`true`
`cpu[n].fpsID`	Value for Floating-point System ID Register.	`0x41033091`
`cpu[n].implements_vfpd16-d31`	If VFP is implemented, support 32 double-precision registers. Otherwise 16 are supported. If NEON is implemented, 32 registers are always supported and this parameter is ignored.	`true`
`cpu[n].implements_vfp_short_vectors`	Enable support for vfp short vector operations, as indicated by `MVFR0[27:24]`.	`true`
`cpu[n].implements_fused_mac`	Implement the vfp fused multiply accumulate operations.	`false`
`cpu[n].implements_sdiv_udiv`	Implement the integer divide operations.	`false`
`cpu[n].vfp-enable_at_reset`	VFP registers are enabled without a requirement to write the corresponding access enable bits first.	`false`
`cpu[n].SMPnAMP`	Place this processor inside the inner shared domain, and participate in the coherency protocol that arranges inner cache coherency among other processors in the domain.	`false`
`cpu[n].use_IR`	Enable operation reordering in conjunction with `delayed_read_buffer`.	0

4.23.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv7-A AEM debug architecture parameters* on page 4-39
- *ARMv7-A AEM message parameters* on page 4-40
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.24 ARMv7-A AEM memory parameters

You can adjust the memory configuration of the multiprocessor with memory parameters.

Table 4-25 Memory parameters

Parameter	Description	Default
`vmsa.implements_fcse`	Support `fcse` in this multiprocessor.	`false`
`vmsa.infinite_write_buffer`	Enable infinite write-buffer.	`false`
`vmsa.write_buffer_delay`	Elapsed time between natural buffer drains.	1000
`vmsa.delayed_read_buffer`	Enable deferred read values in conjunction with `use_IR`.	`false`
`vmsa.cache_incoherence_check`	Enable the check for cache incoherence.	`false`
`vmsa.memory_marking_check`	Enable the check for inconsistent memory marking in the TLB.	`false`
`vmsa.instruction_tlb_lockable_entries`	Number of lockable entries in instruction TLB.	32
`vmsa.instruction_tlb_size`	Total number of entries in instruction TLB.	32
`vmsa.main_tlb_lockable_entries`	Number of lockable entries in data or unified TLB.	32
`vmsa.main_tlb_size`	Total number of entries in data or unified TLB.	32
`vmsa.separate_tlbs`	Separate ITLB and DTLB. If the TLB is unified, its size is defined by parameter `vmsa.main_tlb_size`.	`true`
`vmsa.tlb_prefetch`	Enables aggressive pre-fetching into the TLB.	`false`
`vmsa.implements_outer_shareable`	Distinguish between inner shareable and outer shareable memory access types. Outer shareable is implemented as Non Cacheable.	`true`
`vmsa.access_flags_hardware_management`	Enable support for the hardware management of the Access Flag in the pagetables.	`true`
`dcache-state_modelled`	Allow line allocation in D-side caches at all levels.	`true`
`icache-state_modelled`	Allow line allocation in I-side caches at all levels. Unified caches allocate lines only if these parameters are enabled at both I-side and D-side.	`true`

The `[d|i]cache-state_modelled` parameters control the way that caches are simulated. When switched on, the default mode, all cache behaviors and maintenance operations are modeled fully.

If `false`, the cache is still present in the programmer's view of the multiprocessor but in the simulated implementation there are no memory lines associated with the cache at this level. The programmer-view effect of this is as though the cache cleans and invalidates any line as soon as it is loaded, and can never become incoherent with its backing memory. Although this is an architecturally legal behavior, it is not realistic to any current hardware and is less likely to expose problems in target software. It can, however, be useful when debugging problems that are suspected to be related to cache maintenance, and also has the side effect of permitting the model to run faster.

Compare this to the effect of setting `cpu[n].l2dcache-size_bytes = 0`, which is to simulate a processor that contains only Level 1 caches. In this case, the ID code registers do not describe a Level 2 cache. Level 2 is entirely absent from the processor.

4.24.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv7-A AEM debug architecture parameters* on page 4-39
- *ARMv7-A AEM message parameters* on page 4-40
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.25 ARMv7-A AEM cache geometry parameters

You can configure the multiprocessor with up to four levels of cache. The cache layout is not required to be symmetrical for each processor in the multiprocessor. The cache geometry parameters repeat in groups `cpu0-cpu3`, corresponding to the view, for each processor, of the memory hierarchy.

Table 4-26 General cache parameters

Parameter	Description	Default
`cpu[n].cache-coherency_level`	1-based Level of cache coherency. A value of 2 means that the L2 caches, and all subsequent levels, are coherent.	2
`cpu[n].cache-unification_level`	1-based Level of cache unification. A value of 2 means that the L2 caches, and all subsequent levels, are unified.	2
`cpu[n].cache-outer_level`	Level at which outer cache attributes start to be used. L1 caches always use inner attributes. A value of 2 means that the L2 caches, and all subsequent levels, use outer attributes.	2

Each cache block in the system is configured using the cache block parameters, which repeat for groups `cpu0-cpu3`, and within each group in caches `l1icache`, `l1dcache-l4icache`, `l4dcache`.

The number and type of cache blocks are active depending on the unification level of each processor. Before the unification level, caches are separate on the instruction and data sides, and both sets of parameters are used. After the unification level, the data and instruction sides are unified, and the single cache block is described using the data side parameters only.

Table 4-27 Cache block parameters

Parameter	Description	Default
`cpu[n].[cache]-size_bytes`	Zero if the cache is not present, otherwise the total size in bytes of the cache. Must be divisible by the line length and associativity, and represent a number of cache sets not greater than 32768.	32768
`cpu[n].[cache]-linelength_bytes`	Length of each cache line. Must be 32 or 64.	32
`cpu[n].[cache]-associativity`	Associativity of this cache. Must be between 1 and 1024.	4
`cpu[n].[cache]-read_allocate`	Support allocate-on-read in this cache.	`true`
`cpu[n].[cache]-write_allocate`[a]	Support allocate-on-write in this cache.	`true`
`cpu[n].[cache]-write_back`[a]	Support write-back in this cache.	`true`
`cpu[n].[cache]-write_through`[a]	Support write-through in this cache.	`true`
`cpu[n].[cache]-treat_invalidate_as_clean`[a]	Always clean dirty cache lines before invalidating them.	`false`
`cpu[n].[cache]-shared_key`	If nonzero, mark this cache as being shared with other processors.	0

a. This parameter is not applicable to instruction-side caches.

The parameters for each processor describe the view for that processor of the memory hierarchy. If more than one processor has access to the same cache unit, for example, a shared Level 2 cache, then:

- the cache must be described with all the same parameter settings in every case.

- all caches downstream of a shared cache must also be shared, and in the same order for every observer.
- the `[cache]-shared_key` parameter is set to an arbitrary nonzero value. Any cache in the system that has this value is considered to be one cache block.

You can describe nonlegal cache layouts using the `shared_key` mechanism. Not all bad cases can be easily detected during initialization, so take care to ensure correct cache configuration. The model might behave erratically if the cache layout cannot be rationalized.

