ARM® Cortex®-M33 Processor

Revision: r0p2

Integration and Implementation Manual

Confidential

ARM 100323_0002_00_en

ARM® Cortex®-M33 Processor

Integration and Implementation Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	26 September 2016	Confidential	First release for r0p0.
0001-00	03 February 2017	Confidential	First release for r0p1.
0002-00	15 May 2017	Confidential	First release for r0p2

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

Confidentiality Status

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

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

Web Address

http://www.arm.com

Contents

ARM® Cortex®-M33 Processor Integration and Implementation Manual

Appendix H Revisions

Preface

This preface introduces the *ARM® Cortex®-M33 Processor Integration and Implementation Manual*.

It contains the following:

- *About this book* on page 10.
- *Feedback* on page 14.

About this book

This book is for the ARM®Cortex®-M33 Processor.

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

r*m* Identifies the major revision of the product, for example, r1.

p*n* Identifies the minor revision or modification status of the product, for example, p2.

Implementation obligations

This book is designed to help you implement an ARM product. The extent to which the deliverables may be modified or disclosed is governed by the contract between ARM and Licensee. There may be validation requirements, which if applicable will be detailed in the contract between ARM and Licensee and which if present must be complied with prior to the distribution of any silicon devices incorporating the technology described in this document. Reproduction of this document is only permitted in accordance with the licences granted to Licensee.

ARM assumes no liability for your overall system design and performance, the verification procedures defined by ARM are only intended to verify the correct implementation of the technology licensed by ARM, and are not intended to test the functionality or performance of the overall system. You or the Licensee will be responsible for performing any system level tests.

You are responsible for any applications which are used in conjunction with the ARM technology described in this document, and in order to minimize risks adequate design and operating safeguards should be provided for by you. ARM's publication of any information in this document of information regarding any third party’s products or services is not an express or implied approval or endorsement of the use thereof.

Intended audience

This manual is written for experienced hardware engineers who might or might not have experience of ARM products, but who have experience of writing Verilog and of performing synthesis, and who want to implement and integrate a Cortex®-M33 processor in a *System-on-Chip* (SoC) design.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter provides an overview of the process of integrating and implementing the Cortex-M33 processor.

Chapter 2 Configuration Guidelines
This chapter describes the guidelines for RTL configuration. You can configure the RTL to tailor your implementation to the specific requirements of the target application.

Chapter 3 Key Integration Point
This chapter describes the key integration points that you must consider when you integrate the Cortex-M33 processor in your SoC design.

Chapter 4 Functional Integration Guidelines
This chapter describes the functional integration requirements of the macrocell in your SoC design.

Chapter 5 DFT Integration Guidelines
This chapter describes the DFT integration guidelines for SoC integration, and the issues that relate to DFT that you must consider when you integrate the processor into your SoC design.

Chapter 6 Key Implementation Points

This chapter describes the key implementation points you must consider when you implement the processor.

Chapter 7 Floorplan Guidelines

This chapter describes the floorplan that is used as a starting point for your design.

Chapter 8 Netlist Dynamic Verification

This chapter describes how to test the functionality of your implementation of the processor.

Chapter 9 Sign-off

This chapter describes the sign-off criteria.

Chapter 10 Execution Testbench

This chapter describes the execution testbench for the Cortex-M33 processor.

Chapter 11 Low-power Integration

This chapter describes how to use the low-power features of the Cortex-M33 processor in your system.

Chapter 12 Power Intent

This chapter describes the optional power gating features of the processor and how the logic can be divided into different power domains.

Chapter 13 DSM Generation

This chapter describes how to generate a *Design Simulation Model* (DSM).

Appendix A GPIO Programmers Model

This appendix describes the GPIO programmers model.

Appendix B CoreSight SoC

This appendix describes the location of the CoreSight SoC-400 configuration files for the Cortex-M33 processor, and describes the CoreSight SoC-400 trace comparison tools.

Appendix C DAP and TPIU signals and implementation constraints

This appendix describes the DAP and TPIU signals and implementation constraints.

Appendix D Signal Timing Constraints

This appendix describes the timing constraints on the processor signals.

Appendix E Tarmac Tracing

This appendix describes how to control generation of a `tarmac` trace file during a simulation that traces program execution of a Cortex-M33 processor.

Appendix F TEALMCU

This appendix describes the `TEALMCU` module.

Appendix G IP-XACT

This appendix describes the location and configuration of the IP-XACT files.

Appendix H Revisions

This appendix describes the technical changes between released issues of this document.

Glossary

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

See the *ARM Glossary* for more information.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace`

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

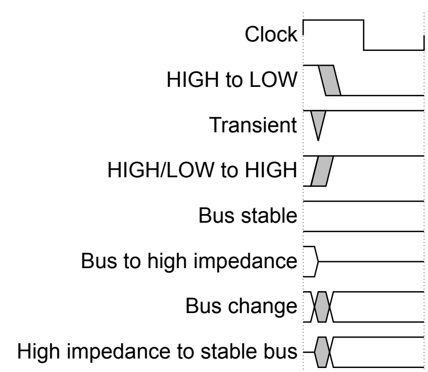


Figure 1 Key to timing diagram conventions

Signals

The signal conventions are:

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name denotes an active-LOW signal.

Additional reading

This book contains information that is specific to this product. See the following documents for other relevant information.

ARM publications

- *ARM®v8-M Architecture Reference Manual* (ARM DDI 0553).
- *ARM® Cortex®-M33 Processor Technical Reference Manual* (ARM 100230).
- *ARM® CoreSight™ ETM-M33 Technical Reference Manual* (ARM 100232)
- *ARM® CoreSight™ MTB-M33 Technical Reference Manual* (ARM 100231)
- *ARM® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (ARM IHI 0031).
- *AMBA® APB Protocol Version 2.0 Specification* (ARM IHI 0024).
- *AMBA® 4 ATB Protocol Specification* (ARM IHI 0032).
- *ARM® AMBA® 5 AHB Protocol Specification* (ARM IHI 0033).
- *Low Power Interface Specification ARM® Q-Channel and P-Channel Interfaces* (ARM IHI 0068).
- *ARM® CoreSight™ SoC-400 Technical Reference Manual* (ARM DDI 0480).
- *ARM® CoreSight™ SoC-400 User Guide* (ARM DUI 0563).
- *ARM® Embedded Trace Macrocell Architecture Specification ETMv4* (ARM IHI 0064).
- *ARM® Cortex®-M33 MCU Release Note* (AT623-DC-06003).
- *CoreSight™ ETM-M33 Release Note* (TM976-DC-06003).
- *CoreSight™ MTB-M33 Release Note* (TM977-DC-06003).

The following confidential books are only available to licensees:

- *ARM® CoreSight™ SoC-400 Implementation Guide* (ARM DDI 0267).

Other publications

None.

Feedback

Feedback on this product

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

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

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

- The title *ARM Cortex-M33 Processor Integration and Implementation Manual*.
- The number ARM 100323_0002_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Chapter 1
Introduction

This chapter provides an overview of the process of integrating and implementing the Cortex-M33 processor.

It contains the following sections:

1.1 About the processor

The Cortex-M33 processor is a high performance, low gate count, highly configurable, and energy efficient processor. It is intended for microcontroller and embedded applications that require an efficient mix of control capability and signal processing instructions.

The Cortex-M33 processor top-level module name is `TEAL` located in the `TEAL.v` file in the `logical/teal/verilog` directory.

Note

The `TEAL` module is also referred to as the processor or Cortex-M33 processor in this book.

This document explains how to integrate and implement the `TEAL` module into your system. The `TEAL` module enables integration of the following main blocks:

- The processor core.
- An optional *Embedded Trace Macrocell* (ETM), which is licensed and delivered separately from the processor. See *ARM® CoreSight™ ETM-M33 Technical Reference Manual* (ARM 100232).
- An optional *Wake-up Interrupt Controller* (WIC).
- An optional *Cross Trigger Interface* (CTI).
- An optional *Micro Trace Buffer* (MTB), which is licensed and delivered separately from the processor. See *ARM® CoreSight™ MTB-M33 Technical Reference Manual* (ARM 100231).
- The Cortex-M33 processor ROM table that identifies the components in `TEAL`.

The following figure shows a block diagram of the `TEAL` module including the main interfaces.

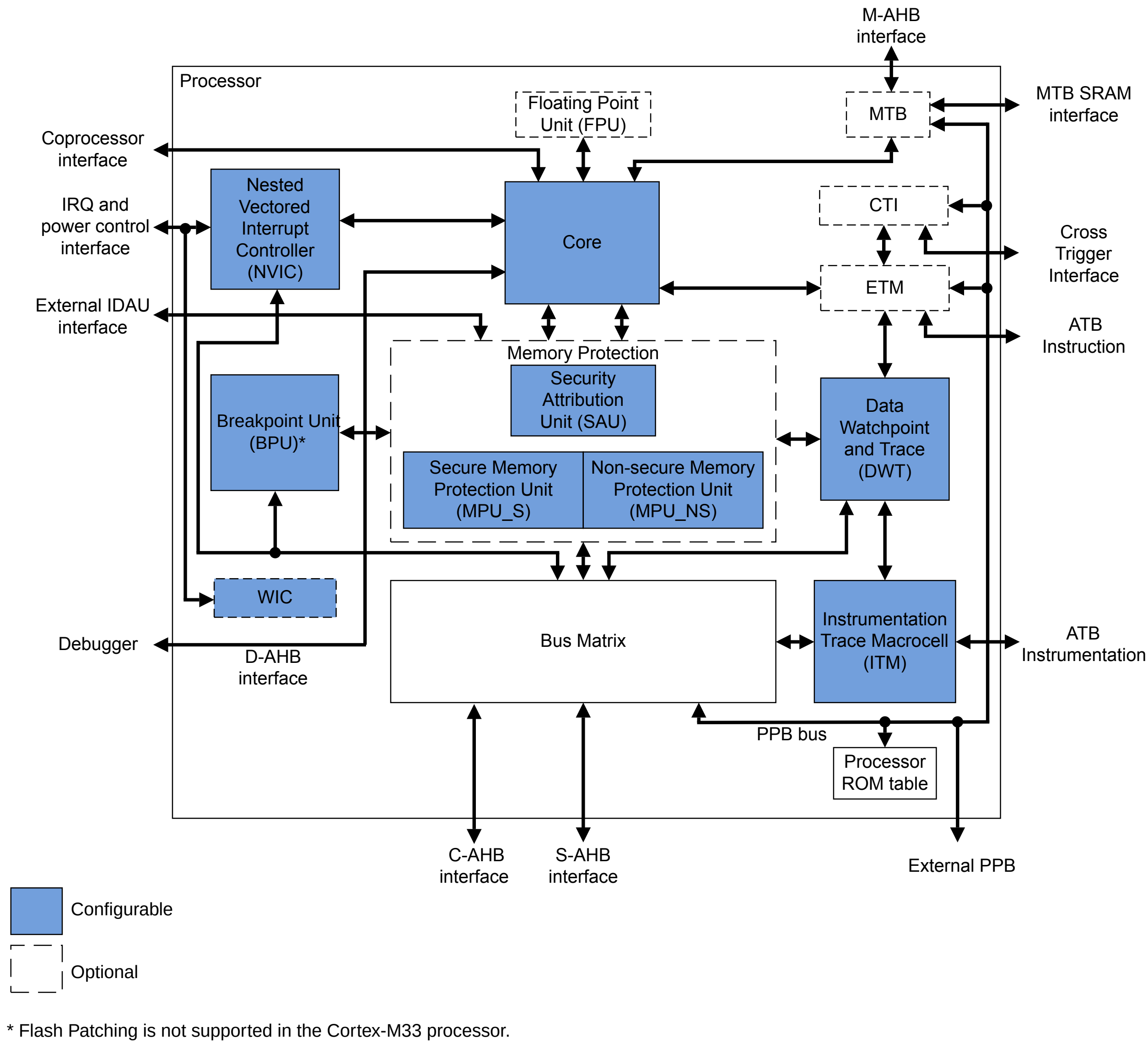


Figure 1-1 TEAL block diagram

TEAL is configurable by setting the parameters listed in the TEAL_CONFIG.v file, see *Table 2-1 Cortex-M33 processor configuration options summary* on page 2-27.

Note

- ARM recommends that you specify the required parameter values in each TEAL module instantiation.
- You must not modify TEAL.v or any other file unless directed to do so.

The following figure shows an example system integration of the TEAL module, with a minimal debug and trace implementation.

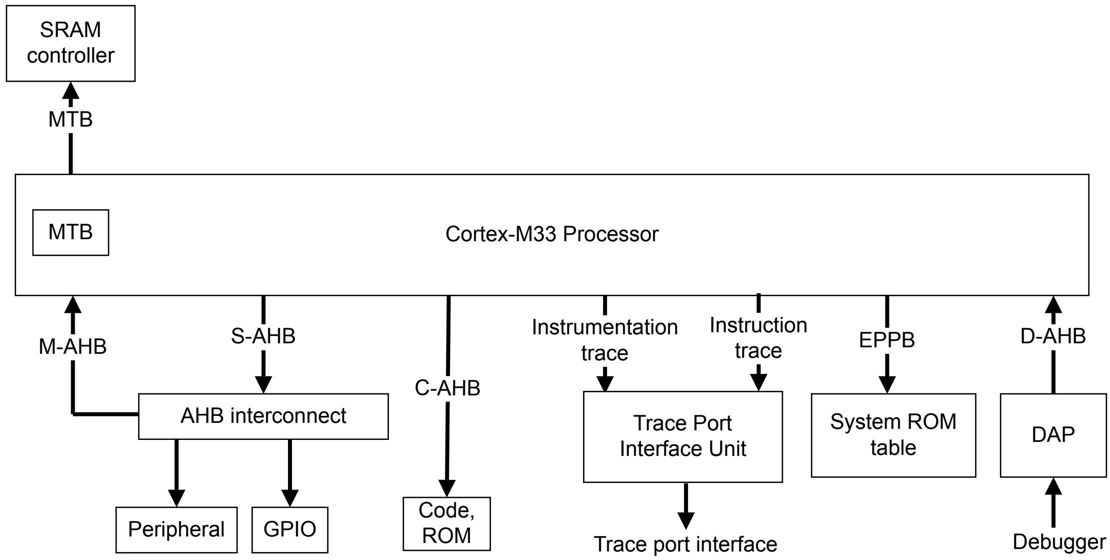


Figure 1-2 Example system integration

Included in the Cortex-M33 processor deliverables is the `TEALMCU` module, which is used in the execution testbench. The module is an example system containing the processor and minimal debug components.

Related concepts

2.2 Configuration options on page 2-27.

Related references

Chapter 10 Execution Testbench on page 10-104.
Appendix F TEALMCU on page Appx-F-228.

1.2 About implementation and integration

The flow that you use to integrate the processor into your SoC depends on your preferred usage model. You can first implement the processor on its own, and then integrate it into your system. Alternatively, you can integrate the processor first, and then implement the processor and your system at the same time.

The following figure shows the recommended implementation and integration flow. The flow assumes that you implement the Cortex-M33 processor and then integrate the implemented processor into your system.