The following figures show examples of processor-cache architecture configurations:

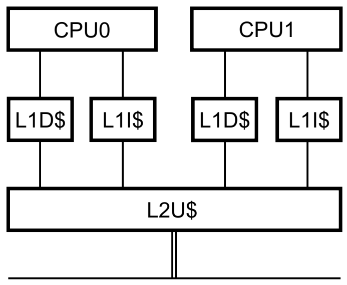


Figure 4-1 Processor-cache architecture configuration - Example 1

```
cpu0.cache-unification_level=2
cpu0.l2dcache-size_bytes=32768
cpu0.l2dcache-shared_key=1
cpu1.cache-unification_level=2
cpu1.l2dcache-size_bytes=32768
cpu1.l2dcache-shared_key=1
```

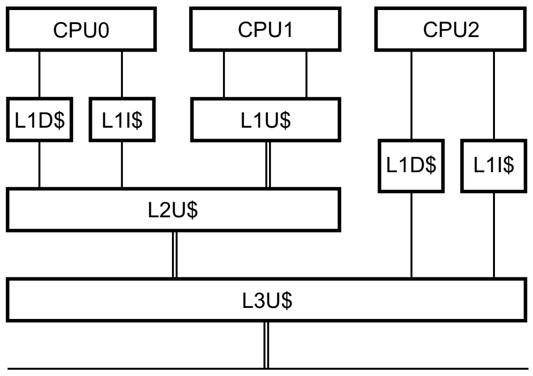


Figure 4-2 Processor-cache architecture configuration - Example 2

```
cpu0.cache-unification_level=2
cpu0.l2dcache-size_bytes=32768
cpu0.l2dcache-shared_key=1
cpu0.l3dcache-size_bytes=65536
cpu0.l3dcache-shared_key=2
cpu1.cache-unification_level=1
cpu1.l2dcache-size_bytes=32768
cpu1.l2dcache-shared_key=1
cpu1.l3dcache-size_bytes=65536
cpu1.l3dcache-shared_key=2
cpu2.cache-unification_level=2
cpu2.l2dcache-size_bytes=65536
cpu2.l2dcache-shared_key=2
```

Note

In the view of CPU2, the shared cache block marked L3U$ is at Level 2 in the memory system hierarchy.

4.25.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM debug architecture parameters* on page 4-39
- *ARMv7-A AEM message parameters* on page 4-40
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.26 ARMv7-A AEM debug architecture parameters

The ARMv7 Debug architecture contains a number of optional features. The debug architecture parameters control which of these features are implemented by the model.

Table 4-28 Debug architecture parameters

Parameter	Description	Default
`implements_OSSaveAndRestore`	Add support for the OS Save and Restore mechanism implemented by DBGOSSRR and other registers.	`true`
`DBGOSLOCKINIT`	Initial value for the Locked bit in DBGOSLSR. When this bit is set, software access to debug registers is restricted.	`0x1`
`implements_secure_user_halting_debug`	Permit debug events in Secure User mode when invasive debug is not permitted in Secure privileged modes. (Deprecated in ARM v7.)	`false`
`DBGPID`	Value for CP14 DBGPID registers.	`0x8000bb000`
`DBGCID`	Value for CP14 DBGCID registers.	`0x0`
`DBGDSCCR_mask`	Implemented bits of DBGDSCCR.	`0x7`
`cpu[`*n*`].DBGDRAR`	Value for Debug ROM address register.	`0x0`
`cpu[`*n*`].DBGSRAR`	Value for Debug Self address register.	`0x0`

4.26.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv7-A AEM message parameters* on page 4-40
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.27 ARMv7-A AEM message parameters

The message severity parameters control how warning and error messages from the architectural checkers are generated.

Table 4-29 Message severity levels

Parameter	Description	Default
`messages.break_warning_level`	The simulation stops in the debugger after emitting a message at this level or higher.	5
`messages.ignore_warning_level`	Messages below this level are ignored and not printed.	1
`messages.suppress_repeated_messages`	The simulation does not emit more than one copy of a message when it is generated from a given point in the target program.	`true`
`messages.output_file`	The path[a] of the file to which messages are written. If blank, messages are sent to `stderr`.	Blank

a. The format of the string follows the normal file path conventions for the host platform. File paths without a leading root are written into the current working directory, which might vary.

Except for fatal errors, the severity level of each message can be reconfigured in parameters `messages.severity_level_[*]`, enabling you to concentrate only on those warnings that are appropriate to your task. See the following table:

Table 4-30 Message parameters

Level	Name	Description
0	Minor Warning	Suspect, but plausibly correct
1	Warning	A likely bug
2	Severe Warning	Technically legal, but believed certain to be a bug
3	Error	A definite architectural violation
4	Severe Error	Target code unlikely to be able to recover
5	Fatal	From which the simulation is unable to continue

4.27.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv7-A AEM debug architecture parameters* on page 4-39
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.28 ARMv8-A AEM parameters

The ARMv8-A *Architecture Envelope Model* (AEM) parameters adjust the behavior of external platform components on the *Versatile™ Express* (VE) system board. The following topics describe these parameters:

- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARMv8-A AEM general processor parameters* on page 4-46
- *ARMv8-A AEM general cache parameters* on page 4-48
- *ARMv8-A AEM memory parameters* on page 4-51
- *ARMv8-A AEM debug architecture parameters* on page 4-52
- *ARMv8-A AEM message parameters* on page 4-53
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.28.1 See also

Reference

- *VE model parameters* on page 4-6
- *Motherboard peripheral parameters* on page 4-7
- *Motherboard virtual component parameters* on page 4-14
- *ARMv8 Instruction Set Overview*,
  `http://infocenter.arm.com/help/topic/com.arm.doc.genc010197a/index.html`.

4.29 ARMv8-A AEM general multiprocessor parameters

You can configure the overall behavior of the models with the general multiprocessor parameters.

Table 4-31 General multiprocessor parameters[a]

Parameter	Description	Type	Range	Default
`NUM_CORES`	Number of processors implemented.	`int`	`0x1-0x4`	`0x1`
`is_uniprocessor`	`true` for a uniprocessor implementation. When `true`, `NUM_CORES` must be `0x1`.	`bool`	`false-true`	`false`
`PA_SIZE`	Physical address size, in bits.	`int`	`0x0-0x30`	`0x28`
`has_16bit_asids`	Enable 16-bit *Address Space IDentifiers* (ASIDs).	`bool`	`false-true`	`true`
`auxilliary_feature_register0`	Value for *Auxiliary Feature Register 0* (ID_AFR0).	`int`	`0x0-0xFFFFFFFF`	`0x0`
`MIDR`	Value for *Main ID Register* (MIDR).	`int`	`0x0-0xFFFFFFFF`	`0x410FD0F0`
`clear_reg_top_eret`	Clear top 32 bits of general purpose registers on exception return. `0x0` = preserve, `0x1` = clear to zero, `0x2` = random choice of preserve or clear to zero.	`int`	`0x0-0x2`	`0x1`
`mixed_endian`	Enable processor to change endianness at runtime. `0x0` = not supported, `0x1` = supported at all exception levels, `0x2` = supported at EL0 only.	`int`	`0x0-0x2`	`0x1`
`take_ccfail_undef`	In AArch32, take Undefined Instruction exception even if the instruction fails its condition-codes check.	`bool`	`false-true`	`true`
`has_thumb2ee`	Enable T32EE support.[b]	`bool`	`false-true`	`false`
`t32ee_bx_to_arm`	Behavior when T32EE attempts to *Branch and Exchange* (`BX`) to A32. `0x0` = stay in T32 state, `0x1` = change to A32 state, `0x2` = handle as Illegal Exception return.	`int`	`0x0-0x2`	`0x0`
`has_el2`	Enable EL2.	`bool`	`false-true`	`true`
`has_el3`	Enable EL3.	`bool`	`false-true`	`true`
`max_32bit_el`	Maximum exception level supporting AArch32 modes. `-0x1` means no support.	`int`	`-0x1-0x3`	`0x3`
`el0_el1_only_non_secure`	Controls security state of EL0 and EL1 if neither EL2 nor EL3 are implemented. `true` means non-secure.	`bool`	`false-true`	`false`
`has_writebuffer`	Implement write access buffering before L1 cache. May affect ext_abort behavior.	`bool`	`false-true`	`false`
`has_delayed_sysreg`	Delay the functional effect of system register writes until ISB or implicit barrier.	`bool`	`false-true`	`false`

Table 4-31 General multiprocessor parameters[a] (continued)

Parameter	Description	Type	Range	Default
`tidcp_traps_el0_undef_imp_def`	The TIDCP bit traps, in EL0, undefined IMPLEMENTATION DEFINED instructions accessing coprocessor registers.	`bool`	`false-true`	`true`
`unpredictable_hvc_behaviour`	Define `HVC` UNPREDICTABLE behavior in HYP mode, EL2, when the SCR.HCE bit is clear. `0x0` = Undefined Instruction, `0x1` = `NOP` instruction.	`int`	`0x0-0x1`	`0x0`
`unpredictable_smc_behaviour`	Define `SMC` UNPREDICTABLE behavior in Secure mode, EL3, when the SCR.SCD bit is clear. `0x0` = Undefined Instruction, `0x1` = `NOP` instruction.	`int`	`0x0-0x1`	`0x0`
`register_reset_data`	Fill data for register bits when they become UNKNOWN at reset.	`int`	-	`0x0`
`scramble_unknowns_at_reset`	Fill in UNKNOWN bits in registers at reset with register_reset_data.	`bool`	`false-true`	`true`
`apsr_read_restrict`	At EL0, UNKNOWN bits of APSR are RAZ.	`bool`	`false-true`	`false`
`warn_unpredictable_in_v7`	Warn of UNPREDICTABLE behavior in ARMv7.	`bool`	`false-true`	`false`
`exercise_stxr_fail`	When true, return a pseudorandom majority of *Store Exclusive Register* (`STXR`) instructions as Failed.	`bool`	`false-true`	`false`
`delay_serror`	Minimum propagation delay of the *System Error* (SERR) signal into the multiprocessor.[c]	`int`	`0x0-0xFFFFFFFF`	`0x0`
`has_eagle_cp15_registers`	In AArch32 state, enable Cortex™-A15 processor set of IMPLEMENTATION DEFINED registers in CP15.	`bool`	`false-true`	`true`

a. Terms such as `cluster` may replace `cpu` on some systems. The parameter `PERIPHBASE` is locked down in the VE FVP.

b. ARM deprecates this optional feature.

c. Accurate in low-latency mode (`-C cpu.scheduler_mode=1`), but otherwise any delay might be larger.

4.29.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM abort parameters* on page 4-44
- *ARMv8-A AEM GIC parameters* on page 4-45
- *ARMv8-A AEM general processor parameters* on page 4-46
- *ARMv8-A AEM general cache parameters* on page 4-48
- *ARMv8-A AEM memory parameters* on page 4-51
- *ARMv8-A AEM debug architecture parameters* on page 4-52
- *ARMv8-A AEM message parameters* on page 4-53
- *ARMv8-A AEM simulator parameters* on page 4-54
- *Boundary features and architectural checkers* on page 4-56
- *IMPLEMENTATION DEFINED features* on page 4-57.

4.30 ARMv8-A AEM abort parameters

You can configure the abort behavior of the models with the abort parameters.

Table 4-32 Abort parameters

Parameter	Description	Type	Range	Default
`abort_execution_from_device_memory`	Abort on execution from device memory.	`bool`	`false-true`	`false`
`ext_abort_normal_cacheable_read_is_sync`	Synchronous reporting of normal cacheable-read external aborts.	`bool`	`false-true`	`true`
`ext_abort_normal_noncacheable_read_is_sync`	Synchronous reporting of normal noncacheable-read external aborts.	`bool`	`false-true`	`true`
`ext_abort_device_read_is_sync`	Synchronous reporting of device read external aborts.	`bool`	`false-true`	`true`
`ext_abort_so_read_is_sync`	Synchronous reporting of strongly ordered read external aborts.	`bool`	`false-true`	`true`
`ext_abort_normal_cacheable_write_is_sync`	Synchronous reporting of normal cacheable write external aborts.	`bool`	`false-true`	`false`
`ext_abort_normal_noncacheable_write_is_sync`	Synchronous reporting of normal noncacheable write external aborts.	`bool`	`false-true`	`false`
`ext_abort_device_write_is_sync`	Synchronous reporting of device write external aborts.	`bool`	`false-true`	`false`
`ext_abort_so_write_is_sync`	Synchronous reporting of strongly ordered write external aborts.	`bool`	`false-true`	`true`
`ext_abort_ttw_cacheable_read_is_sync`	Synchronous reporting of TTW cacheable read external aborts.	`bool`	`false-true`	`true`
`ext_abort_ttw_noncacheable_read_is_sync`	Synchronous reporting of TTW noncacheable read external aborts.	`bool`	`false-true`	`true`
`ext_abort_prefetch_is_sync`	Synchronous reporting of instruction fetch external aborts.	`bool`	`false-true`	`true`
`ext_abort_fill_data`	Returned data, if external aborts are asynchronous.	`int`	-	`0xFDFDFDFCFCFDFDFD`
`unpredictable_exclusive_abort_memtype`	MMU abort if exclusive access is not supported. 0 = none, exclusives allowed in all memory, 1 = exclusives abort in Device memory, 2 = exclusives abort in any memory type other than WB inner cacheable.	`int`	`0x0-0x2`	`0x0`

4.30.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42.

4.31 ARMv8-A AEM GIC parameters

You can configure the *Generic Interrupt Controller* (GIC) behavior of the models with the GIC parameters.

Table 4-33 GIC parameters

Parameter	Description	Type	Range	Default
`dic-spi_count`	Number of *Shared Peripheral Interrupts* (SPIs) supported.	`int`	`0x0-0xE0`	`0x40`
`non_secure_vgic_alias_when_ns_only`	If no EL3 and no Secure state, the VGIC has a Secure alias. If this parameter is nonzero, the model forms a Non-secure alias from its value for the VGIC, aligned to 32KiB.	`int`	`0x0-0xFFFFFFFFFFFF`	`0x0`
`internal_vgic`	Enable VGIC peripheral.[a]	`bool`	`false-true`	`true`
`gicv3_cpu_interface`	Enable GICv3 processor interface in each processor model.[b]	`bool`	`false-true`	`false`
`gicv3.STATUSR-implemented`	If GICv3 processor interface enabled, enable STATUS registers.	`bool`	`false-true`	`true`
`gicv3.IIDR_base`	Base value for calculating GICC_IIDR value.	`int`	`0x0-0xFFFFFFFF`	`0x43B`
`gicv3.BPR-min`	Minimum value for GICC_BPR.[c]	`int`	`0x0-0x3`	`0x2`

a. Enable unless a shared VGIC is present.
b. Disable unless a GICv3 distributor is present.
c. Non-secure copy will be this value + 1.

4.31.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARM Generic Interrupt Controller Architecture Specification*, `http://infocenter.arm.com/help/topic/com.arm.doc.ihi0048-/index.html`.

4.32 ARMv8-A AEM general processor parameters

Each processor in the multiprocessor has its own parameters. The models use the parameters for processors in sequence, from `cpu0` onwards. In cases where fewer processors than the maximum number are instantiated, any parameters for uninstantiated processors are ignored.