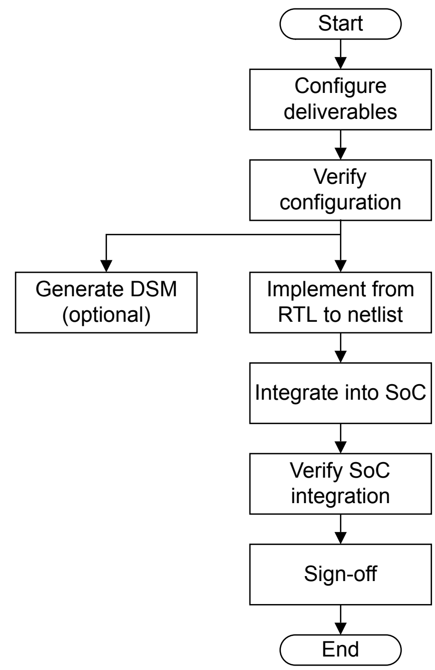


Figure 1-3 Implementation and integration flow

The following figure shows the integration and implementation flow when you integrate the Cortex-M33 processor into your system first, and then implement your system.

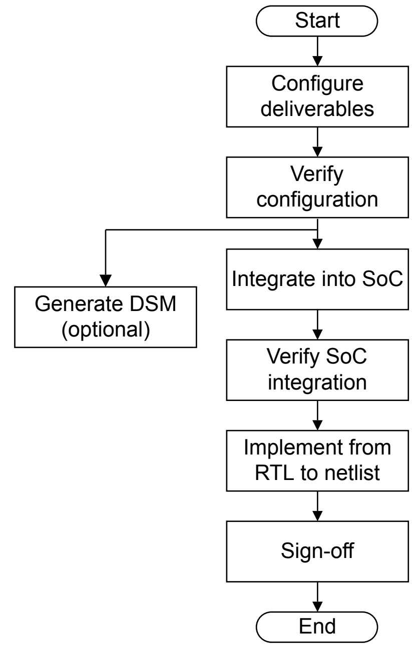


Figure 1-4 Integration and implementation flow

1.2.1 Integration

Integration is the process of including the processor in your SoC design.

Integration involves:

- Connecting the required clocks and resets to the processor.
- Connecting the processor to all the necessary peripherals and buses.
- Connecting the processor to the DFT logic in your SoC design.
- Tying off the configuration and any unused input signals.
- Verifying the processor within your SoC design.

Although you might have extra elements in your design, the main connections are the:

- *Code region AHB* (C-AHB) interface.
- *System AHB* (S-AHB) interface.
- *External PPB* (EPPB) APB interface.
- *Debug AHB* (D-AHB) interface.
- ATB trace master interfaces.

1.2.2 Implementation

The following figure shows the top-level inputs, resources, outputs, and controls and constraints for implementation.

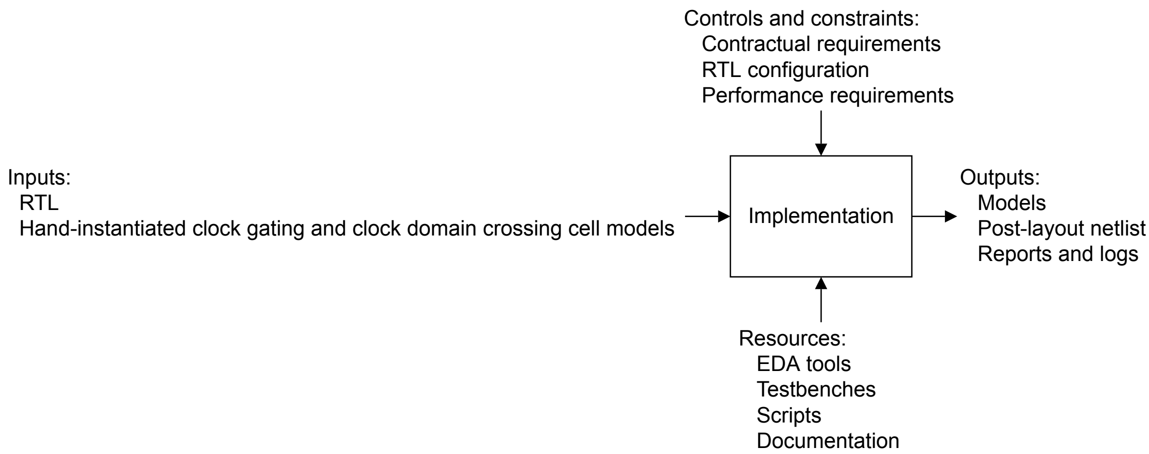


Figure 1-5 Implementation process

Implementation resources

This book assumes that you have suitable EDA tools and compute resources for implementation.

For a list of deliverables, and any specific tool revisions that are required for implementation, see the:
- *ARM® Cortex®-M33 MCU Release Note* .
- *CoreSight™ ETM-M33 Release Note* .
- *CoreSight™ MTB-M33 Release Note* .

Implementation controls and constraints

The general controls and constraints that apply to implementation, and implementing the device in accordance with your contract are given in this section.

Figure 1-5 Implementation process on page 1-21 shows the general controls and constraints that apply to the implementation. You must implement the device in accordance with your contract, see *Implementation obligations* on page 10.

Implementation inputs

The *ARM® Cortex®-M33 MCU Release Note* describes the deliverables that are inputs to the implementation flow. These deliverables include:
- *Register Transfer Level* (RTL) code.
- Implementation scripts.
- Documentation.

Implementation outputs

The outputs from the implementation flow are:
- Logs and reports:
  — Synthesis logs and reports.
  — Post-layout *Static Timing Analysis* (STA) logs and reports.
  — Logs and reports showing logical equivalence of post-layout netlist with implemented RTL.
- Components:
  — Post-layout netlist.
  — Layout database.
  — *Standard Delay Format* (SDF).
- Test:
  — *Automatic Test Pattern Generation* (ATPG) vectors.

Related concepts

1.4 Reference data on page 1-24.

Related references

Implementation obligations on page 10.

1.2.3 Implementation flow

The following figure shows the implementation flow.

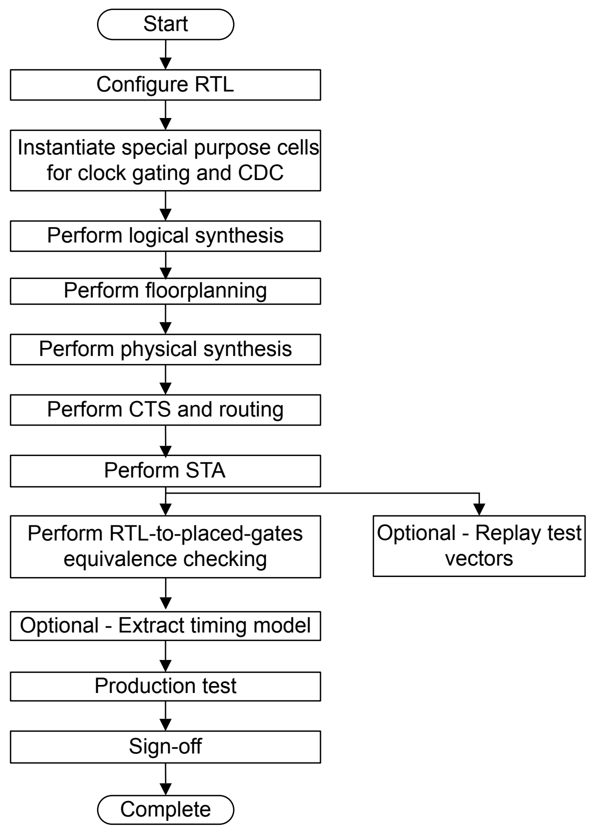


Figure 1-6 Implementation flow

Note

For information on contractual obligations to complete sign-off as part of the complete flow, see *Sign-off* and the *Cortex®-M33 Implementation Reference Methodology* documentation.

Related references

Chapter 9 Sign-off on page 9-98.

1.3 About sign-off

In addition to your normal sign-off checks, you must satisfy certain verification criteria before you sign off the design.

Related references

Chapter 9 Sign-off on page 9-98.

1.4 Reference data

Before starting, you must ensure that the unpacked deliverables are located in the correct directory structure.

The following figure shows the directory structure when you unpack the deliverables.

```
├─Cortex-M33-<version>-<quality>_ReleaseNote.pdf
├─CoreSight ETM-M33-<version>-<quality>_ReleaseNote.pdf†
├─CoreSight MTB-M33-<version>-<quality>_ReleaseNote.pdf†
└─teal/
   ├─documentation/
   ├─implementation_<foundry>_<process>/
   └─logical/
      ├─models/
      │  └─cells/
      │     └─generic/
      ├─teal/
      │  ├─dsm/
      │  ├─ipxact/
      │  ├─verilog/
      │  └─power_intent
      ├─tealbpu/
      │  └─verilog/
      ├─tealcore/
      │  └─verilog/
      ├─tealcti/
      │  └─verilog/
      ├─tealdap/
      │  └─verilog/
      ├─tealdwt/
      │  └─verilog/
      ├─tealetm†/
      │  └─verilog/
      ├─tealfpu†/
      │  └─verilog/
      ├─tealitm/
      │  └─verilog/
      ├─tealmcu/
      │  └─verilog/
      ├─tealmpu/
      │  └─verilog/
      ├─tealmtb†/
      │  └─verilog/
      ├─tealmtx/
      │  └─verilog/
      ├─tealnvic/
      │  └─verilog/
      ├─tealpcr/
      │  └─verilog/
      ├─tealtpiu/
      │  └─verilog/
      ├─testbench/
      │  ├─execution_tb/
      │  │  ├─verilog/
      │  │  │  └─example_sys/
      │  │  └─tests/
      │  │     ├─pre_compiled
      │  │     ├─CMSIS/
      │  │     └─Device/
      │  ├─integration_cssoc/
      │  ├─shared/
      │  │  ├─tarmac/
      │  │  └─shared/
      │  │     └─tools/
      │  │        └─bin/
      │  └─vector_replay_tb/
      └─shared/
         ├─tools/
         │  └─bin/
         └─ipxact/
            └─busdefs/
```

†if included, licensed and delivered separately from the processor.

Figure 1-7 Release directory structure

Chapter 2
Configuration Guidelines

This chapter describes the guidelines for RTL configuration. You can configure the RTL to tailor your implementation to the specific requirements of the target application.

It contains the following sections:

2.1 About configuration guidelines

For successful configuration of the RTL, you must set up the configurable options.

Caution

Failure to complete all the necessary configuration processes can result in malfunction.

2.2 Configuration options

Verilog parameters control all the configuration options. You must either alter the default values of the parameters in the configuration file, or override these parameters using instantiation.

The configuration method that you choose depends on the integration and implementation flow that you use. If you are implementing the processor on its own, ARM recommends that configuration is performed using the `TEAL_CONFIG.v` file in the `logical/teal/verilog` directory. If you are implementing the processor in a system, ARM recommends that you set the configuration parameters in the instantiation of the processor.

The following table shows a summary of the Cortex-M33 processor configuration options.

Table 2-1 Cortex-M33 processor configuration options summary

Parameter	Default value	Supported values	Description
`FPU`	1	0, 1	Specifies whether *Floating Point Unit* (FPU) is present. The options are: **0** No FPU hardware. **1** Single-precision FPU.
`DSP`	1	0, 1	Specifies whether the *Digital Signal Processing* (DSP) extension instructions are included. The options are: **0** DSP Extension not included. **1** DSP Extension included.
`SECEXT`	1	0, 1	Specifies whether the ARMv8-M Security Extension is included. The options are: **0** Security Extension not included. **1** Security Extension included.
`CPIF`	1	0, 1	Specifies whether the coprocessor interface is included. The options are: **0** Coprocessor interface is not included. **1** Coprocessor interface is included.
`MPU_NS`	8	0, 4, 8, 12, 16	Specifies the number of Non-secure *Memory Protection Unit* (MPU) regions included. The options are: **0** No MPU regions. **4** 4-region MPU. **8** 8-region MPU. **12** 12-region MPU. **16** 16-region MPU. If `SECEXT` is set to 0, this is the total number of MPU regions included. **Note** All Non-secure MPU regions can be disabled by the input signal **MPUNSDISABLE**.

Table 2-1 Cortex-M33 processor configuration options summary (continued)

Parameter	Default value	Supported values	Description
`MPU_S`	8	0, 4, 8, 12, 16	Specifies the number of Secure MPU regions included. The options are: **0** No MPU regions. **4** 4-region MPU. **8** 8-region MPU. **12** 12-region MPU. **16** 16-region MPU. If `SECEXT` is set to 0, this parameter is ignored. **Note** If `SECEXT` is set to 1, all Secure MPU regions can be disabled using the input signal **MPUSDISABLE**.
`SAU`	4	0, 4, 8	Specifies the number of *Security Attribution Unit* (SAU) regions included. The options are: **0** No SAU regions. **4** 4-region SAU. **8** 8-region SAU. If `SECEXT` is set to: **0** This parameter is ignored. **1** All SAU regions can be disabled using the input signal **SAUDISABLE**. If `SAU` is set to 0, an external *Implementation Defined Attribution Unit* (IDAU) interface can still specify the security of the memory regions.
`NUMIRQ`	32	1-480	Specifies the number of implemented user interrupts. The options are: **1** Input signal `IRQ`[0] **2** Input signal `IRQ`[1:0] **...** **480** Input signal `IRQ`[479:0]
`IRQLVL`	3	3-8	Specifies the number of bits of interrupt priority implemented in the NVIC. For example, a value of 3 results in 8 levels of priority, and a value of 8 results in 256 levels. See *ARM®v8-M Architecture Reference Manual*.
`IRQLATENCY`	`0xFFFFFFFF`	`[NUMIRQ-1:0]`	Specifies the individual interrupts which support the lowest interrupt latency. Each bit in `IRQLATENCY` corresponds to an interrupt. If the value of a bit in `IRQLATENCY` is `0b1`, it indicates that the corresponding `IRQ` is low latency.
`IRQDIS`	0	`[NUMIRQ-1:0]`	Disables support for individual interrupts. Each bit in `IRQDIS` corresponds to an interrupt. If the value of a bit in `IRQDIS` is `0b1`, it indicates that the corresponding `IRQ` is not present.
`DBGLVL`	2	0, 1, 2	Specifies the number of debug resources included. The options are: **0** Minimal debug. No Halting debug or memory access. **1** Reduced set. Two watchpoint and four breakpoint comparators. **2** Full set. Four watchpoint and eight breakpoint comparators. Debug monitor mode is always supported.