Table 4-34 General processor parameters

Parameter	Description	Type	Range	Default
`cpu[n].CONFIG64`	Enable AArch64.	`bool`	`false-true`	`true`
`cpu[n].POWERCTLI`	Default power control state.	`int`	`0x0-0xFFFFFFFF`	`0x0`
`cpu[n].SMPnAMP`	This processor is in the inner shared domain, and uses its cache coherency protocol.	`bool`	`false-true`	`true`
`cpu[n].CFGEND`	Use big-endian order.	`bool`	`false-true`	`false`
`cpu[n].CP15SDISABLE`	Disable access to some CP15 registers.	`bool`	`false-true`	`false`
`cpu[n].ase-present`	Enable NEON™.	`bool`	`false-true`	`true`
`cpu[n].VINITHI`	Enable high vectors. Base address `0xFFFF0000`.	`bool`	`false-true`	`false`
`cpu[n].RVBAR`	Reset Vector Base Address when resetting into AArch64.	`int`	`0x0-0xFFFFFFFFFFFC`	`0x0`
`cpu[n].vfp-present`	Enable floating-point arithmetic.	`bool`	`false-true`	`true`
`cpu[n].vfp-enable_at_reset`	Enable coprocessor access and VFP at reset.[a]	`bool`	`false-true`	`false`
`cpu[n].vfp-traps`	Enable hardware trapping of VFP exceptions for VFPv4U.	`bool`	`false-true`	`true`
`cpu[n].force-fpsid`	Override the FPSID value.	`bool`	`false-true`	`false`
`cpu[n].force-fpsid-value`	Value for the overridden FPSID.	`int`	`0x0-0xFFFFFFFF`	`0x0`
`cpu[n].TEINIT`	Controls the initial state of SCTLR.TE in AArch32. When set, causes AArch32 exceptions (including reset) to be taken in T32 mode.	`bool`	`false-true`	`false`
`cpu[n].etm-present`	Enable *Embedded Trace Macrocell* (ETM).	`bool`	`false-true`	`true`
`cpu[n].min_sync_level`	Minimum CADI syncLevel. 0 = off, 1 = syncState, 2 = postInsnIO, 3 = postInsnAll.	`int`	`0x0-0x3`	`0x0`

a. This is a model-specific behavior with no hardware equivalent.

4.32.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARMv8-A AEM cryptography parameters* on page 4-47
- *ARMv8-A AEM general cache parameters* on page 4-48
- *ARMv8-A AEM memory parameters* on page 4-51
- *ARMv8-A AEM debug architecture parameters* on page 4-52
- *Semihosting parameters* on page 4-55.

4.33 ARMv8-A AEM cryptography parameters

You can configure the cryptographic behavior of the processors in the models with the cryptography parameters.

Table 4-35 Cryptography parameters

Parameter	Description	Type	Range	Default
`cpu[n].crypto_aes`	AES hash level. 0 = AES-128, 1 = AES-192, 2 = AES-256.	`int`	`0x0-0x2`	`0x2`
`cpu[n].crypto_sha1`	Enable SHA1.	`int`	`0x0-0x1`	`0x1`
`cpu[n].crypto_sha256`	Enable SHA256.	`int`	`0x0-0x1`	`0x1`

4.33.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general processor parameters* on page 4-46.

4.34 ARMv8-A AEM general cache parameters

You can configure the caches of the multiprocessor with the general cache parameters.

Table 4-36 General cache parameters

Parameter	Description	Type	Range	Default
`cache_maintenance_hits_watchpoints`	Enable AArch32 cache maintenance by `DCIMVAC` to trigger watchpoints.[a]	`bool`	`false-true`	`false`
`dcache-state_modelled`	Stateful implementation, with line allocation in D-caches at all levels.[b]	`bool`	`false-true`	`true`
`icache-state_modelled`	Stateful implementation, with line allocation in I-caches at all levels.[b]	`bool`	`false-true`	`true`
`memory.l2_cache.is_inner_cacheable`	L2 cache is inner cacheable, not outer cacheable.	`bool`	`false-true`	`true`
`memory.l2_cache.is_inner_shareable`	L2 cache is inner shareable, not outer shareable.	`bool`	`false-true`	`true`
`cache-log2linelen`	Log_2(cache-line length, in bytes)	`int`	`0x4-0x8`	`0x6`
`cpu[`*`n`*`].DCZID-log2-block-size`	Log_2(block size) cleared by DC ZVA instruction[c]	`int`	`0x0-0x9`	`0x8`
`dcache-size`	L1 D-cache size, in bytes	`int`	`0x4000-0x100000`	`0x8000`
`dcache-ways`	Number of L1 D-cache ways[d]	`int`	`0x1-0x40`	`0x2`
`icache-size`	L1 I-cache size, in bytes	`int`	`0x4000-0x100000`	`0x8000`
`icache-ways`	Number of L1 I-cache ways[d]	`int`	`0x1-0x40`	`0x2`
`l2cache-size`	L2 cache size, in bytes	`int`	`0x0-0x1000000`	`0x80000`
`l2cache-ways`	Number of L2 cache ways[d]	`int`	`0x1-0x40`	`0x10`

a. UNPREDICTABLE.
b. Unified caches allocate lines only if these parameters are enabled at both I-side and D-side.
c. As read from DCZID_EL0.
d. Sets are implicit from size.

4.34.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARMv8-A AEM general processor parameters* on page 4-46
- *ARMv8-A AEM L2 cache-controller parameters* on page 4-49
- *ARMv8-A AEM TLB parameters* on page 4-50
- *ARMv8-A AEM memory parameters* on page 4-51
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

4.35 ARMv8-A AEM L2 cache-controller parameters

You can configure the *Level 2* (L2) cache-controller of the multiprocessor with the L2 cache-controller parameters.

Table 4-37 L2 cache-controller parameters

Parameter	Description	Type	Range	Default
`l2cc.cache-state_modelled`	Model a functional cache state	`bool`	`false-true`	`false`
`l2cc.ASSOCIATIVITY`	Associativity for Auxiliary Control Register	`int`	`0x0-0x1`	`0x0`
`l2cc.CACHEID`	Cache controller cache ID	`int`	`0x0-0x3F`	`0x0`
`l2cc.WAYSIZE`	Size of ways for Auxiliary Control Register	`int`	`0x0-0x7`	`0x1`
`l2cc.CFGBIGEND`	Access configuration registers as big-endian on reset	`int`	`0x0-0x1`	`0x0`
`l2cc.LOCKDOWN_BY_MASTER`	Lockdown by master[a]	`int`	`0x0-0x1`	`0x0`
`l2cc.LOCKDOWN_BY_LINE`	Lockdown by line[b]	`int`	`0x0-0x1`	`0x0`

a. The value is reflected in CacheType register Bit 26, but the feature is not switched off when the parameter is 0.

b. The value is reflected in CacheType register Bit 25, but the feature is not switched off when the parameter is 0.

4.35.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general cache parameters* on page 4-48.

4.36 ARMv8-A AEM TLB parameters

You can configure the *Translation Lookaside Buffer* (TLB) configuration of the multiprocessor with the TLB parameters.

Table 4-38 TLB parameters

Parameter	Description	Type	Range	Default
`stage12_tlb_size`	Number of stage 1 and stage 2 TLB entries	`int`	`0x1-0xFFFFFFFF`	`0x80`
`stage1_tlb_size`	Number of stage 1 TLB entries	`int`	`0x0-0xFFFFFFFF`	`0x0`
`stage2_tlb_size`	Number of stage 2 TLB entries	`int`	`0x0-0xFFFFFFFF`	`0x0`
`stage1_walkcache_size`	Number of stage 1 TLB walk cache entries	`int`	`0x0-0xFFFFFFFF`	`0x0`
`stage2_walkcache_size`	Number of stage 2 TLB walk cache entries	`int`	`0x0-0xFFFFFFFF`	`0x0`
`instruction_tlb_size`	Number of stage 1 and stage 2 ITLB entries[a]	`int`	`0x0-0xFFFFFFFF`	`0x0`
`enable_tlb_contig_check`	Check consistency of TLB entries in regions with the Contiguous Bit set	`bool`	`false-true`	`true`
`has_tlb_conflict_abort`	Inconsistent TLB content generates aborts	`bool`	`false-true`	`false`
`use_tlb_contig_hint`	Pagetable entries with the Contiguous Bit set generate large TLB entries	`bool`	`false-true`	`false`

a. 0 for unified ITLB + DTLB.

4.36.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general cache parameters* on page 4-48.

4.37 ARMv8-A AEM memory parameters

You can configure the memory of the multiprocessor with the memory parameters.

Table 4-39 Memory parameters

Parameter	Description	Type	Range	Default
`elfloader.elf`	ELF file name	`string`	-	-
`elfloader.lfile`	Load file for large address mapping	`string`	-	-
`elfloader.ns_copy`	Copy whole file to NS memory space	`bool`	`false-true`	`true`

4.37.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARMv8-A AEM general processor parameters* on page 4-46
- *ARMv8-A AEM general cache parameters* on page 4-48.

4.38 ARMv8-A AEM debug architecture parameters

You can configure the debug architecture with the debug architecture parameters.

Table 4-40 Debug architecture parameters

Parameter	Description	Type	Range	Default
`DBGPIDR`	If zero, build a value for the *DeBuG Peripheral Identification Register* (DBGPIDR). If nonzero, override DBGPIDR with this value.	`int`	`0x0-0xFFFFFFFFFF`	`0x0`
`cpu[n].number-of-breakpoints`	Number of breakpoints.	`int`	`0x2-0x10`	`0x10`
`cpu[n].number-of-watchpoints`	Number of watchpoints.	`int`	`0x2-0x10`	`0x10`
`cpu[n].number-of-context-breakpoints`	Number of context-aware breakpoints.	`int`	`0x0-0x10`	`0x10`
`cpu[n].unpredictable_WPMASKANDBAS`	Constrained unpredictable handling of watchpoints when mask and BAS fields specified. 0 = IGNOREMASK, 1 = IGNOREBAS, 2 = REPEATBAS8, 3 = REPEATBAS.	`int`	`0x0-0x3`	`0x1`
`cpu[n].unpredictable_non-contigous_BAS`	Treat noncontiguous BAS field in watchpoint control register as all ones.	`bool`	`false-true`	`true`
`cpu[n].cti-number_of_triggers`	Number of CTI event triggers.	`int`	`0x0-0x8`	`0x8`
`cpu[n].cti-intack_mask`	Set bits mean the corresponding triggers need software acknowledgment through CTIINTACK.[a]	`int`	`0x0-0xFF`	`0x1`
`v8ect.has_CTIAUTHSTATUS`	Enable CTIAUTHSTATUS register.	`bool`	`false-true`	`true`
`v8ect.number-of-channels`	Number of channels in Cross Trigger Matrix.	`int`	`0x3-0x20`	`0x4`
`watchpoint-log2secondary_restriction`	Log_2(secondary restriction of FAR/EDWAR) on watchpoint hit for load/store operations.	`int`	`0x0-0x3F`	`0x0`

a. One bit per trigger.

4.38.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARMv8-A AEM general processor parameters* on page 4-46
- *ARMv8-A AEM general cache parameters* on page 4-48
- *ARMv8-A AEM memory parameters* on page 4-51
- *ARMv8-A AEM message parameters* on page 4-53
- *ARM Architecture Reference Manual ARMv8 edition*.

4.39 ARMv8-A AEM message parameters

You can configure the warning and error messages with the message parameters.

Table 4-41 Message parameters

Parameter	Description	Type	Default
`TRACE.ArchMsg.suppress_repeated`	Suppress repeated messages from similar call sites	`bool`	`true`
`TRACE.ArchMsg.suppress_sources`	Space-separated list of components or events to not print	`string`	-
`TRACE.ArchMsg.trace-file`	ArchMsg output file	`string`	-

4.39.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM debug architecture parameters* on page 4-52
- *Boundary features and architectural checkers* on page 4-56.

4.40 ARMv8-A AEM simulator parameters

You can configure the simulator with these parameters.

Table 4-42 ARMv8-A AEM simulator parameters

Parameter	Description	Type	Default
`scheduler_mode`	Control instruction interleaving. `0x0` = default long quantum, `0x1` = low latency mode, short quantum and signal checking, `0x2` = lock-breaking mode, long quantum with additional context switches near load-exclusive instructions.	`int`	`0x0-0x2`
`cpu[`*n*`].max_code_cache`	Maximum cache size for code translations, in bytes.	`int`	-

4.40.1 See also

Reference

- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARMv8-A AEM general cache parameters* on page 4-48.

4.41 Semihosting parameters

Semihosting is a method of running your target software on the model to communicate with the host environment. The AEM models permit the target C library to access the I/O facilities of the host computer, such as the filesystem, keyboard input, and clock.

The semihosting parameters are repeated in groups for each processor in the multiprocessor, from `cpu0` onwards.

Table 4-43 Semihosting parameters

Parameter	Description	Type	Range	Default
`cpu[n].semihosting-ARM_SVC`	A32 SVC number for semihosted calls	`int`	`0x0-0xFFFFFFFF`	`0x123456`
`cpu[n].semihosting-Thumb_SVC`	T32 SVC number for semihosted calls	`int`	`0x0-0xFFFFFFFF`	`0xAB`
`cpu[n].semihosting-cmd_line`	Program name and arguments, `argc`, `argv`, for target programs using the semihosted C library	`string`	-	-
`cpu[n].semihosting-cwd`	Virtual address of CWD	`string`	-	-
`cpu[n].semihosting-enable`	Enable semihosting of SVC instructions	`bool`	`false-true`	`true`
`cpu[n].semihosting-heap_base`	Virtual address of heap base	`int`	-	`0x00000000`
`cpu[n].semihosting-heap_limit`	Virtual address of top of heap	`int`	-	`0x0F000000`
`cpu[n].semihosting-stack_base`	Virtual address of base of descending stack	`int`	-	`0x10000000`
`cpu[n].semihosting-stack_limit`	Virtual address of stack limit	`int`	-	`0x0F000000`

4.41.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv7-A AEM general multiprocessor parameters* on page 4-30
- *ARMv7-A AEM multiprocessor parameters* on page 4-32
- *ARMv7-A AEM processor parameters* on page 4-33
- *ARMv7-A AEM memory parameters* on page 4-34
- *ARMv7-A AEM cache geometry parameters* on page 4-36
- *ARMv8-A AEM parameters* on page 4-41
- *ARMv8-A AEM general multiprocessor parameters* on page 4-42
- *ARMv8-A AEM general processor parameters* on page 4-46
- *ARMv8-A AEM general cache parameters* on page 4-48
- *ARMv8-A AEM memory parameters* on page 4-51.

4.42 Boundary features and architectural checkers

Boundary features and architectural checkers are model capabilities that help your development and testing process by exposing latent problems in the target code. Certain boundary features or architectural checkers, however, might have an adverse effect on the overall running speed of target code.

4.42.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41
- *Semihosting parameters* on page 4-55
- *IMPLEMENTATION DEFINED features* on page 4-57
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

4.43 IMPLEMENTATION DEFINED features

Some aspects of the behavior of the processor are IMPLEMENTATION DEFINED in the ARM architecture, meaning that they can legally vary between different processor implementations. Any code that is intended to be run portably across the multiple ARM implementations must take care when using any of these facilities, because they might or might not be present.

4.43.1 See also

Reference

- *ARMv7-A AEM parameters* on page 4-29
- *ARMv8-A AEM parameters* on page 4-41
- *Semihosting parameters* on page 4-55
- *Boundary features and architectural checkers* on page 4-56
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

4.44 Differences between the VE and CoreTile hardware and the models

The following topics describe features of the VE hardware that are not implemented in the models, or that have significant differences in implementation:

- *Memory map* on page 4-59
- *Memory aliasing* on page 4-60
- *Features not present in the model* on page 4-61
- *Features partially implemented in the model* on page 4-62
- *Restrictions on the processor models* on page 4-63
- *Timing considerations* on page 4-65.