Table 2-1 Cortex-M33 processor configuration options summary (continued)

Parameter	Default value	Supported values	Description
`ITM`	1	0, 1	Specifies the level of instrumentation trace supported. The options are: **0** No *Instrumentation Trace Macrocell* (ITM) trace included. DWT triggers and counters are not included. **1** Include DWT and ITM trace. **Note** If `DBGLVL` is set to 0, no trace is included in the processor indicating ITM is ignored.
`ETM`	0	0, 1	Specifies support for ETM trace. The options are: **0** No ETM trace included. **1** ETM is included. **Note** If `ETM` is set to 1, you must have licensed the ETM unit. The `ETM` parameter is ignored if `DBGLVL` is 0.
`MTB`	0	0, 1	Specifies support for MTB trace. The options are: **0** No MTB trace included. **1** MTB included. **Note** If `MTB` is set to 1, you must have licensed the MTB unit. The `MTB` parameter is ignored if `DBGLVL` is 0.
`MTBAWIDTH`	32	5-32	Specifies the MTB RAM interface address width. The RAM data width is always 32 bits. **Note** If you exclude the MTB unit, this parameter has no effect. See the `MTB` parameter description in this table.
`WIC`	1	0, 1	Specifies whether the WIC is included. The options are: **0** WIC not included. **1** WIC included.

Table 2-1 Cortex-M33 processor configuration options summary (continued)

Parameter	Default value	Supported values	Description
`WICLINES`	`NUMIRQ+3`	4 to `NUMIRQ+3`	Makes the WIC sensitive to a configurable number of interrupts or a configurable number of events. The range of this parameter is 4-483. Three WIC lines are always used by the **RXEV**, **NMI**, and **EDBGRQ** signals. The remaining WIC lines are used by the **IRQ** signals. The minimum width of the **IRQ** signal is 1. See the `NUMIRQ` parameter. The options are: **4** **RXEV**, **NMI**, **EDBGRQ**, and **IRQ[0]** **5** **RXEV**, **NMI**, **EDBGRQ**, and **IRQ[1:0]** **...** ... **483** **RXEV**, **NMI**, **EDBGRQ**, and **IRQ[479:0]** **Note** If you exclude the WIC interface, this parameter has no effect. See the `WIC` parameter description in this table.
`CTI`	1	0, 1	Specifies whether CTI unit is included. The options are: **0** CTI not included. **1** CTI is included.
`RAR`	0	0, 1	Specifies whether all the synchronous states are, or only the architecturally required state is reset. The options are: **0** Only reset the architecturally required state. **1** Reset all synchronous states. **Note** Setting this parameter increases the size of the registers that are not reset by default, and also increases the overall area of the implementation.

Chapter 3
Key Integration Point

This chapter describes the key integration points that you must consider when you integrate the Cortex-M33 processor in your SoC design.

It contains the following sections:

3.1 About key integration points

When you integrate the processor with your SoC design, you must consider several key integration points and complete all the necessary integration steps. You can use the information in this section to check that you have covered everything.

3.2 Key integration tasks

List of key integration tasks for the Cortex-M33 processor.

Table 3-1 Key integration tasks

Key task	Description
1. Connect the **CLKIN** clock and clock enable signals.	See *4.2 Clocking and resets* on page 4-36
2. Connect the **nPORESET**, and **nSYSRESET** resets.	See *4.2 Clocking and resets* on page 4-36
3. Tie off or connect the following interface inputs appropriately: • Configuration signals. • Instruction execution control signals. • Code and System AHB interfaces. • D-AHB interface. • External Private Peripheral Bus. • External coprocessor interface. • Debug signals. • Power control and sleep interface. • ITM interface signals. • ETM instruction trace interface. • Interrupt interface. • Miscellaneous signals. • FPU exception signals. • Events and errors. • External Secure Attribution Unit Interface. • SysTick signals. • Test interfaces. • CoreSight system integration.	See *Chapter 4 Functional Integration Guidelines* on page 4-34
4. Verify your design using the execution testbench.	See *Chapter 10 Execution Testbench* on page 10-104
5. CoreSight system integration.	See *Appendix B CoreSight SoC* on page Appx-B-186 and *Appendix F TEALMCU* on page Appx-F-228

Chapter 4
Functional Integration Guidelines

This chapter describes the functional integration requirements of the macrocell in your SoC design.

It contains the following sections:

4.1 About functional integration

This section includes key information that you must consider before or during the integration of the processor into your SoC design.

4.2 Clocking and resets

To integrate the processor, you must to know how to use the processor clock in your SoC design, and how to connect and use the Cortex-M33 processor resets. You must also know how to reset different parts of the design independently.

The processor has one input clock signal, **CLKIN**, for all internal logic.

To minimize the dynamic power used by the processor, this clock is gated throughout the design according to the structural hierarchy of the design and the operating mode.

Note

You must ensure that all other input signals to the processor are synchronized to **CLKIN** using the appropriate circuit in the system unless explicitly stated in the tables in this chapter.

4.2.1 Clock and clock enable signals

The following table shows the Cortex-M33 processor clock and clock enable signals.

Table 4-1 Clock and clock enable signals

Signal name	Direction	Description	Connection information
CLKIN	Input	Primary processor clock. This is gated internally for functional units when required depending on the operating mode of the processor.	**CLKIN** must always be running, unless the processor is powered down.
CORECLKEN	Output	Core clock enable. Indicates whether the Core domain clock is gated or enabled.	**CORECLKEN** must be used to control the system clock that is associated with the AHB bus to ensure that it is always able to accept requests on C-AHB and S-AHB from the processor.
SSTCLKEN	Input	Synchronous enable that is used with **CLKIN** to derive the secure system SysTick clock.	If an asynchronous system clock is used, **SSTCLKEN** must be generated using an appropriate synchronizer circuit followed by an edge detector.
NSSTCLKEN	Input	Synchronous enable that is used with **CLKIN** to derive the Non-secure system SysTick clock.	If an asynchronous system clock is used, **NSSTCLKEN** must be generated using an appropriate synchronizer circuit followed by an edge detector.

4.2.2 Reset signals

The following table shows the Cortex-M33 processor reset signals.

Table 4-2 Reset signals

Signal name	Direction	Description	Connection information
nPORESET	Input	Powerup reset. Resets entire processor.	Must be asserted on powerup. This signal is synchronized with **CLKIN** inside the processor. This means **nPORESET** can be asserted and deasserted asynchronously in the system.
nSYSRESET	Input	The **nSYSRESET** signal resets the processor, except the debug logic, D-AHB interface, CTI, MTB, and ETM.	**nSYSRESET** can be asserted without **nPORESET**. It is not necessary to assert **nSYSRESET** on powerup. This signal is synchronized with **CLKIN** inside the processor. This means **nSYSRESET** can be asserted and deasserted asynchronously in the system.
SYSRESETREQ	Output	Request to assert **nSYSRESET**.	Connect to **nSYSRESET** control logic.

4.2.3 Reset modes

The reset signals present in the processor design enable you to reset different parts of the design independently.

The following table shows the reset modes, and the combinations and possible applications you can use.

Table 4-3 Reset modes

Reset mode	nSYSRESET	nPORESET	Application
Powerup reset	X	0	Reset on power up, full system reset. Known as Cold reset.
Processor reset	0	1	Processor core reset only, excluding debug logic. Known as Warm or Soft reset.

Note

- **nPORESET** resets a superset of the **nSYSRESET** logic.
- A processor or Warm reset initializes most of the processor logic that is associated with regular execution of software excluding NVIC debug logic, D-AHB interface, ETM, MTB, CTI, BPU, DWT, and ITM. This reset can be used for resetting a processor that has been operating for some time, for example, a watchdog reset.
- During normal operation **nPORESET** and **nSYSRESET** must be deasserted.
- Both **nPORESET** and **nSYSRESET** are synchronized to **CLKIN** inside the processor.

Reset and low-power support

The processor supports a number of internal power domains that the Q-channel interfaces, when connected to a *Power Management Unit* (PMU) in the system, can enable and disable. When a domain is enabled, the processor generates a local reset automatically to ensure correct operation. To prevent resets from clearing retained state the processor includes input signals that mask the power domain local reset operation.

Processor configuration at reset

The processor includes a number of configuration input signals that are asserted during reset. Some of these can only be set during powerup reset and others can be set at either powerup or processor reset.

Debug during reset

Cortex-M33 supports access to processor resources through a debug agent during reset and before software execution commences.

Software reset request

Software running on the processor can request a reset to the system using the AIRCR.SYSRESETREQ bit field. On the Cortex-M33 processor, writing to this field asserts the **SYSRESETREQ** output signal.

Resets and clocking during DFT scan

During scan testing, the assertion of the DFT input signals **DFTCGEN** and **DFTRSTDISABLE[1:0]** bypasses the reset synchronizers, clock gating cells, and **nSYSRESET**.

4.3 Static configuration signals

Descriptions of the configuration signals that are present on the processor, and how you must set the signals in your SoC design.

The configuration signals in the following table can only be changed at powerup reset, when **nPORESET** is asserted. They are intended to be static configuration signals that are fixed for a given integration of the processor.

Table 4-4 Static configuration signals

Signal name	Direction	Description	Connection information
CFGBIGEND	Input	Static endianness selection for data accesses. **0** Little-endian data. **1** BE8 big-endian data. This signal affects all data side memory interfaces identically except for the PPB region, which is always little-endian. Instructions fetches are always performed as little-endian.	Tie these signals in accordance with your requirements or processor configuration.
CFGSSTCALIB[25:0]	Input	Secure SysTick calibration configuration. **CFGSSTCALIB[23:0]** TENMS **CFGSSTCALIB[24]** SKEW **CFGSSTCALIB[25]** NOREF	
CFGNSSTCALIB[25:0]	Input	Non-secure SysTick calibration configuration. **CFGNSSTCALIB[23:0]** TENMS **CFGNSSTCALIB[24]** SKEW **CFGNSSTCALIB[25]** NOREF	
CFGFPU	Input	If configured, enables support for hardware floating-point	
CFGDSP	Input	If configured, enables support for ARMv8-M DSP extension.	
CFGSECEXT	Input	If configured, enables support for ARMv8-M Security Extension.	
MPUNSDISABLE	Input	If configured, disables support for the Non-secure MPU.	
MPUSDISABLE	Input	If configured, disables support for the Secure MPU.	
SAUDISABLE	Input	If configured, disables support for the SAU.	

4.3.1 SysTick signals

The ARMv8-M system timer, SysTick, is a system-agnostic timer implementation for operating system use. Software can configure the SysTick timer to select **CLKIN** as its clock source, or an alternative clock source.

An external reference source can control the SysTick system timers using the input enable signals **SSTCLKEN** and **NSSTCLKEN** for the Secure and Non-secure timers respectively. The enable signals must be synchronous to **CLKIN**. If you use an asynchronous system clock, you must generate **SSTCLKEN** and **NSSTCLKEN** using an appropriate synchronizer circuit followed by an edge detector.

The following table shows the **CFGSSTCALIB[25:0]** and **CFGNSSTCALIB[25:0]** values required for the software developer using the SysTick calibration registers in the NVIC memory map.

Table 4-5 CFGSSTCALIB and CFGNSSTCALIB signal encodings

Signal name	Mapping	Description
CFGSSTCALIB[25], **CFGNSSTCALIB[25]**	NOREF	Indicates whether the external reference clock is implemented: **0** Reference clock is implemented. **1** Reference clock is not implemented. When `0b1`, the CLKSOURCE bit of the SYST_CSR register is forced to `0b1` and cannot be cleared to `0b0`.
CFGSSTCALIB[24], **CFGNSTCALIB[24]**	SKEW	Tie this LOW if **CLKIN** or the external reference clock, as indicated by **CFGSSTCALIB[25]** **CFGNSSTCALIB[25]**, can guarantee an exact multiple of 10ms. Otherwise, tie this signal HIGH.
CFGSSTCALIB[23:0], **CFGNSSTCALIB[23:0]**	TENMS	Provides an integer value to compute a 10ms, 100Hz, delay from either the external reference clock, or **CLKIN** if the reference clock is not implemented. For example, apply the value `F423F999999` if no reference is implemented, and **CLKIN** is 100MHz.

For an implementation where no alternative reference clock is provided, and the frequency of **CLKIN** is not computable in hardware, tie:

Secure SysTick timer
- **SSTCLKEN** LOW.
- **CFGSSTCALIB[25]** HIGH.
- **CFGSSTCALIB[24:0]** LOW.

Non-secure SysTick timer
- **NSSTCLKEN** LOW.
- **CFGNSSTCALIB[25]** HIGH.
- **CFGNSSTCALIB[24:0]** LOW.

4.4 Reset configuration signals

Descriptions of the reset configuration signals that can only be changed under processor reset or at powerup reset. The reset configuration signals can be used more dynamically than the static configuration signals.

Signal name	Direction	Description	Connection information
INITSVTOR[31:7]	Input	Signals that set the vector base address bit field in the Secure Vector Table Offset Register, VTOR_S.TBLOFF[31:7], out of reset.	Tie these signals in accordance with your requirements or processor configuration.
INITNSVTOR[31:7]	Input	Signals that set the vector base address bit field in the Non-secure Vector Table Offset Register, VTOR_NS.TBLOFF[31:7], out of reset.	

4.5 Instruction execution control signals

Description of the instruction execution control signals, hints on how to use them, and their limitations.

Table 4-6 Instruction execution control signals

Signal name	Direction	Description	Connection information
CPUWAIT	Input	Stall the core out of reset. The **CPUWAIT** signal, when HIGH out of reset, forces the core into a quiescent state. The core boot-up sequence and instruction execution is delayed until this signal is driven LOW. During this time, the processor does not perform any memory accesses. Debugger accesses continue when this signal is HIGH. **CPUWAIT** has no effect if driven HIGH when the processor is running.	-
CODEHINT[2:0]	Output	Prefetch hints. **CODEHINT[2]** When HIGH, indicates that the next instruction fetch transaction is not going to be sequential. For example, the next instruction is to be an unconditional branch or there is an interrupt pending. **CODEHINT[1]** When HIGH, indicates that the following instruction to be executed is a currently unresolved conditional backward branch. **CODEHINT[0]** When HIGH, indicates that the following instruction to be executed is a currently unresolved conditional forward branch. The system can use these signals to control instruction prefetching. The system must always manage these signals as speculative hints. The processor does not guarantee that all non-sequential instruction fetches indicate in advance on **CODEHINT**.	-
IFLUSH	Input	Flush instructions fetched on C-AHB or S-AHB on the previous **HREADY** cycle. **IFLUSH** is held HIGH for one clock cycle. If the previous AHB transaction on either C-AHB or S-AHB was not an instruction fetch, this signal must be LOW. If the previous AHB instruction fetch transaction resulted in an error indicated by **HRESP**, this signal must be LOW.	-
CURRNS	Output	Current Security state of the Cortex-M33 processor: **HIGH** Processor is in Secure state. **LOW** Processor is in Non-secure state. If the Cortex-M33 processor is not configured for ARMv8-M Security Extension support, the **CURRNS** signal is LOW.	-

The processor supports prefetch hints using the **CODEHINT** signal. This signal indicates speculatively that a non-sequential fetch, from either a branch or a taken exception might take place before it is indicated on the C-AHB or S-AHB interface. A system can use this signal to optimize the behavior of a prefetch buffer or instruction cache.

The following table shows the **CODEHINT** encoding, together with the speculative limitations of the hint.