4.44.1 See also

Reference

- *VE model memory map* on page 4-3
- *VE model parameters* on page 4-6.

4.45 Memory map

The model is based on the memory map of the hardware VE platform, but is not intended to be an accurate representation of a specific VE hardware revision. The memory map in the supplied model is sufficiently complete and accurate to boot the same operating system images as the VE hardware.

In the memory map, memory regions that are not explicitly occupied by a peripheral or by memory are unmapped. This includes regions otherwise occupied by a peripheral that is not implemented, and those areas that are documented as reserved. Accessing these regions from the host processor results in the model presenting a warning.

4.45.1 See also

Reference

- *Memory aliasing* on page 4-60
- *Features not present in the model* on page 4-61
- *Features partially implemented in the model* on page 4-62
- *Restrictions on the processor models* on page 4-63
- *Timing considerations* on page 4-65.

4.46 Memory aliasing

The model implements address space aliasing of the DRAM. This means that the same physical memory locations are visible at different addresses. The lower 2GB of the DRAM is accessible at `0x00_80000000`. The full 8GB of DRAM is accessible at `0x08_00000000` and again at `0x80_00000000`.

4.46.1 See also

Reference

- *Memory map* on page 4-59
- *Features not present in the model* on page 4-61
- *Features partially implemented in the model* on page 4-62
- *Restrictions on the processor models* on page 4-63
- *Timing considerations* on page 4-65.

4.47 Features not present in the model

The following features present on the hardware version of the VE motherboard are not implemented in the system models:

- two-wire serial bus interfaces
- USB interfaces
- PCI Express interfaces
- compact flash
- *Digital Visual Interface* (DVI)
- debug and test interfaces
- *Dynamic Memory Controller* (DMC)
- *Static Memory Controller* (SMC).

4.47.1 See also

Reference

- *VE model memory map* on page 4-3
- *Memory map* on page 4-59
- *Memory aliasing* on page 4-60
- *Features partially implemented in the model* on page 4-62
- *Restrictions on the processor models* on page 4-63
- *Timing considerations* on page 4-65.

4.48 Features partially implemented in the model

The Sound feature present on the hardware version of the VE motherboard is only partially implemented in the Fixed Virtual Platforms. Partial implementation means that some of the components are present, but the functionality has not been fully modeled. If you use such features, they might not work as you expect. Check the model release notes for the latest information.

For the Sound feature, the VE FVPs implement the PL041 AACI PrimeCell and the audio CODEC as in the VE hardware, but with a limited number of sample rates.

4.48.1 See also

Reference

- *Memory map* on page 4-59
- *Memory aliasing* on page 4-60
- *Features not present in the model* on page 4-61
- *Restrictions on the processor models* on page 4-63
- *Timing considerations* on page 4-65.

4.49 Restrictions on the processor models

Detailed information concerning what features are not fully implemented in the processor models included with the VE FVPs can be found in separate documentation. See the reference information at the end of this topic.

The following general restrictions apply to the Fixed Virtual Platform implementations of ARM processors:

- The simulator does not model cycle timing. In aggregate, all instructions execute in one processor master clock cycle, with the exception of Wait For Interrupt.
- Write buffers are not modeled, except in AEMs.
- Most aspects of TLB behavior are implemented in the models. In ARMv7 models, and later ones, the TLB memory attribute settings are used when stateful cache is enabled.
- No device-accurate MicroTLB is implemented.
- A single memory access port is implemented. The port combines accesses for instruction, data, DMA and peripherals. Configuration of the peripheral port memory map register is ignored.
- All memory accesses are atomic and are performed in programmer's view order. All transactions on the PVBus are a maximum of 64 bits wide. Unaligned accesses are always performed as byte transfers.
- Some instruction sequences are executed atomically, ahead of the component master clock, so that system time does advance during their execution. This can sometimes have an effect in sequential access of device registers where devices are expecting time to move on between each access.
- Interrupts are not taken at every instruction boundary.
- Integration and test registers are not implemented.
- Not all CP14 debug registers are implemented on all processors.
- Breakpoint types supported directly by the model are:
  - single address unconditional instruction breakpoints
  - single address unconditional data breakpoints
  - unconditional instruction address range breakpoints
- Processor exception breakpoints are supported by pseudoregisters in the debugger. Setting an exception register to a nonzero value stops execution on entry to the associated exception vector.
- Performance counters are not implemented on all models.

The following additional restrictions apply to the Fixed Virtual Platform implementation of a Cortex-A9 MPCore multiprocessor:

- The Cortex-A9MPCore multiprocessor contains some memory-mapped peripherals. These are modeled by the FVP.
- Two 4GB address spaces are seen by the model multiprocessor, one as seen from secure mode and one as seen from normal mode. The address spaces contain zero-wait state memory and peripherals, but a lot of the space is unmapped.
- The RR bit in the SCTLR is ignored.

- The Power Control Register in the system control coprocessor is implemented but writing to it does not change the behavior of the model.
- The SCU is only partially modeled:
  - The SCU enable bit is ignored. The SCU is always enabled.
  - The SCU ignores the invalidate all register.
  - Coherency operations are represented by a memory write followed by a read to refill from memory, rather than using cache-to-cache transfers.
  - There is no address filtering within the SCU. The enable bit for this feature is ignored.

4.49.1 See also

Reference

- *Memory map* on page 4-59
- *Memory aliasing* on page 4-60
- *Features not present in the model* on page 4-61
- *Features partially implemented in the model* on page 4-62
- *Timing considerations* on page 4-65
- *Fast Models Reference Manual*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0423-/index.html`.

4.50 Timing considerations

The Fixed Virtual Platforms provide an environment that enables running software applications in a functionally-accurate simulation. However, because of the relative balance of fast simulation speed over timing accuracy, there are situations where the models might behave unexpectedly.

When code interacts with real world devices like timers and keyboards, data arrives in the modeled device in real-world, or wall-clock, time, but simulation time can be running much faster than the wall clock. This means that a single keypress might be interpreted as several repeated key presses, or a single mouse click incorrectly becomes a double click.

The VE FVPs provide the Rate Limit feature to match simulation time to match wall-clock time. Enabling Rate Limit, either by using the Rate Limit button in the CLCD display, or the `rate_limit-enable` model instantiation parameter, forces the model to run at wall-clock time. This avoids issues with two clocks running at significantly different rates. For interactive applications, ARM recommends enabling Rate Limit.

4.50.1 See also

Reference

- *Memory map* on page 4-59
- *Memory aliasing* on page 4-60
- *Features not present in the model* on page 4-61
- *Features partially implemented in the model* on page 4-62
- *Restrictions on the processor models* on page 4-63.

Chapter 5
Programmer's Reference for the MPS FVPs

The following topics describe the memory map and the configuration registers for the peripheral and system component models:

Reference

5.1 MPS model memory map

This section describes the MPS memory map. For standard ARM peripherals, see the TRM for that device.