Table 4-7 CODEHINT signal limitations

<table>
<tr><th>Signal</th><th>Limitations</th></tr>
<tr><td>CODEHINT[2]</td><td>This signal is asserted for the following cases:
• New interrupt arrival.
• Unconditional direct branch not in an IT block.
• Indirect branches BX, BLX, BXNS, BLXNS, MOV PC, Rm, and not in an IT block.
• Dual issued BX LR, or B not in an IT block. This includes a dual issued B<cond> and B pair, because one of the instructions is guaranteed to result in a branch taken.
• SVC instruction not in an IT block.</td></tr>
<tr><td>CODEHINT[1]</td><td rowspan="2">These signals are only asserted for direct branches that include explicit condition codes in the instruction, B<cond> <label> for example.

These signals are not asserted for indirect branch instructions or unconditional direct branch instructions that are made conditional in an IT block.

If an unconditional direct branch is older than a dual issued conditional branch, CODEHINT[1:0] is not suppressed. When this occurs CODEHINT[2] must be examined to override this case.</td></tr>
<tr><td>CODEHINT[0]</td></tr>
</table>

4.6 Code and System AHB interfaces

The C-AHB interface and S-AHB interface are used for any instruction fetch and data access to regions of the memory map.

The C-AHB interface is used for any instruction fetch and data access to the Code region of the ARMv8-M memory map. For example, in a microcontroller system, you can connect flash memory to the interface.

The S-AHB interface is used for any instruction fetch and data access to the SRAM, peripheral, external RAM and external device regions of the memory map. It can also be used for data accesses to the Vendor_SYS region of the memory map where instruction fetches are not allowed.

Because the processor uses a Harvard architecture, instruction fetches and data accesses can happen in parallel to addresses associated with one of the interfaces. The internal bus matrix arbitrates access to the C-AHB and S-AHB interfaces when this occurs, with higher priority given to data read and write requests. Debug accesses from the D-AHB interface can also access this bus interface. Accesses from the processor have higher priority in the arbiter, than debug accesses on this bus, so debug accesses are waited until processor accesses have completed when there are simultaneous processor and debug access to the bus interface. The arbiter contains quality-of-service logic to ensure that debug accesses always complete eventually.

Cortex-M33 instruction fetches and data accesses always use INCR undefined length bursts. Debug Accesses, indicated by **HMASTER** set to 1, always use SINGLE bursts. **HSIZE[2]** is always 0 because the processor cannot issue memory accesses with size greater than 32-bits.

The processor provides hint signals for both data access and instruction fetches on the C-AHB and S-AHB interfaces to optimize the behavior of the system.

The **HHINTC** and **HHINTS** signals synchronous to the AHB address phase indicate whether a data access is associated with either an exception vector fetch or a load instruction with PC relative base address and also if the request was made from software running in Thread or Handler mode.

Table 4-8 C-AHB interface signals

Signal name	Direction	Description	Connection Information
HADDRC[31:0]	Output	Transfer address.	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HBURSTC[2:0]	Output	Transfer burst length.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HEXCLC	Output	Exclusive request. Address phase control signal that indicates whether an access is because of an `LDREX` or `STREX` instruction: **0** Non-exclusive, standard, transaction. **1** Exclusive transaction.	To support exclusive transfers on the C-AHB interface, connect this signal to a global exclusive monitor, otherwise leave it unconnected.
HEXOKAYC	Input	Exclusive response. Data phase signal that is sampled on **HREADYC** that indicates whether the exclusive request was granted or not: **0** Exclusive access has failed. **1** Exclusive access is successful.	To support exclusive transfers to shared memory on the C-AHB interface, connect this signal to a global exclusive monitor, otherwise tie it LOW.

Table 4-8 C-AHB interface signals (continued)

Signal name	Direction	Description	Connection Information
HHINTC[2:0]	Output	Hint signal. This signal is synchronous to **HTRANSC[1:0]**. **HHINTC[0]** Indicates that a read access is to an exception vector. **HHINTC[1]** Indicates that a PC relative load instruction has generated a read access. **HHINTC[2]** Indicates whether the read or write access was associated with Handler or Thread mode. 1 indicates access is a Handler mode fetch or read/write in Handler mode. 0 indicates access is a Thread mode fetch or read/write in Thread mode.	-
HINNERC[4:0]	Output	Inner memory attributes using the same format as **HPROTC[6:2]**.	-
HMASTERC	Output	Initiator of the Transfer. **0** Processor. **1** Debugger.	-
HNONSECC	Output	Security level, asserted to indicate a Non-secure transfer.	-
HPROTC[6:0]	Output	Protection and outer memory attributes.	-
HRDATAC[31:0]	Input	Read data.	-
HREADYC	Input	Slave ready.	-
HRESPC	Input	Slave response.	-
HSIZEC[2:0]	Output	Transfer size.	-
HTRANSC[1:0]	Output	Transfer type.	-
HWDATAC[31:0]	Output	Write data.	-
HWRITEC	Output	Write transfer.	-

Table 4-9 HEXOKAYC with an implemented global monitor

Transaction properties		Required HEXOKAYC
HEXCLC	**Load/Store**	
LOW	-	X. Not an exclusive access and so the **HEXOKAYC** signal has no effect.
HIGH	Load	HIGH if a global exclusive monitor is implemented that covers the access address. Otherwise LOW.
HIGH	Store	HIGH if a global exclusive monitor is implemented that covers the access address and the exclusive check succeeds. Otherwise LOW.

Note

- Your software must avoid exclusive accesses to shared regions of memory unless you have implemented a global exclusive monitor that covers the region in question. The processor treats such accesses as an error condition and takes a BusFault exception if a load is performed with **HEXCLC** HIGH and receives **HEXOKAYC** LOW.
- **HEXOKAYC** is ignored if an ERROR response is returned on **HRESPC**.

Table 4-10 S-AHB interface signals

Signal name	Direction	Description	Connection Information
HADDRS[31:0]	Output	Transfer address.	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HBURSTS[2:0]	Output	Transfer burst length.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HEXCLS	Output	Exclusive request. Address phase control signal that indicates whether an access is because of an `STREX` or `LDREX` instruction: **0** Non-exclusive, standard, transaction. **1** Exclusive transaction.	To support exclusive transfers on the S-AHB interface, connect this signal to the global exclusive monitor, otherwise leave it unconnected.
HEXOKAYS	Input	Exclusive response. Data phase signal sampled on **HREADYS** that indicates whether the exclusive request was granted or not: **0** Exclusive access has failed. **1** Exclusive access is successful.	To support exclusive transfers to shared memory on the S-AHB interface, connect this signal to a global exclusive monitor, otherwise tie it LOW.
HHINTS[2:0]	Output	Hint signal. This signal is synchronous to **HTRANSS**. **HHINTS[0]** Indicates that a read access is to an exception vector. **HHINTS[1]** Indicates a PC relative load instruction generates a read access. **HHINTC[2]** Indicates whether the read or write access was associated with Handler or Thread mode. 1 indicates access is a Handler mode fetch or read/write in Handler mode. 0 indicates access is a Thread mode fetch or read/write in Thread mode.	Connect masters through the bus infrastructure.
HINNERS[4:0]	Output	Inner memory attributes using the same format as **HPROTS[6:2]**.	-
HMASTERS	Output	Initiator of the access: **0** Processor. **1** Debugger.	-
HNONSECS	Output	Security level, asserted to indicate a Non-secure transfer.	-
HPROTS[6:0]	Output	Protection and outer memory attributes.	-
HRDATAS[31:0]	Input	Read data.	-
HREADYS	Input	Slave ready.	-
HRESPS	Input	Slave response.	-
HSIZES[2:0]	Output	Transfer size.	-
HTRANSS[1:0]	Output	Transfer type.	-
HWDATAS[31:0]	Output	Write data.	-
HWRITES	Output	Write transfer.	-

Table 4-11 HEXOKAYS with an implemented global monitor

Transaction properties		Required HEXOKAYS
HEXCLS	**Load/Store**	
LOW	-	X. Not an exclusive access and so the **HEXOKAYS** signal has no effect.
HIGH	Load	HIGH if a global exclusive monitor is implemented that covers the access address. Otherwise LOW.
HIGH	Store	HIGH if a global exclusive monitor is implemented that covers the access address and the exclusive check succeeds. Otherwise LOW.

Note

- Your software must avoid exclusive accesses to shared regions of memory unless you have implemented a global exclusive monitor that covers the region in question. The processor treats such accesses as an error condition and takes a BusFault exception if a load is performed with **HEXCLS** HIGH and receives **HEXOKAYS** LOW.
- **HEXOKAYS** is ignored if an ERROR response is returned on **HRESPS**.

4.7 D-AHB interface

The D-AHB slave is a 32-bit AMBA AHB that enables Cortex-M33 to perform external debug interaction.

The interface is designed for integration with a CoreSight AHB-AP. It provides:

- A debugger with access to all processor control and debug resources.
- A view of memory that is consistent with software load and store operations.

Debugger accesses are distributed to the appropriate internal and external resource according to the address of the request. Access on D-AHB is reflected on the C-AHB, S-AHB, or EPPB interfaces appropriately.

D-AHB accesses are always little-endian.

Note

D-AHB accesses:

- To the PPB and EPPB memory regions return an error if they are not marked as privileged, **HPROTD[1]** must be HIGH.
- Are not subject to MPU protection checks, however they are subject to security attribution and protection checks.

The following determine the security of a debug transaction:

- The debug access control signals. See *4.7.2 Debug access control* on page 4-50.
- The mapping of the address in the SAU or IDAU.
- The internal debug state of the processor in DHCSR.S_SDE.
- The **HNONSECD** signal value that is associated with the D-AHB debug request.

Cortex-M33 supports a separate power domain for most of the debug logic in the processor. If this domain is inactive when a debug access is made on D-AHB, the processor requests the domain to be automatically powered up before completing the request.

The following table shows the signals for the D-AHB interface.

Table 4-12 D-AHB interface signals

Signal name	Direction	Description	Connection information
HTRANSD[1:0]	Input	Indicates the type of current transfer. **Note** **HTRANSD[0]** is ignored by the processor, all transactions are treated as either Non-sequential or Idle.	Connect to a *Debug Access Port* (DAP) AHB-AP
HBURSTD[2:0]	Input	Indicates whether the transfer is part of a burst. For debug accesses, transfers appear as SINGLE, tied to `0b000`.	
HADDRD[31:0]	Input	32-bit transfer address bus.	
HWRITED	Input	Write transfer.	
HSIZED[2:0]	Input	Indicates the size of the access. Accesses can be: `0b000` Byte. `0b001` Halfword. `0b010` Word. **Note** **HSIZED[2]** is ignored by the processor.	
HWDATAD[31:0]	Input	Data write bus.	
HPROTD[6:0]	Input	Protection and outer memory attributes. Provides information on the access. **Note** **HPROTD[0]** is ignored by the processor, all debug transactions are treated as data accesses.	
HNONSECD	Input	Debug access security level request. When asserted, **HNONSECD** indicates a Non-secure transfer. The resultant security level of the debug access depends on the debug control registers in the processor and the debug access control input signals to the processor.	
HREADYD	Output	When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.	
HRDATAD[31:0]	Output	Read data.	
HRESPD	Output	The transfer response status: **LOW** `OKAY`. **HIGH** `ERROR`.	

4.7.1 Debugger access and attributes

The attributes that perform a debugger access are derived from the **HPROTD** value of the transaction. The system uses attributes differently depending on the interface that the D-AHB access is mapped to.

The following table shows how to use the **HPROTD** attributes for all interfaces.

Table 4-13 HPROTD attributes

Interface	Description
C-AHB	**HPROTD[0]** ignored. All debugger accesses are performed with **HPROTC[0]** is 1. **HPROTD[6:1]** passed through to C-AHB. **HMASTERC** is asserted indicating a debugger access.
S-AHB	**HPROTD[0]** ignored. All debugger accesses are performed with **HPROTS[0]** is 1. **HPROTD[6:1]** passed through to S-AHB. **HMASTERS**is asserted indicating a debugger access.
Internal PPB	**HPROTD[0]** ignored. **HPROTD[1]** used for register-specific privilege checks. **HPROTD[6:2]** ignored. **PADDR31** is asserted indicating a debugger access. Unprivileged D-AHB accesses to privileged registers returns **HRESP** = `ERROR`.
External PPB (EPPB)	**HPROT[0]** ignored. All debugger accesses are performed with **PPROT[2]** is 0. **HPROT[1]** passed through to **PPROT[0]**. **PADDR31** is asserted indicating debugger access.

In typical devices, it is expected that Device regions are software-context independent and a debugger can reliably accesses these regions. For such devices, the debugger is strongly advised to:

- Perform all accesses to Device regions with **HPROTD[6:2]** = `0b00000`, Device-nE.
- Perform all other accesses with **HPROTD[6:2]** = `0b00010`, Normal Write-Through.

Using Write-Through attributes for Normal memory guarantees debug reads and writes are always visible to software running on the processor even when there is an external cache in the system.

4.7.2 Debug access control

The **DBGEN**, **NIDEN**, **SPIDEN**, and **SPNIDEN** signals debug and trace control access to Secure state and Secure memory.

The following tables show the effect of the signals for invasive debug and for non-invasive debug.

If ARMv8-M Security Extension is not configured in the processor, then the **SPIDEN** and **SPNIDEN** signals are not used because all state and memory is treated as Non-secure.

Table 4-14 Invasive debug access control

External signals		Invasive debug status	Invasive debug permitted states
DBGEN	SPIDEN		
Low	X	Disabled	None
High	Low	Enabled	All Non-secure states
	High		All states

Table 4-15 Non-invasive debug access control

External signals				States in which non-invasive debug is permitted
DBGEN	**NIDEN**	**SPIDEN**	**SPNIDEN**	
Low	Low	X	X	None
	High	Low	Low	All Non-secure states
			High	All States
		High	X	
High	X	Low	Low	All Non-secure states
			High	All states
		High	X	

Related references

4.7.1 Debugger access and attributes on page 4-49.

4.8 External Private Peripheral Bus

The EPPB is a 32-bit AMBA4 APB interface.

See the *AMBA® APB Protocol Version 2.0 Specification*.

The EPPB provides data accesses to the memory region `0xE0044000-0xE00FEFFF`. Instruction accesses to this region are not permitted.

The interface supports:
- Little-endian accesses. The endianness configuration of the processor is ignored.
- All accesses are treated as Device.
- Exclusive accesses are not supported.
- Aligned accesses. Unaligned accesses are UNPREDICTABLE.
- Privileged accesses. Unprivileged accesses take a BusFault exception.

The EPPB can perform debugger-initiated transactions while the processor is in Soft reset.

The following table shows the signals for the APB interface. Only the address bits necessary to decode the EPPB space are supported on this interface.

Table 4-16 EPPB signals

Signal name	Direction	Description	Connection information
PSEL	Output	APB device select. Indicates that a data transfer is requested.	Connect to your system CoreSight components as required. **Note** ARM recommends that all non-debug peripherals are integrated on the S-AHB interface.
PENABLE	Output	APB control signal. Strobe to time all accesses. Indicates the access phase of an APB transfer.	
PPROT[2:0]	Output	Transfer privilege and security level.	
PWRITE	Output	APB transfer direction. Write not read.	
PADDR[19:2]	Output	APB 18-bit Address bus. Only the bits that are relevant to the External Private Peripheral Bus are driven.	
PADDR31	Output	Initiator of the transfer. This signal is driven HIGH when the DAP is the requesting master. It is driven LOW when the processor is the requesting master.	
PWDATA[31:0]	Output	APB 32-bit write data bus.	
PREADY	Input	APB slave ready signal. This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.	
PSLVERR	Input	APB slave error signal. This signal is driven HIGH if the currently accessed APB device cannot handle the requested transfer.	
PRDATA[31:0]	Input	APB 32-bit read data bus.	

4.9 External coprocessor interface

The external coprocessor interface allows the integration of tightly coupled accelerator hardware with the processor, and to communicate with it in software using architectural coprocessor instructions.

The external coprocessor interface:

- Supports up to eight separate coprocessors, CP0-CP7. The remaining coprocessor numbers, CP8-CP15, are reserved. CP10 and CP11 are always reserved for hardware floating-point. See the *ARM®v8-M Architecture Reference Manual* for more information.
- Supports low-latency data transfer from the processor to and from the accelerator components.
- Has a sustained bandwidth up to twice that of the processor memory interface.

Operation

The interface provides the external devices with information about the:

- Processor privilege and security state.
- Instruction type and associated register and coprocessor-specific opcode fields that are defined by the architecture.

The following instruction types are supported:

- Register transfer from the Cortex-M33 processor to a coprocessor `MCR`, `MCRR`, `MCR2`, `MCRR2`.
- Register transfer from the coprocessor to a Cortex-M33 processor `MRC`, `MRRC`, `MRC2`, `MRRC2`.
- Data processing instructions `CDP`, `CDP2`.

Note

The regular and extension forms of the coprocessor instructions, `MRC` and `MCR2` for example, have the same functionality, but different encodings.

The `MRC` and `MRC2` instructions support the transfer of APSR.NZCV flags when the processor register field is set to PC, for example `Rt` = `0xF`.

The interface also provides:

- A handshake mechanism to indicate to the coprocessor that an instruction committed in the processor pipeline cannot be interrupted.
- The ability for a coprocessor to stall the processor in a way that can always be interrupted, sometimes called a BUSYWAIT, and also to indicate that an error has occurred pending an UNDEFINSTR UsageFault.

Usage restrictions

The interface includes some restrictions in the use of coprocessor instructions:

- The `LDC(2)>` or `STC(2)` instructions are not supported. If these are included in software with the <coproc> field set to a value between 0 and 7 and the coprocessor is present and enabled in the appropriate fields in the CPASR/NSACR registers the Cortex-M33 processor always attempts to take a UNDEFINSTR UsageFault exception.
- The processor register field(s) for data transfer instructions should not include the stack pointer (Rt = 0xD), this encoding is UNPREDICTABLE in the ARM v8-M architecture and will result in a UNDEFINSTR UsageFault exception in Cortex-M33 if the coprocessor is present and enabled in the CPASR/NSACR registers.
- If any coprocessor instruction is executed when the corresponding coprocessor is either not present or disabled in the CPACR/NSACR register the Cortex-M33 processor will always attempt to take a NOCP UsageFault exception.

Data transfer rates

The following table shows the ideal data transfer rates for the coprocessor interface. This means that the coprocessor responds immediately and does not set BUSYWAIT.

The ideal data transfer rates are sustainable if the corresponding coprocessor instructions are executed back-to-back.

Table 4-17 Ideal data transfer rates for the coprocessor interface

Instructions	Direction	Ideal data-rate
`MCR`, `MCR2`	Processor to coprocessor	32 bits per cycle
`MRC`, `MRC2`	Coprocessor to processor	32 bits per cycle
`MCRR`, `MCRR2`	Processor to coprocessor	64 bits per cycle
`MRRC`, `MRRC2`	Coprocessor to processor	64 bits per cycle

External coprocessor Interface signals

The following table lists the coprocessor interface signals.

Table 4-18 External coprocessor Interface signals

Signal name	Direction	Description	Connection information
CPENABLED[7:0]	Output	Indicates which coprocessor is enabled in the: • *Coprocessor Access Control Register* (CPACR) associated with the security state of the processor. • *Non-Secure Access Control Register* (NSACR) register if the processor is executing in Non-secure state. **Note** The CPACR is banked when the implementation includes the ARMv8-M Security Extension. See *CPENABLED* on page 4-56.	Connect to the external coprocessors.
CPPWRSU[7:0]	Output	Indicates which coprocessors are permitted to become UNKNOWN. See *CPPWRSU* on page 4-57.	
CPSPRESENT[7:0]	Input	Indicates which Secure coprocessors are present in the system. See *CPSPRESENT, CPNSPRESENT* on page 4-57.	
CPNSPRESENT[7:0]	Input	Indicates which Non-secure coprocessors are present in the system. See *CPSPRESENT, CPNSPRESENT* on page 4-57.	
CPCDP	Output	Coprocessor command operation. See *CPCDP, CPMCR, CPMRC* on page 4-57.	
CPMCR	Output	Coprocessor register transfer from processor operation. See *CPCDP, CPMCR, CPMRC* on page 4-57.	
CPMRC	Output	Coprocessor register transfer to processor operation. See *CPCDP, CPMCR, CPMRC* on page 4-57.	
CPSIZE	Output	Coprocessor size operation. See *CPSIZE* on page 4-57.	
CPNUM[2:0]	Output	Coprocessor number request. See *CPNUM* on page 4-57.	
CPREGS[11:0]	Output	Operation register fields. See *CPREGS* on page 4-58.	
CPOPC[8:0]	Output	Operation opcode fields. See *CPOPC* on page 4-58.	
CPPRIV	Output	Indicates operation privilege. See *CPPRIV, CPNSATTR* on page 4-58.	
CPNSATTR	Output	Indicates operation security state. See *CPPRIV, CPNSATTR* on page 4-58.	
CPVALID	Output	Indicates whether the coprocessor operation is valid. See *CPVALID* on page 4-59.	
CPREADY	Input	Indicates whether the coprocessor is stalled or ready. See *CPREADY* on page 4-59.	
CPERROR	Input	Indicates that the coprocessor is not present or the instruction is not supported. See *CPERROR* on page 4-59.	
CPWDATA[63:0]	Output	The coprocessor write data bus. See *Data signals* on page 4-59.	
CPRDATA[63:0]	Input	The coprocessor read data bus. See *Data signals* on page 4-59.	

Configuring which coprocessors are included in Secure and Non-secure state

The system can configure which coprocessors are included in Secure and Non-secure state using the input signals **CPSPRESENT[n]** and **CPNSPRESENT[n]** where **n** is 0-7. These signals are read at powerup reset by the CPACR and NSACR registers where software discovers which coprocessors are present.

The following table shows the relationship between the input signals and the access control registers.

Table 4-19 Behavior CPACR and NSACR relative to CPSPRESENT[7:0] and CPNSPRESENT[7:0] at reset

CPSPRESENT[n]	CPNSPRESENT[n]	CPACR[2n+1:2n]		NSACR[n]
		Secure	Non-secure	
0	0	RAZ/WI	RAZ/WI	RAZ/WI
0	1	RAZ/WI	RAZ/WI	RAZ/WI
1	0	RW, reset to 0	RAZ/WI	RAZ/WI
1	1	RW, reset to 0	RW, reset to 0	Unknown

For coprocessors that should only be available to Secure software, setting **CPSPRESENT[n]** HIGH and **CPNSPRESENT[n]** LOW causes the corresponding coprocessor to be non-detectable or useable from Non-secure state.

———— **Note** ————

Setting {**CPSPRESENT[n]**, **CPNSPRESENT[n]**} to {0,1} is treated the same as setting the signals to {0,0}.

————————————

If the Cortex-M33 processor is not configured to support the ARMv8-M Security Extension, **CPSPRESENT[n]** and **CPNSPRESENT[n]** must both be asserted to indicate to the processor that coprocessor n is present in the system.

Accessing a non-existent coprocessor results in a NOCP UsageFault.

Coprocessor configuration and enable signals

The interface provides four signals to allow both the processor and coprocessor to determine the system configuration and the programmers model status of the coprocessors, **CPENABLED**, **CPPWRSU**, **CPSPRESENT** and **CPNSPRESENT**.

CPENABLED

This signal is output from the processor. **CPENABLED[n]** indicates that the corresponding coprocessor is enabled in the current CPACR register associated with the security state of the processor. If the current state is Non-secure, the corresponding coprocessor is also enabled NSACR register.

———— **Note** ————

CPACR is banked when the ARMv8-M Security Extension is included.

————————————

CPENABLED[n] can be used by the coprocessor to switch in and out of low-power state, for example, it can control a clock-gate.

The following properties hold for the signal:
- **CPENABLED[n]** implies CPACR[2n+1:2n] != `0x0`.
- **CPVALID** to coprocessor n is never asserted when **CPENABLED[n]** is LOW.
- The processor can still make speculative requests on **CPCDP**, **CPMCR**, **CPMRC** for coprocessor n when **CPENABLED[n]** is LOW.

CPPWRSU

This signal is output from the processor. **CPPWRSU[n]** indicates that the state associated with the corresponding coprocessor is permitted to become UNKNOWN. This signal is a reflection of the CPPWR.SUn bit-fields, see the *ARM®v8-M Architecture Reference Manual*. When **CPPWRSU[n]** is:

0

The state associated with coprocessor n is not permitted to become UNKNOWN.

1

The state that is associated with coprocessor n is permitted to become UNKNOWN. This can be used as a hint to power control logic that the coprocessor may be powered down. Accesses to the coprocessor generate a NOCP UsageFault.

If an external coprocessor is not present in the system as specified by **CPSPRESENT** and **CPNSPRESENT**, then the corresponding bit of **CPPWRSU** is zero.

CPSPRESENT, CPNSPRESENT

These signals are input to the processor and are captured at powerup reset. The combination of **CPSPRESENT[n]** and **CPNSPRESENT[n]** indicate that corresponding coprocessor is present in the system and supported in either Secure state only or in both Secure and Non-secure state. The following properties hold for the signal:

- If **CPSPRESENT[n]** is LOW writes to CPACR[2+1:2n], are ignored. NSACR[n] behaves as RAZ/WI.
- If **CPNSPRESENT[n]** is LOW writes to the Non-secure CPACR[2+1:2n], are ignored. NSACR[n] behaves as RAZ/WI.
- If **CPNSPRESENT[n]** is LOW any attempt to execute a committed coprocessor instruction for coprocessor n in Non-secure state results in a NOCP UsageFault exception.
- **!CPSPRESENT[n]** implies **!CPENABLED[n]**.
- **!CPNSPRESENT[n]** implies **!CPENABLED[n]** when the processor is in Non-secure state.

Speculative operation signals

The processor indicates to the interface when a coprocessor instruction is speculatively executing in the pipeline using the following signals.

CPCDP, CPMCR, CPMRC

These signals indicate the class of the instruction, either a data processing operation, or register transfer. The three signals are always mutually exclusive.

CPSIZE

This signal indicates whether register transfer instructions use 32-bit or 64-bit data. When **CPSIZE** is combined with the **CPMCR** and **CPMRC** signals, **CPSIZE** indicates that the instruction is `MCRR` or `MRRC` respectively. **CPSIZE** is never asserted with the **CPCDP** signal.

CPNUM

This signal indicates the coprocessor number used in the instruction between 0 and 7.

CPPRIV, CPNSATTR

These signals indicate the privilege and security state of the processor when the instruction executed. These can be used to control access to functionality in the coprocessor to a certain level of privilege or security, or to provide a mechanism for banking state.

The following table shows examples of speculative operation signal encoding.

Table 4-20 Examples of coprocessor instruction encoding on the speculative operation signals

Instruction	{CPCDP, CPMCR, CPMRC}	CPSIZE	CPNUM
`CDP p0`	`0b100`	`0b0`	`0b000`
`MCR p7`	`0b010`	`0b0`	`0b111`
`MRRC p4`	`0b001`	`0b1`	`0b100`

CPREGS

This signal encodes the instruction coprocessor register fields `CRm`, `CRd` and `CRn`. The following table shows the register fields and the associated instruction types.

If the coprocessor instruction does not support a particular field the corresponding signal is UNKNOWN.

Table 4-21 CPREGS encoding for coprocessor instruction types

Field	Content	Instruction type supported
CPREGS[3:0]	`CRm`	`CDP`, `MCR`, `MRC`, `MCRR`, `MRRC`
CPREGS[7:4]	`CRd`	`CDP`
CPREGS[11:8]	`CRn`	`CDP`, `MCR`,`MRC`

CPOPC

This signal encodes the instruction coprocessor opcode fields, opc1 and opc2 and the extension space (‘2’ variants of the instructions). The following table shows the coprocessor instruction types and the encoding fields that are available for them. If the coprocessor instruction does not support a particular field the corresponding signal is UNKNOWN.

Table 4-22 CPOPC encoding for coprocessor instruction types

Instruction	CPOPC[8]	CPOPC[7:4]	CPOPC[3:0]
`CDP`	Select extension space	opc1	{opc2, UNKNOWN}
`MCR`, `MRC`		{opc1, UNKNOWN}	{opc2, UNKNOWN}
`MCRR`, `MRRC`		UNKNOWN	opc1

The following table shows example encodings for coprocessor instructions. An x in the encoding indicates that the field is UNKNOWN.

Table 4-23 Example encodings for CPREGS and CPOPC

Instruction	CPREGS	CPOPC
`CDP p1, 0, CR1, CR2, CR3, 7`	`0b001000010011`	`0b00000111x`
`MCRR p5, 1, Rt, Rt2, CR7`	`0bxxxxxxxx1111`	`0b0xxxx0001`
`MRC2 p7, 2, Rt, CR0, CR1, 0`	`0b0000xxxx0001`	`0b1010x000x`

Handshake signals

The interface uses a set of handshake signals to indicate that an instruction is committed by the processor and whether the coprocessor responds with a stall request or an error. If the handshake indicates a successful transaction, then any associated data must be driven in the next clock cycle.

CPVALID

The processor asserts **CPVALID** in the same cycle as the speculative operation signals when the associated coprocessor instruction is:

- Using a coprocessor number that is enabled in the CPACR, and the NSACR, when the ARMv8-M Security Extensions are included.
- Non-speculative.
- Not stalled in the processor pipeline.
- Outside an IT block or passes the conditions of an IT block.

The signal obeys the following properties:

- The transaction is committed if and only if **CPVALID** && **CPREADY**.
- **CPVALID** asserted implies {**CPCDP**, **CPMCR**, **CPMRC**} is one hot.
- **CPVALID** can be retracted if the coprocessor is busy and an interrupt with sufficient priority to preempt the current process becomes pending.

CPREADY

This signal is asserted by the coprocessor to indicate the operation is either complete, or ready for data transfer in the following clock cycle. If this signal is not asserted and **CPVALID** is asserted then the processor stalls waiting for the coprocessor, as long as no interrupt becomes pending.

CPERROR

This signal is asserted by the coprocessor together with **CPREADY** to indicate the speculative operation is not supported.

The coprocessor response signals **CPREADY** and **CPERROR** should not depend on the **CPVALID** signal, only on the operation signals.

The following lists the handshake encodings for the **CPVALID**, **CPREADY** and **CPERROR** signals.