Table 5-1 Overview of MPS memory map

Description	Modeled	Address range
4MB Remap region for SRAM0 overlay of Flash	Yes	`0x00000000–0x003FFFFF`
Non remapped Flash memory	Yes	`0x00400000–0x03FFFFFF`
SRAM for code and data storage (remap RAM)	Yes	`0x10000000–0x103FFFFF`
SRAM for code and data storage	Yes	`0x10400000–0x107FFFFF`
FLASH aliased for programming	Yes	`0x18000000–0x1BFFFFFF`
Processor system registers	Yes	`0x1F000000–0x1F000FFF`
Reserved for SMC configuration registers	N/A	`0x1F001000–0x1F002FFF`
I2C for DVI	Yes	`0x1F003000–0x1F003FFF`
PL022 SPI for Touch Screen	Yes	`0x1F004000–0x1F004FFF`
PL011 UART	Yes	`0x1F005000–0x1F005FFF`
Reserved	N/A	`0x1F006000–0x1FFFFFFF`
SP805 Watchdog	Yes	`0x40000000–0x4000FFFF`
PL031 RTC	Yes	`0x40001000–0x40001FFF`
SP804 Timer (0)	Yes	`0x40002000–0x40002FFF`
SP804 Timer (1)	Yes	`0x40003000–0x40003FFF`
DUT system registers	Yes	`0x40004000–0x40004FFF`
PL181 SD/MMC controller	Yes	`0x40005000–0x40005FFF`
Reserved	N/A	`0x40006000–0x40006FFF`
PL011 UART (1)	Yes	`0x40007000–0x40007FFF`
PL011 UART (2)	Yes	`0x40008000–0x40008FFF`
PL011 UART (3)	Yes	`0x40009000–0x40009FFF`
PL041 AC97 controller	Yes	`0x4000A000–0x4000AFFF`
DS702 I2C (ADCDAC)	Partial[a]	`0x4000B000–0x4000BFFF`
DUT Character LCD	Yes	`0x4000C000–0x4000CFFF`
Reserved	N/A	`0x4000D000–0x4000EFFF`
Reserved	N/A	`0x4FFA0000–0x4FFAFFFF`
Flexray	Partial[a]	`0x4FFB0000–0x4FFBFFFF`
CAN	Partial[a]	`0x4FFC0000–0x4FFCFFFF`

Table 5-1 Overview of MPS memory map (continued)

Description	Modeled	Address range
LIN	Partial[a]	`0x4FFD0000–0x4FFDFFFF`
Ethernet	Partial[a]	`0x4FFE0000–0x4FFEFFFF`
Video	Yes	`0x4FFF0000–0x4FFFFFFF`
External AHB interface to DUT FPGA	Yes	`0x50000000–0x5FFFFFFF`
DMC	Yes	`0x60000000–0x9FFFFFFF`
SMC	Yes	`0xA0000000–0xAFFFFFFF`
Private Peripheral Bus	Yes	`0xE0000000–0xE00FFFFF`
System bus interface to DUT FPGA	Yes	`0xE0100000–0xFFFFFFFF`

a. This model is represented by a register bank and has no functionality beyond this.

——— **Note** ———

A Bus Error is generated for accesses to memory areas not shown in this table.

Any memory device that does not occupy the total region is aliased within that region.

———————————

5.1.1 See also

Reference

- *MPS registers* on page 5-4
- *MPS parameters* on page 5-11
- *Differences between the MPS hardware and the system model* on page 5-16.

5.2 MPS registers

The following topics describe the MPS memory-mapped registers:

- *Processor system registers* on page 5-5
- *DUT system registers* on page 5-6
- *Character LCD registers* on page 5-7
- *Memory configuration and remap* on page 5-8
- *Switches* on page 5-9
- *Seven-segment display* on page 5-10.

5.3 Processor system registers

The following table provides a description of the processor system registers.

Table 5-2 MPS processor system registers

Register name	Address	Access	Description
SYS_ID	`0x1f000000`	read/write	Board and FPGA identifier
SYS_MEMCFG	`0x1f000004`	read/write	Memory remap and alias
SYS_SW	`0x1f000008`	read/write	Indicates user switch settings
SYS_LED	`0x1f00000C`	read/write	Sets LED outputs
SYS_TS	`0x1f000010`	read/write	Touchscreen register

5.3.1 See also

Reference

- *MPS registers* on page 5-4
- *DUT system registers* on page 5-6
- *Character LCD registers* on page 5-7
- *Memory configuration and remap* on page 5-8
- *Switches* on page 5-9
- *Seven-segment display* on page 5-10.

5.4 DUT system registers

The following table provides a description of the DUT system registers.

Table 5-3 MPS DUT system registers

Register name	Address	Access	Description
SYS_ID	`0x40004000`	read/write	Board and FPGA identifier
SYS_PERCFG	`0x40004004`	read/write	Peripheral control signals
SYS_SW	`0x40004008`	read/write	Indicates user switch settings
SYS_LED	`0x4000400C`	read/write	Sets LED outputs
SYS_7SEG	`0x40004010`	read/write	Sets seven-segment LED outputs
SYS_CNT25MHZ	`0x40004014`	read/write	Free running counter incrementing at 25MHz.
SYS_CNT100HZ	`0x40004018`	read/write	Free running counter incrementing at 100Hz

5.4.1 See also

Reference

- *MPS registers* on page 5-4
- *Processor system registers* on page 5-5
- *Character LCD registers* on page 5-7
- *Memory configuration and remap* on page 5-8
- *Switches* on page 5-9
- *Seven-segment display* on page 5-10.

5.5 Character LCD registers

The following table provides a description of the character LCD registers.

Table 5-4 MPS LCD registers

Register name	Address	Access	Description
CHAR_COM	`0x4000C000`	write	Command register. The command set is compatible with the commands of the Hitachi HD44780U controller.
CHAR_DAT	`0x4000C004`	write	Write data register.
CHAR_RD	`0x4000C008`	read	Read data register.
CHAR_RAW	`0x4000C00C`	read/write	Raw interrupt.
CHAR_MASK	`0x4000C010`	read/write	Interrupt mask.
CHAR_STAT	`0x4000C014`	read/write	Masked interrupt.

5.5.1 See also

Reference

- *MPS registers* on page 5-4
- *Processor system registers* on page 5-5
- *DUT system registers* on page 5-6
- *Memory configuration and remap* on page 5-8
- *Switches* on page 5-9
- *Seven-segment display* on page 5-10.

5.6 Memory configuration and remap

The following table provides a description of the memory configuration register.

Table 5-5 Memory configuration

Name	Bits	Access	Power On Reset	Description
Reserved	31:3	-	-	-
SWDPEN	2	RW	0b	Single Wire Debug Port Enable. 1 is SWD 0 JTAG
ALIAS	1	RW	1b	Alias FLASH. 1 is Aliased on 0 Aliased off
REMAP	0	RW	0b	Remap SSRAM. 1 is Remap on 0 Remap off

The ability to remap the static RAM into the bottom of memory (overlaying the Flash) is required for booting and code execution to enable the interrupt vector table to be modified. It is also used to enable boot code execution from SRAM for code development, rather than programming the FLASH each time.

The aliasing of the Flash memory into SRAM space is required to permit the Flash memory to be reprogrammed at this offset. It also enables full flash memory access when remapping is enabled. If remapping of flash is disabled only the Flash memory above 4MB is accessible.

5.6.1 See also

Reference

- *MPS registers* on page 5-4
- *Processor system registers* on page 5-5
- *DUT system registers* on page 5-6
- *Character LCD registers* on page 5-7
- *Switches* on page 5-9
- *Seven-segment display* on page 5-10.

5.7 Switches

The following table lists the bits for the user switch inputs.

Table 5-6 User switches

Name	Bits	Access	Reset	Note
Reserved	31:8	-	-	-
USER_BUT[3:0]	7:4	RO	-	Always returns value of user buttons
USER_SW[3:0]	3:0	RO	-	Always returns value of user switches

5.7.1 See also

Reference

- *MPS registers* on page 5-4
- *Processor system registers* on page 5-5
- *DUT system registers* on page 5-6
- *Character LCD registers* on page 5-7
- *Memory configuration and remap* on page 5-8
- *Seven-segment display* on page 5-10.

5.8 Seven-segment display

The following table lists the bits that control the seven-segment display.

Table 5-7 Seven-segment register

Name	Bits	Access	Reset	Note
DISP3	31:24	RW	`0x00`	Segments for display 3
DISP2	23:16	RW	`0x00`	Segments for display 2
DISP1	15:8	RW	`0x00`	Segments for display 1
DISP0	7:0	RW	`0x00`	Segments for display 0

5.8.1 See also

Reference

- *MPS registers* on page 5-4
- *Processor system registers* on page 5-5
- *DUT system registers* on page 5-6
- *Character LCD registers* on page 5-7
- *Memory configuration and remap* on page 5-8
- *Switches* on page 5-9.