Table 4-24 Coprocessor handshake response signals

Request signal	Response signals	Behavior
CPVALID	**CPREADY** && !**CPERROR**	Transaction is accepted
!**CPVALID**		Speculative instruction, no effect
CPVALID	!**CPREADY** && !**CPERROR**	Coprocessor is busy
!**CPVALID**		Speculative instruction, no effect
CPVALID	**CPREADY** && **CPERROR**	Coprocessor cannot support the request
!**CPVALID**		Speculative instruction, no effect
-	!**CPREADY** && **CPERROR**	Illegal response

Data signals

The interface includes two 64-bit data signals, **CPRDATA** and **CPWDATA**, for transferring data to and from the Cortex-M33 processor to a coprocessor. These signals are always valid one clock cycle after a handshake for a committed data transfer transaction is completed, indicated by **CPVALID** && **CPREADY** && !**CPERROR**.

For `MCR` and `MRC` instructions only the lower 32 bits of the signals are used, the upper 32 bits are UNKNOWN. The `MCRR` and `MRRC` instructions use all of the 64-bit data signals. The `CDP` instructions do not use data signals and all 64 bits are therefore UNKNOWN in the cycle following the completion of a handshake. If a transaction handshake indicates the response from the coprocessor is BUSYWAIT or error, on the following clock cycle the data signals are also UNKNOWN.

Interface timing

For the speculative operation signals, the coprocessor interface uses combinatorial protocol for the coprocessor response. To allow the external hardware enough time to evaluate the operation the processor drives the speculative signals early in the clock cycle. The **CPVALID** signal is driven later in the cycle because it requires the commit status of the instruction to be evaluated.

ARM recommends that you tightly integrate coprocessors into the processor sub-block of the system, and implement as a single entity in synthesis.

Example transactions

The following figure shows an example data processing operation transaction containing `MCR`, `CDP`, and `MRC` instructions carried out over the interface to an accelerator-like coprocessor.

t0 and t1 Two `MCR` instructions are executed to transfer data from the Cortex-M33 processor.

t2 A `CDP` is sent to request the coprocessor to execute a data processing operation.

t3 An `MRC` is executed to read the result back from the coprocessor.

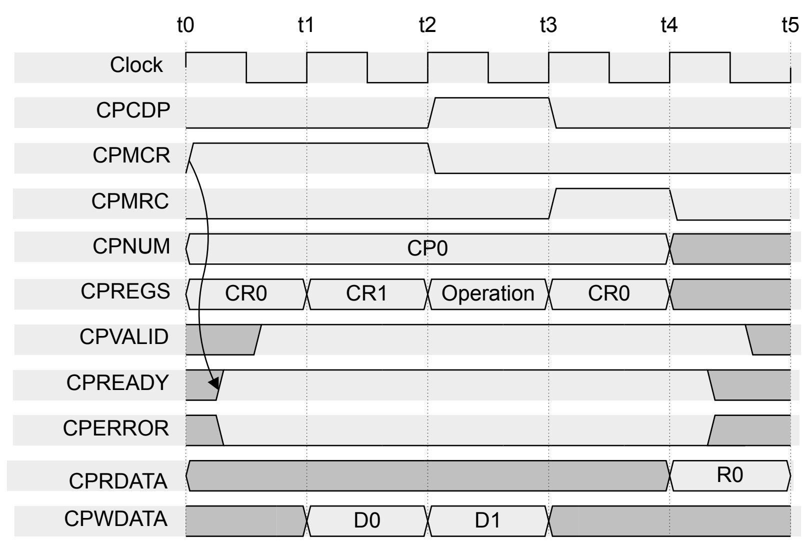


Figure 4-1 Example transaction containing MCR, CDP, and MRC instructions

The following figure shows a more complex example of the interface protocol. At:

t0 **CPMCR** is asserted HIGH, indicating a speculative `MCR` operation from the processor. **CPVALID** is held LOW, indicating the operation is not committed.

t1 **CPMCR** is HIGH again and this time **CPVALID** is also HIGH indicating the operation is committed. The coprocessor is not ready to accept the speculative `MCR` operation, indicated by **CPREADY** held LOW.

t2 With both **CPVALID** and **CPREADY** asserted HIGH, the speculative `MCR` operation is committed to coprocessor.

t3
- **CPWDATA** transfers data for the MCR operation committed in t2.
- The processor commits an `MRC` operation that is not supported by the coprocessor. The **CPERROR** signal is asserted terminating the transaction with no data phase.

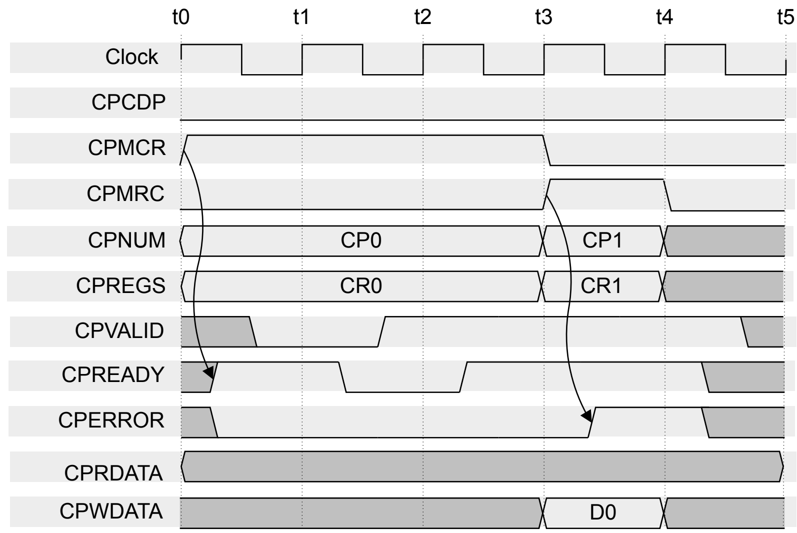


Figure 4-2 Example transaction including coprocessor BUSYWAIT and error response

Support for multiple coprocessors

Multiple coprocessors can be connected to the Cortex-M33 processor by multiplexing the handshake and data bus signals.

Handshake signal integration

The handshake signals from the individual coprocessors can be combined as follows, where **<SIGNAL>s[]** indicates a vector of the response signals from all the coprocessors:

- **CPREADY** = |**CPREADYs**[].
- **CPERROR** = |**CPERRORs**[].
- !**CPREADYs[n]** || **CPERRORs[n]** implies (**CPNUM** == **n**).
- (**CPNUM** != **n**) implies **CPREADYs[n]** && !**CPERRORs[n]**.

Data bus selection

The read data bus signal can be combined by registering **CPNUM** when a transaction is committed and then using the registered value to select between the coprocessors, for example. **CPRDATA** = **CPRDATAs**[**CPNUM_q**].

The following figure shows an example of an integration of two coprocessors to the Cortex-M33 processor.

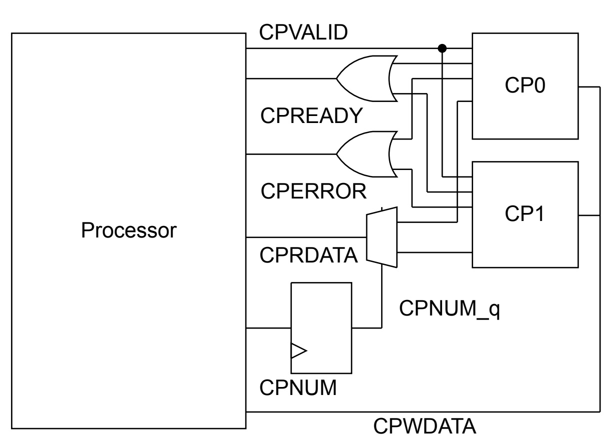


Figure 4-3 Example integration of two coprocessors to the Cortex-M33 processor

Debug access to coprocessor registers

The coprocessor interface does not support a mechanism to read and write registers located in external coprocessors.

ARM recommends you implement a coprocessor with a dedicated AHB or APB slave interface for the system to access the registers. If the debug view of the coprocessor is located in the PPB region of the memory map, you can use this coprocessor interface to connect to the EPPB interface of the Cortex-M33 processor.

If Secure debug is disabled, you must ensure the Secure information in the coprocessor is protected and not accessible when using a Non-secure debugger.

If the debug slave interface to the coprocessor is connected to the processor C-AHB or S-AHB master interfaces or the EPPB interface, you can use the **HNONSEC** and **PPROT[2]** signals on the AHB and APB interfaces respectively. This is because the security level of the debug requests routed through the processor from the D-AHB interface are subject to the debug access and authentication checks.

If coprocessor state is memory-mapped then software can also access the information using load and store instructions. If your implementation uses this functionality, you must ensure the appropriate barrier instructions are included to guarantee ordering between coprocessor instructions and load/store operations to the same state.

Exceptions and context switch

The interface does not include support for automatic save and restore of coprocessor registers on entry and exit to exceptions, unlike the internal processor integer and floating-point registers. Any coprocessor state that must be maintained across a context switch must be carried out by the software that is aware of the coprocessor requirements.

You must ensure that when the coprocessor contains Secure state it is not accessible by Non-secure exception handlers.

Response to coprocessor errors

The coprocessor must not rely on a synchronous exception being taken when asserting a **CPERROR** response to a coprocessor transaction because the UNDEFINSTR UsageFault might be preempted by a higher priority interrupt in the processor. The processor does guarantee that there are no side effects from the erroneous instruction.

Hazard between load and store instructions followed by coprocessor transactions

To decouple the data-side AHB **HREADY** input signal from the **CPVALID** output signal, a coprocessor instruction following a load/store instruction in the processor pipeline always stalls for a clock cycle when **HREADY** is asserted and the load/store completes. This does not add any additional stall cycles to the data hazard already included in the case where the result of a load is consumed by a coprocessor data transfer instruction for example,`LDR Rd, [X]; MCR Rd`.

4.10 Debug signals

Descriptions of the debug interface signals that are present on the processor, and how you must set the signals in your SoC design are given in this section.

Table 4-25 Debug signals

Signal name	Direction	Description	Connection information
HALTED	Output	In halting mode debug. **HALTED** remains asserted while the processor is in debug.	Connect to external CoreSight CTI when an internal CTI is not implemented.
DBGRESTART	Input	Request for synchronized exit from halt mode. Forms a handshake with **DBGRESTARTED**. If multiprocessor debug support is not required, **DBGRESTART** must be tied LOW.	
DBGRESTARTED	Output	Handshake for **DBGRESTART**.	
EDBGRQ	Input	External debug request. A debug agent in the system asserts this signal to request that the processor enters Debug state.	
DBGEN	Input	Invasive debug enable. When LOW, disables all halt-mode and invasive debug features.	Tie-off or connect to debug authentication module.
NIDEN	Input	Non-invasive debug enable. When LOW, disables all trace and non-invasive debug features.	Either tie HIGH or connect to debug authentication module.
SPIDEN	Input	Secure invasive debug enable. When LOW, disables all halt mode and invasive debug features when the processor is in Secure state.	Tie-off or connect to debug authentication module.
SPNIDEN	Input	Secure non-invasive debug enable. Controls access to non-invasive debug features when the processor is in Secure state and **SPIDEN** is LOW.	Tie-off or connect to debug authentication module.
CTICHIN[3:0]	Input	CTI channel input.	Either connect to **CTICHOUT** of system level CTI or CTM, or tie all LOW.
CTICHOUT[3:0]	Output	CTI channel output.	Either connect to **CTICHIN** of system CTI or CTM, or leave unconnected.
CTIIRQ[1:0]	Output	CTI interrupt, active HIGH.	Either connect to two of **IRQ[479:0]** inputs or an external interrupt controller, or leave unconnected.
TPIUACTV	Input	TPIU has data.	Can be connected to TPIU TPIUACTV output, or tied LOW if the TPIU is not included in the system.
TPIUBAUD	Input	Unsynchronized baud indicator from TPIU.	Can be connected to TPIU TPIUBAUD output, or tied LOW if the TPIU is not included in the system.

You might have a requirement to use CTI channels, for example when multiple processors are present in your SoC. In this case, you must use the ARM CoreSight SoC-400 product. This applies both to systems containing multiple Cortex-M33 processors, and to systems implementing processors of different types.

4.11 Power control interface

This section describes the power control interface.

4.11.1 Q-Channel interface

Descriptions of the Q-Channel interface signals for use in power management, and how you must set the signals in your design.

The Q-Channel input ***QREQn** signals are asynchronous to **CLKIN** and are synchronized inside the Cortex-M33 processor.

Table 4-26 Q-Channel interface signals

Signal name	Direction	Description	Connection information
COREQREQn	Input	Core quiescence request signal	Connect to your power management unit. **Note** For more information on the Q-Channel interface signals, see the *Low Power Interface Specification ARM® Q-Channel and P-Channel Interfaces*
COREQACCEPTn	Output	Core quiescence request accepted	
COREQDENY	Output	Core quiescence request denied	
COREQACTIVE	Output	Core active or activation request	
FPUQREQn	Input	FPU domain quiescence request signal	
FPUQACCEPTn	Output	FPU domain quiescence request accepted	
FPUQDENY	Output	FPU domain quiescence request denied	
FPUQACTIVE	Output	FPU logic active or activation request	
DBGQREQn	Input	Debug domain quiescence request signal	
DBGQACCEPTn	Output	Debug domain quiescence request accepted	
DBGQDENY	Output	Debug domain quiescence request denied	
DBGQACTIVE	Output	Debug logic active or activation request	
MTBQREQn	Input	MTB domain quiescence request signal	
MTBQACCEPTn	Output	MTB domain quiescence request accepted	
MTBQDENY	Output	MTB domain quiescence request denied	
MTBQACTIVE	Output	MTB logic active or activation request	
CORERET	Input	Core power domain in retention	
FPURET	Input	FPU power domain in retention	
DBGRET	Input	Debug power domain in retention	

4.11.2 Power control and sleep interface

Descriptions of the power control and sleep interface signals that are present on the processor, and how you must set the signals in your SoC design. This interface controls the low-power modes of the processor.

Table 4-27 Power control and sleep interface signals

Signal name	Direction	Description	Connection information
SLEEPING	Output	When HIGH indicates that the processor is ready to enter a low-power state. When LOW, indicates that the processor is running or wants to leave sleep mode. If **SLEEPHOLDACKn** is LOW, then the processor does not perform any fetches until **SLEEPHOLDREQn** is driven HIGH.	Connect to your power management unit
SLEEPDEEP	Output	Indicates that the processor and ETM are ready to enter a low-power state and the wake-up time is not critical. Only active when **SLEEPING** is HIGH.	
SLEEPHOLDACKn	Output	Acknowledge signal for **SLEEPHOLDREQn**. If this signal is LOW, irrespective of the **SLEEPING** signal value, the processor does not advance in execution and does not perform any memory operations.	Connect to your power management unit
SLEEPHOLDREQn	Input	Request to extend the processor sleeping state regardless of wake-up events. If the processor acknowledges this request driving **SLEEPHOLDACKn** LOW, this guarantees the processor remains idle even on receipt of a wake-up event.	
WICSENSE[482:0]	Output	Active HIGH signal. Indicates which input events can cause the WIC to generate the **WAKEUP** signal. The `WICLINES` configuration parameter determines the usable width of this signal. Therefore only the **WICSENSE**[`WICLINES-`1:0] bits are implemented and the remaining bits are driven LOW. The mapping to input events is: **WICSENSE[482:3]** **IRQ[479:0]**. **WICSENSE[2]** **EDBGRQ**. **WICSENSE[1]** **NMI**. **WICSENSE[0]** **RXEV**.	Connect to low-power control logic, or leave unconnected if the WIC is not present
WICENREQ	Input	Active HIGH request for deep sleep to be WIC-based deep sleep. This is driven from the power management unit.	Connect to low-power control logic, or tie LOW if the WIC is not present
WICENACK	Output	Active HIGH acknowledge signal for **WICENREQ**.	Connect to low-power control logic, or leave unconnected if the WIC is not present
WAKEUP	Output	Active HIGH signal to the power management unit that indicates a wake-up event has occurred and the processor system domain requires its clocks and power restored.	

4.12 ITM interface signals

Descriptions of the ITM interface signals that are present on the processor and how you must set the signals in your SoC design.

See the *AMBA® 4 ATB Protocol Specification* for more information.

Table 4-28 ITM interface signals

Signal name	Direction	Description	Connection information
AFREADYI	Output	CoreSight trace system ATB interface indicates that FIFO flush is finished.	If the `ITM` parameter is set to 1, connect to your CoreSight trace infrastructure, for example a *Trace Port Interface Unit* (TPIU). Tie all input signals LOW if this interface is not used.
AFVALIDI	Input	ATB interface FIFO flush request.	
ATDATAI[7:0]	Output	ATB interface data.	
ATIDI[6:0]	Output	ATB interface trace source ID.	
ATREADYI	Input	ATDATA can be accepted.	
ATVALIDI	Output	ATB interface data valid.	
DSYNC	Output	DWT synchronization request. Periodic formatter protocol synchronization request for the Cortex-M33 TPIU. If the TPIU is used, then this signal must be connected to its **SYNCREQ** input.	
SYNCREQI	Input	Trace synchronization request from instruction trace sink.	

4.13 ETM instruction trace interface

Descriptions of the ETM instruction trace interface signals that are present on the processor, and how you must set the signals in your SoC design.

See the *AMBA® 4 ATB Protocol Specification* for more information.

Table 4-29 ETM instruction trace interface signals

Signal name	Direction	Description	Connection information
AFREADYE	Output	CoreSight trace system ATB interface indicates that FIFO flush is finished.	If the ETM configuration parameter value is 1, connect to your CoreSight trace infrastructure. For example, a TPIU. If the ETM parameter value is 0, the ETM interface is not used and you must tie all input signals LOW and leave the outputs unconnected.
AFVALIDE	Input	ATB interface FIFO flush request.	
ATDATAE[7:0]	Output	ATB interface data.	
ATIDE[6:0]	Output	ATB interface trace source ID.	
ATREADYE	Input	ATDATA can be accepted.	
ATVALIDE	Output	ATB interface data valid.	
SYNCREQE	Input	Trace synchronization request from instruction trace sink.	
ETMTRIGOUT	Output	ETM event output bit 0. Can be connected to a TPIU trigger input.	

4.14 MTB interface

Descriptions of the MTB interface signals that are present on the processor, and how to connect the signals in your SoC design.

The MTB interface connects to SRAM and can be used for both trace and general-purpose storage by the processor.

Table 4-30 MTB interface signals

Signal name	Direction	Description	Connection information
MTBSRAMBASE[31:5]	Input	Location of MTB SRAM in processor memory map.	Connect to your trace or general-purpose storage as required.
HSELM	Input	Select access to MTB SRAM.	
HTRANSM[1:0]	Input	Transfer type.	
HBURSTM[2:0]	Input	Transfer burst length.	
HADDRM[31:0]	Input	Transfer address.	
HWRITEM	Input	Write transfer.	
HSIZEM[2:0]	Input	Transfer size.	
HNONSECM	Input	MTB AHB security level request. When asserted, **HNONSECM** indicates a Non-secure transfer.	
HWDATAM[31:0]	Input	Write data.	
HPROTM[6:0]	Input	Protection and outer memory attributes.	
HREADYM	Input	Ready for MTB.	
HREADYOUTM	Output	Ready out of MTB.	
HRDATAM[31:0]	Output	Read data.	
HRESPM	Output	Slave response.	
RAMCS	Output	RAM chip select.	
RAMAD[29:0]	Output	RAM address.	
RAMRD[31:0]	Input	RAM read data.	
RAMWD[31:0]	Output	RAM write data.	
RAMWE[3:0]	Output	RAM Write byte strobes.	

Note

- The value of the **MTBSRAMBASE** input is reflected in the MTB_BASE register. See *ARM® CoreSight™ MTB-M33 Technical Reference Manual*.
- **HSIZEM[2]** is ignored because the M-AHB and MTB SRAM data width is 32-bit.

4.15 External maskable and non-maskable interrupts

Descriptions of the interrupt interface signals that are present on the processor, and how you must set the signals in your SoC design.

Table 4-31 Interrupt interface

<table>
<tr><th>Signal name</th><th>Direction</th><th>Description</th><th>Connection information</th></tr>
<tr><td>IRQ[479:0]</td><td>Input</td><td>External interrupt signals. The <code>NUMIRQ</code> parameter configures the implemented bits of this signal.
——— Note ———
• IRQ and NMI signals are active HIGH and the hardware is agnostic between pulse- and level-signaled interrupts.
• You must ensure that the IRQ and NMI signals to the processor are synchronized to CLKIN using the appropriate circuit.
————————</td><td rowspan="2">Connect to interrupt logic. The number of functional interrupt signals depends on your implementation. Tie any bits that are not implemented LOW.</td></tr>
<tr><td>NMI</td><td>Input</td><td>Non-Maskable Interrupt.</td></tr>
<tr><td>CURRPRI[7:0]</td><td>Output</td><td>Current interrupt priority level</td><td rowspan="2">Might be connected to your interrupt logic or remain unconnected.</td></tr>
<tr><td>INTNUM[8:0]</td><td>Output</td><td>Interrupt number of the current execution context, from the Interrupt Program Status Register (IPSR).
——— Note ———
When the processor is in Thread mode, INTNUM is 0.
————————</td></tr>
</table>

Related concepts

2.2 Configuration options on page 2-27.

4.16 Miscellaneous signals

Descriptions of the miscellaneous signals that are present on the processor, and how you must connect these signals in your SoC design. The configuration input signals are sampled at reset.

Table 4-32 Miscellaneous interface signals

Signal name	Direction	Description	Connection information
ECOREVNUM[35:0]	Input	ECO revision number. The ECO revision field mappings are: **[35:32]** MTB. **[31:28]** ETM. **[27:24]** CTI. **[23:20]** ROM table. **[19:16]** ITM. **[15:12]** SCS. **[11:8]** DWT. **[7:4]** BPU. **[3:0]** CPUID revision.	Tie all bits LOW. This signal must be brought up to the top level on your SoC design to prevent synthesis tools optimizing out the logic this signal drives. ARM provides instructions on how to tie these signals in the event of an ECO change.
TRCENA	Output	Trace Enable. This signal reflects the setting of the DEMCR.TRCENA, indicating that the DWT and ITM units are enabled (when implemented).	Connect to clock gating and power gating logic for the TPIU.
TSVALUEB[63:0]	Input	Global timestamp value.	Connect to a natural binary count value if global timestamping is required. If not used, you must tie all bits LOW.
TSCLKCHANGE	Input	Timestamp clock ratio change.	Pulse this input if either **CLKIN** or the timestamp clock changes, even if the ratio does not change. Tie LOW if **TSVALUEB** is not used, or if **TSVALUEB** is generated from **CLKIN**.
LOCKSVTAIRCR	Input	Asserting this signal prevents changes to: • The Secure vector table base address. • Handling of Secure interrupt priority. • BusFault, HardFault, and NMI security target settings in the processor. When this signal is: **HIGH** Disables writes to the VTOR_S, AIRCR.PRIS, and AIRCR.BFHFNMINS registers. **LOW** Unlocks these registers.	This signal can be changed dynamically. If you want the registers unlocked, tie LOW, otherwise drive with external logic. **Caution** Tying these signal HIGH causes loss of interrupt control.
LOCKNSVTOR	Input	Asserting this signal prevents changes to the Non-secure vector table base address. When this signal is: **HIGH** Disables writes to the VTOR_NS register. **LOW** Unlocks this register.	These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic. **Caution** Tying these signal HIGH causes loss of interrupt control.

Table 4-32 Miscellaneous interface signals (continued)

<table>
<tr><th>Signal name</th><th>Direction</th><th>Description</th><th>Connection information</th></tr>
<tr><td>LOCKSMPU</td><td>Input</td><td>Asserting this signal prevents changes to programmed Secure MPU memory regions and all writes to the registers are ignored.
When this signal is:
HIGH Disables writes to the MPU_CTRL, MPU_RNR, MPU_RBAR, MPU_RLAR, MPU_RBAR_An and MPU_RLAR_An from software or from a debug agent connected to the processor in Secure state.
LOW Unlocks these registers.
This signal has no affect if the Cortex-M33 processor has not been configured with support for the ARMv8-M Security Extension, or if no Secure MPU regions have been configured.</td><td>These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic.
Caution
Tying these signal HIGH causes loss of memory protection control.</td></tr>
<tr><td>LOCKNSMPU</td><td>Input</td><td>Asserting this signal prevents changes to Non-secure MPU memory regions already programmed. All writes to the registers are ignored.
HIGH disables writes to the MPU_CTRL_NS, MPU_RNR_NS, MPU_RBAR_NS, MPU_RLAR_NS, MPU_RBAR_A_NSn and MPU_RLAR_A_NSn from software or from a debug agent connected to the processor.
LOW Unlocks these registers.
This signal has no affect if the Cortex-M33 processor has been configured without any Non-secure MPU regions.</td><td>These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic.
Caution
Tying these signal HIGH causes loss of memory protection control.</td></tr>
<tr><td>LOCKSAU</td><td>Input</td><td>Asserting this signal prevents changes to Secure SAU memory regions already programmed. All writes to the registers are ignored.
HIGH Disables writes to the SAU_CTRL, SAU_RNR, SAU_RBAR and SAU_RLAR registers from software or from a debug agent connected to the processor.
LOW Unlocks these registers.
This signal has no affect if the Cortex-M33 processor has not been configured with support for the ARMv8-M Security Extension, or if no SAU regions have been configured.</td><td>These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic.
Caution
Tying these signal HIGH causes loss of security attribution control.</td></tr>
</table>

4.17 FPU exception signals

Descriptions of the FPU exception signals that are present on the processor, and how you must set the signals in your SoC design.

The FPU signals indicate mathematical errors that cause floating-point exceptions. Using the FPU signals to indicate floating-point exceptions permits such exceptions to be diagnosed independently from software. For example, in safety-critical systems, exceptions can be routed directly to an on-chip safety controller.

Note

The FPU exception signals not related to the ARMv8-M exception handling model. This means you can connect the FPU exception signals to IRQ lines as your system design requires..

Table 4-33 FPU signals

Signal name	Direction	Description	Connection information
FPIXC	Output	Masked floating-point inexact exception	Cumulative exception flags from the *Floating Point Status and Control Register* (FPSCR). These signals indicate when a floating-point exception has occurred.
FPIDC	Output	Masked floating-point input denormal exception	
FPOFC	Output	Masked floating-point overflow exception	
FPUFC	Output	Masked floating-point underflow exception	
FPDZC	Output	Masked floating-point divide-by-zero exception	
FPIOC	Output	Invalid operation	

4.18 Events and Lockup

Descriptions the events and Lockup signals that are present on the processor, and how you must set the signals in your SoC design.

Table 4-34 Events and errors

Signal name	Direction	Description	Connection information
TXEV	Output	Event transmitted as a result of `SEV` instruction. This is a single-cycle pulse. You can use it to implement a more power efficient spin-lock in a multiprocessor system.	Connect to other processors in a multiprocessor system. In a multiprocessor system, **TXEV** from each processor can be broadcast to the **RXEV** input of the other processors. Leave unconnected if not required.
RXEV	Input	When HIGH this signal sets the Event register in the processor that is defined in the ARMv8-M architecture. This causes a WFE instruction to complete. It also wakes up the processor, if it is sleeping because it executed a WFE instruction.	You must construct the input to this signal as the logical-OR of all non-interrupt event generating sources of interest in your system. For example, the **TXEV** output of other ARM processors, or a single cycle completion signal from peripherals not already connected to any interrupt lines. You must add synchronization logic if this signal is driven from a different clock domain. Tie this input LOW if there are no non-interrupt event generating sources in your system.
LOCKUP	Output	When HIGH, indicates that the processor is in the architected Lockup state, because of an unrecoverable exception. See the *ARM®v8-M Architecture Reference Manual* for more information.	You can connect this signal to your own logic, for example a watchdog device, that can reset the processor using **nSYSRESET**. If your system executes instructions from a programmable memory, for example flash, after powerup, you must consider how that memory is programmed. The processor might enter Lockup state very quickly if the memory is uninitialized. If **nSYSRESET** is asserted immediately, there might not be enough time to connect a debugger to halt the processor and leave Lockup state. ARM recommends that your watchdog logic includes a software-programmable enable bit that gates the assertion of **nSYSRESET** because of **LOCKUP**. If you require entry into Lockup state to reset the system, your code must enable the functionality in your watchdog unit.

4.19 Implementation Defined Attribution Unit Interface

Descriptions of the external *Implementation Defined Attribution Unit* (IDAU) interface present on the processor, and how to connect the signals in your SoC design.

An IDAU can control the security attributes for most of the memory the Cortex-M33 processor addresses to a granularity of 32 bytes.

The following table shows the regions in the memory map:

- Where attributes are determined only by the security state of the processor.
- That cannot be controlled using the SAU or IDAU.

These regions are all associated with either *System Control Space* (SCS) or debug and trace components.

Table 4-35 Memory regions for SCS, debug, and trace components

Address range	Region or peripheral name
`0xE0000000-0xE0000FFF`	ITM
`0xE0001000-0xE0001FFF`	DWT
`0xE0002000-0xE0002FFF`	BPU
`0xE000E000-0xE000EFFF`	SCS
`0xE002E000-0xE002EFFF`	SCS Non-secure alias
`0xE0040000-0xE0040FFF`	TPIU[a]
`0xE0041000-0xE0041FFF`	ETM
`0xE0042000-0xE0042FFF`	CTI
`0xE0043000-0xE0043FFF`	MTB
MCUROMADDR-MCUROMADDR+`0xFFF`	MCU ROM table[a]
`0xE00FF000-0xE00FFFFF`	Processor ROM table

The security level returned by the Cortex-M33 MPU is a combination of:

- The region type defined in the internal SAU, if configured.
- The type returned on the associated IDAU interface.

If an address maps to regions defined by both internal and external attribution units, the region of the highest security level is selected.

At reset, before any SAU regions are programmed, the SAU_CTRL.ALLNS register bit selects the default internal security level. On reset the SAU_CTRL.ALLNS register is always reset to zero, setting all memory, apart from some specific regions in the PPB space, to Secure state. Setting SAU_CTRL.ALLNS bit to zero prevents an IDAU overriding any security level.