5.9 MPS parameters

The following topics describe the system parameters that can be configured at runtime:

- *MPS visualization parameters* on page 5-12
- *DUT parameters* on page 5-13
- *Terminal parameters* on page 5-14
- *Processor parameters* on page 5-15.

5.9.1 See also

Reference

- *MPS model memory map* on page 5-2
- *Differences between the MPS hardware and the system model* on page 5-16.

5.10 MPS visualization parameters

The following table provides a description of the visualization parameters for the MPSVisualisation component.

Table 5-8 Visualization parameters

Parameter name	Description	Type	Allowed value	Default value
`trap_key`	trap key that works with Left Ctrl key to toggle mouse pointer display	Integer	valid ATKeyCode key value[a]	74[b]
`rate_limit_enable`	rate limit simulation	Boolean	`true` or `false`	`true`
`disable_visualisation`	enable/disable visualization	Boolean	`true` or `false`	`false`

a. If you have Fast Models installed, see the header file, `%PVLIB_HOME%\components\KeyCode.h`, for a list of ATKeyCode values. On Linux use `$PVLIB_HOME/components/KeyCode.h`.

b. This is equivalent to the Left Alt key.

5.10.1 See also

Reference

- *DUT parameters* on page 5-13
- *Terminal parameters* on page 5-14
- *Processor parameters* on page 5-15.

5.11 DUT parameters

The following table provides a description of the parameters for the DUT.

Table 5-9 DUT parameters

Parameter name	Description	Type	Allowed value	Default value
`mps_dut.dut_sysregs.user_switches_value`	User switches	Integer	`0x0–0xFF`	0
`mps_dut.mmc.p_mmc_file`	MMC contents file name	String		mmc.dat
`mps_dut.sp805.simhalt`	Halt on reset	Boolean	`true` or `false`	`false`
`mps_dut.uart[0|1|2].untimed_fifos`	Operate UART FIFO in fast (no timing) mode	Boolean	`true` or `false`	`false`
`mps_dut.uart[0|1|2].unbuffered_output`	Unbuffered output	Boolean	`true` or `false`	`false`

5.11.1 See also

Reference

- *MPS visualization parameters* on page 5-12
- *Terminal parameters* on page 5-14
- *Processor parameters* on page 5-15.

5.12 Terminal parameters

When the MPS FVP starts, a TCP/IP port for each enabled Terminal is opened. This is port 5000 by default, but increments by 1 until a free user port is found. For more information on using the Terminal component, see the reference information at the end of this topic.

The table below lists the terminal instantiation-time parameters that you can change when you start the model. The syntax to use in a configuration file is:

```
terminal_x.parameter=value
```

where *x* is the terminal ID 0, 1, 2 or 3 and *parameter* is the parameter name.

Note

The telnet Terminal does not obey control flow signals. This means that the timing characteristics of Terminal are not the same as a standard serial port.

Table 5-10 Terminal instantiation parameters

Component name	Parameter	Description	Type	Values	Default
`terminal_[0-3]`	`mode`	Terminal operation mode.	String	`telnet`[a], `raw`[b]	`telnet`
`terminal_[0-3]`	`start_telnet`	Enable terminal when the system starts.	Boolean	`true` or `false`	`true`
`terminal_[0-3]`	`start_port`	Port used for the terminal when the system starts. If the specified port is not free, the port value is incremented by 1 until a free port is found.	Integer	Valid port number	5000

a. In telnet mode, the Terminal component supports a subset of the telnet protocol defined in RFC 854.

b. In raw mode, the Terminal component does not interpret or modify the byte stream contents.

5.12.1 See also

Reference

- *Using a terminal with a system model* on page 3-17
- *MPS visualization parameters* on page 5-12
- *DUT parameters* on page 5-13
- *Processor parameters* on page 5-15.

5.13 Processor parameters

This section describes the configuration parameters for the ARM Cortex-M3 and Cortex-M4 processor models.

Table 5-11 Processor parameters

Parameter	Description	Type	Allowed Value	Default Value
`semihosting-cmd_line`[a]	Command line available to semihosting SVC calls.	String	no limit except memory	[empty string]
`semihosting-cwd`	Virtual address of CWD.	String	-	-
`semihosting-enable`	Enable semihosting SVC traps. **Caution** Applications that do not use semihosting must set this parameter to false.	Boolean	`true` or `false`	`true`
`semihosting-Thumb_SVC`	T32 SVC number for semihosting.	Integer	8 bit integer	`0xAB`
`semihosting-heap_base`	Virtual address of heap base.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x0`
`semihosting-heap_limit`	Virtual address of top of heap.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10700000`
`semihosting-stack_base`	Virtual address of base of descending stack.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10700000`
`semihosting-stack_limit`	Virtual address of stack limit.	Integer	`0x00000000 - 0xFFFFFFFF`	`0x10800000`
`coretile.fname`	Flash loader filename.	String	-	[empty string]
`coretile.flashloader.fnameWrite`	Filename to write to if flash image is modified.	String	-	[empty string]
`coretile.uart3.untimed_fifos`	Ignore the clock rate and transmit or receive serial data immediately.	Boolean	`true` or `false`	`false`
`coretile.uart3.unbuffered_output`	Unbuffered output.	Boolean	`true` or `false`	`false`

a. The value of argv[0] points to the first command line argument, not to the name of an image.

5.13.1 See also

Reference

- *MPS visualization parameters* on page 5-12
- *DUT parameters* on page 5-13
- *Terminal parameters* on page 5-14.

5.14 Differences between the MPS hardware and the system model

The following topics describe features of the MPS hardware that are not implemented in the models or have significant differences in implementation:

- *Features not present in the model* on page 5-17
- *Timing considerations* on page 5-18.

5.14.1 See also

Reference

- *MPS model memory map* on page 5-2
- *MPS parameters* on page 5-11.

5.15 Features not present in the model

The Ethernet component is currently not implemented in either the model or the hardware.

The following features present on the hardware version of the MPS are not implemented in the system models:

- I2C interface
- CAN interface
- LIN
- FlexRay.

The MPS model implements the PL041 AACI PrimeCell and the audio CODEC as in the MPS hardware, but with a limited number of sample rates. AACI Audio Input is not supported.

——— **Note** ———

The sound component present on the hardware version of the MPS is only partially implemented in the model.

Partial implementation means that some of the components are present, but the functionality has not been fully modeled. If you use these features, the model might not behave as you expect. Check the model release notes for the latest information.

5.15.1 See also

Reference

- *Differences between the MPS hardware and the system model* on page 5-16
- *Timing considerations* on page 5-18.

5.16 Timing considerations

The Fixed Virtual Platforms provide you with an environment that enables running software applications in a functionally-accurate simulation. However, because of the relative balance of fast simulation speed over timing accuracy, there are situations where the models might behave unexpectedly.

If your code interacts with real world devices such as timers and keyboards, data arrives in the modeled device in real world, or wall clock, time, but simulation time can be running much faster than the wall clock. This means that a single keypress might be interpreted as several repeated key presses, or a single mouse click incorrectly becomes a double click.

To correct for this, the MPS FVP provides the Rate Limit feature. Enabling Rate Limit, either using the Rate Limit button in the CLCD display, or the `rate_limit-enable` model instantiation parameter, forces the model to run at wall clock time. This avoids issues with two clocks running at significantly different rates. For interactive applications, ARM recommends enabling Rate Limit.

5.16.1 See also

Reference

- *Differences between the MPS hardware and the system model* on page 5-16
- *Features not present in the model* on page 5-17.