To allow an IDAU to specify the security level for all memory regions after reset, use secure software to disable all the internal SAU regions and set SAU_CTRL.ALLNS register bit to 1.

The interfaces are only functional when the processor has been configured with the ARMv8-M Security Extension using the Verilog SECEXT parameter and the input configuration signal CFGSECEXT is HIGH.

[a] The TPIU and MCU ROM table components are only included when the Teal MCU layer is used in the system. The base address of the ROM table is determined by the Verilog parameter MCUROMADDR.

The following table shows the signals for the two IDAU interfaces, A and B. The two interfaces are identical. All signal directions are relative to the Cortex-M33 processor.

The response from the IDAU on each interface must be identical for a given address.

Table 4-36 External SAU Interface

Signal Name	Direction	Description	Connection Information
IDAUADDRA[26:0]	Output	Address of the region. **IDAUADDRA[26:0]** is the 32-byte IDAU region associated with the access address. For example, for a 32-bit memory address A, **IDAUADDRA[26:0]** is **A[31:5]**.	-
IDAUADDRB[26:0]	Output	Address of the region. **IDAUADDRB[26:0]** is the 32-byte IDAU region associated with the access address. For example, for a 32-bit memory address B, **IDAUADDRB[26:0]** is **B[31:5]**.	-
IDAUNSA	Input	Non-secure region response. The **IDAUNSA** signal defines the attributes of the IDAU region.	Tie this input HIGH if an IDAU is not included.
IDAUNSB	Input	Non-secure region response. The **IDAUNSB** signal defines the attributes of the IDAU region.	Tie this input HIGH if an IDAU is not included.
IDAUNSCA	Input	Non-secure-callable region response. The **IDAUNSCA** signal defines the attributes of the IDAU region.	Tie this input LOW if an IDAU is not included.
IDAUNSCB	Input	Non-secure-callable region response. The **IDAUNSCB** signal defines the attributes of the IDAU region.	Tie this input LOW if an IDAU is not included.
IDAUIDA[7:0]	Input	Region number. **IDAUIDA[7:0]** is the 8-bit region identifier associated with the IDAU region. The value is written to the IREGION field of the result register value, Rd[31:24], the destination register of a `TT` instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.
IDAUIDB[7:0]	Input	Region number. **IDAUIDB[7:0]** is the 8-bit region identifier associated with the IDAU region. The value is written to the IREGION field of the result register value, Rd[31:24], the destination register of a TT instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.
IDAUIDVA	Input	Region number valid. **IDAUIDVA** indicates that the IDAU region number is valid. The value is written to the IRVALID field of the result register value, Rd[23], the destination register of a TT instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.
IDAUIDVB	Input	Region number valid. **IDAUIDVB** indicates that the IDAU region number is valid. The value is written to the IRVALID field of the result register value, Rd[23], the destination register of a TT instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.

Table 4-36 External SAU Interface (continued)

Signal Name	Direction	Description	Connection Information
IDAUNCHKA	Input	Region exempt from attribution check. When **IDAUNCHKA** is HIGH, the address associated with the IDAU region is not subject to attribution or security checks. The security attribution is determined only by the processor security state for software reads and writes to the address or by **HNONSECD** on D-AHB and DHCSR.S_SDE for debug accesses. This behavior is independent of any security attribution associated with the address in the processor SAU or presented on the IDAU interface. When **IDAUNCHKA** is LOW, then the security attribution is determined by the processor SAU or IDAU unless the address is specified as always exempt from security attribution checks.	Tie this input LOW if an IDAU is not included
IDAUNCHKB	Input	Region exempt from attribution check. When **IDAUNCHKB** is HIGH, the address associated with the IDAU region is not subject to attribution or security checks. The security attribution is determined only by the processor security state for software reads and writes to the address or by **HNONSECD** on D-AHB and DHCSR.S_SDE for debug accesses. This behavior is independent of any security attribution associated with the address in the processor SAU or presented on the IDAU interface. When **IDAUNCHKB** is LOW, then the security attribution is determined by the processor SAU or IDAU unless the address is specified as always exempt from security attribution checks.	Tie this input LOW if an IDAU is not included.

4.20 Test interfaces

For a list of test interface signals that are present on the processor, refer to the descriptions of the DFT signals.

Related references

Chapter 5 DFT Integration Guidelines on page 5-80.

4.21 CoreSight system integration

The Cortex-M33 processor optionally includes the CTI, MTB-M33 and ETM-M33 that are CoreSight compliant devices. It also contains an *Instrumentation Trace Macrocell* (ITM), *Debug and Watchpoint Trace* (DWT), and *Breakpoint Unit* (BPU) for use in a CoreSight compliant debug and trace system.

If you intend to integrate the processor into a CoreSight compliant debug system, refer to the sections on CoreSight ROM tables, and debugger connection and CoreSight discovery.

4.21.1 CoreSight ROM tables

The TEALMCU includes an internal CoreSight-compliant ROM table, `tealmcu/verilog/teal_mcu_apb_rom_table.v`, and debug resources, to enable you to build CoreSight-compliant debug systems.

The `teal_mcu_apb_rom_table.v` file is an example APB four-entry CoreSight ROM table that uses parameters to define its content. If you use this component as a system level ROM table, you must ensure the instantiation of the module uses your own JEP-106 manufacturer ID value and a part number that uniquely identifies your system.

ARM recommends that you build a CoreSight-compliant system to enable debug tools to identify the various system components correctly. See the *ARM® CoreSight™ Architecture Specification v2.0* for more information.

The Cortex-M33 processor ROM table provides pointers to the memory-mapped debug and trace resources inside the processor. See the *ARM® Cortex®-M33 Processor Technical Reference Manual* for more information about the processor ROM table and its functionality. This ROM table is fixed in the processor memory map at address `0xE00FF000` and is not modifiable.

If you are using the processor as part of a debug system, you must include one, or more, additional ROM tables within your system to enable a debugger to locate the debug and trace components inside the processor. ARM recommends that:
- You include a system level ROM table that includes your own JEP-106 ID and part number values, to enable debuggers to identify your system.
- An external ROM table includes an entry that points to the Cortex-M33 processor ROM at address `0xE00FF000`.

The example low-area debug and trace integration, `TEALMCU`, contains a system ROM table. See *Appendix F TEALMCU* on page Appx-F-228. The execution testbench includes a test you can use to verify the ROM table structure in the system. See *Chapter 10 Execution Testbench* on page 10-104 for more information.

Related references

Appendix F TEALMCU on page Appx-F-228.
Chapter 10 Execution Testbench on page 10-104.

4.21.2 Debugger connection and CoreSight discovery

The CoreSight Discovery mechanism allows a compliant debugger to locate and identify all CoreSight compliant debug components within your system by traversing the ROM tables and reading each component's ID registers. When you build your system, you must consider the conditions that require a debugger to connect to your system and how it might attempt the discovery process.

To be successful, the debugger must have sufficient time to connect to your system without being reset. If the debugger attempts the discovery process, accesses to the ROM tables and CoreSight components must also complete successfully.

You must consider how your EPPB bus infrastructure and CoreSight components are reset, and how your reset controller and any watchdog components interact. You might find it useful to use the **CPUWAIT** signal in conjunction with the various reset signals to allow you to reset your bus infrastructure without allowing the processor to begin executing code immediately. Until **CPUWAIT** is deasserted, the processor is effectively being held in reset while allowing debug slave port accesses on D-AHB.

Related references

4.5 Instruction execution control signals on page 4-42.

Chapter 5
DFT Integration Guidelines

This chapter describes the DFT integration guidelines for SoC integration, and the issues that relate to DFT that you must consider when you integrate the processor into your SoC design.

It contains the following sections:

5.1 About DFT integration

The processor supports scan and *Automatic Test Pattern Generation* (ATPG) techniques for production test vectors.

When you design the DFT strategy for your SoC, you must:

- Consider access to the processor signals for control and observation during test.
- Ensure that, during production test, you can control the processor for all required test modes.

Failure to do this might make the processor impossible to test. There are two cases for controlling the processor signals:

- The dynamic signals that must be controlled on a cycle-by-cycle basis must be either:
  - Brought directly to the top level of the chip for control by the tester.
  - Multiplexed with normal functional signals and made available during the manufacturing test modes using a test mode controller.

  This case also applies to all output signals that are compared during test modes.
- An SoC test controller can drive the control signals that are static during a test or they can be directly connected to a top-level SoC pin.

5.2 ATPG Test Interface

This section describes the ATPG test signals, scan test ports, and how to use them.

5.2.1 Scan test ports

The following table lists scan test ports.

Table 5-1 Scan test ports

Signal name	Direction	Description	Connection information
DFTCGEN	Input	Force all architectural clock gates open	These signals must be LOW during functional mode
DFTRSTDISABLE[1:0]	Input	Synchronized multi-layer logic resets disabled	

DFTCGEN

The clock gate enable signal, **DFTCGEN**, forces all the architectural clock gates on so that all internal clocks always run. An additional signal, **DFTSE**, can be created during implementation to control whether registers are in shift or capture mode.

ARM expects implementers to use this signal to enable the overrides to all clock gates and any clock gates that are added by synthesis.

Because the scan enable signal only controls the scan function that is in the implementation model, it must be disabled during formal equivalence checking between the gate level model and the RTL. Formal equivalence checking cannot check the scan chain shift path or the override on the clock gates that are added by synthesis.

DFTRSTDISABLE

The following figure shows the DFT reset structure. The reset logic contains multiple levels of reset synchronizers.

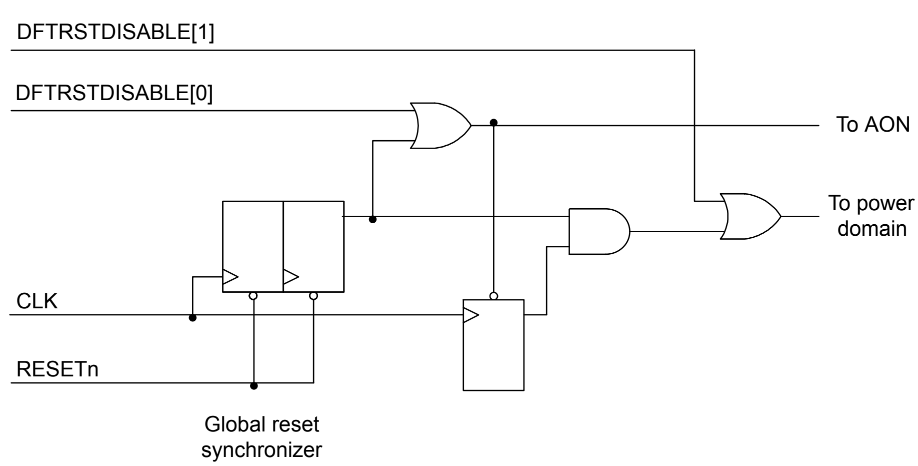


Figure 5-1 DFT reset structure

There are two reset disable signals that alternate for each level of reset synchronization. The reason for the two signals is to allow testing of both the asynchronous assertion of reset, and the synchronous deassertion of reset, during at-speed testing.

Some ATPG tools do not allow multicycle constraints specified separately on a rising or falling edge basis. To allow at-speed testing of reset deassertion, a single **DFTRSTDISABLE** signal can be deasserted at a time. While asserted, reset assertions are prevented from propagating through reset synchronizers and erroneously being tested for at-speed operation.

During scan shifting, both **DFTRSTDISABLE**[0] and **DFTRSTDISABLE**[1] must be asserted to allow safe shifting of patterns. **DFTRSTDISABLE**[0] and **DFTRSTDISABLE**[1] must never be deasserted at the same time during at-speed testing.

Chapter 6
Key Implementation Points

This chapter describes the key implementation points you must consider when you implement the processor.

It contains the following sections:

6.1 About key implementation points

This chapter lists the main points to consider when you implement the Cortex-M33 processor. Read this chapter in conjunction with the rest of the information in this guide and your Cortex-M33 processor reference methodology documentation.

You can use this chapter to check that you have covered the implementation steps described in the other chapters.

6.2 Key implementation tasks

The following table lists the key tasks for implementation.

Table 6-1 Key implementation tasks

Key task	Description
1. Select level of hierarchy to implement. This can be either: • The processor top level, `TEAL`. • A higher level in your SoC that includes the Cortex-M33 processor.	See *2.1 About configuration guidelines* on page 2-26 and *2.2 Configuration options* on page 2-27.
2. Configure the processor parameters.	See *2.2 Configuration options* on page 2-27.
3. Select appropriate library cells for clock gating and *Clock-Domain Crossing* (CDC) purposes.	See *6.3 Other considerations for implementation* on page 6-87.
4. Determine optimum floorplan.	See *7.5 Considerations for floorplans* on page 7-94.
5. Perform synthesis and scan insertion.	See the reference methodology documents from your EDA tool vendor for information on equivalence checking tools.
6. Create layout.	
7. Perform *Layout Versus Schematic* (LVS) checks and *Design Rule Checks* (DRC).	
8. Perform timing verification.	
9. Perform characterization.	
10. Run ATPG.	See *Chapter 5 DFT Integration Guidelines* on page 5-80 and the reference methodology documents from your EDA tool vendor.
11. Perform netlist dynamic verification.	See *Chapter 8 Netlist Dynamic Verification* on page 8-96.
12. Perform functional verification using logical equivalence checking tools. Optionally, you can also replay test vectors.	See the reference methodology documents from your EDA tool vendor.
13. Perform sign-off in accordance with the agreed criteria and your sign-off obligations.	
14. Sign off your implementation.	See *Chapter 9 Sign-off* on page 9-98.

Note

You must complete the implementation process to produce complete and verified deliverables.

Related references

Implementation obligations on page 10.

6.3 Other considerations for implementation

There are points that you must consider when you implement design options that are not covered by the configuration options.

6.3.1 Special purpose cells

More components are provided in the deliverables that you might optionally use in your system. Some of these components have clock domain crossing paths and clock gates that require similar special purpose modules to those used in the processor. You are only required to implement technology-specific versions of these modules if you are using the optional components.

About these modules

- You can use these files for RTL simulation.
- You must copy the files that you require for your implementation into a new directory for the technology that you are using.
- You must use the original RTL files as part of the validated processor RTL during logical equivalence checking as part of the sign-off procedure.
- You must not use these files for synthesis.
- You must implement an equivalent module that instantiates cells with the required properties from your cell library for synthesis. You must also ensure that the cells you instantiate are maintained throughout the implementation flow and are not resynthesized to alternative cells.
- You must implement the modules using cells from your technology library that have the characteristics that are described in the comments in the technology independent versions.
- Example modules containing cell instantiations that are required for implementation are provided with the Reference Methodology. See the Reference Methodology release note for the location of these files.
- To make it easier for LEC tools to check equivalence between the technology independent and technology-specific versions of the special purpose cells, ARM recommends that you use the same instance name with the `_reg` suffix for your flip-flops as the Verilog reg nets that infer flip-flops in the generic modules.

The following table shows all the cells in the Cortex-M33 processor, Cortex-M33 ETM, and example system components.

Table 6-2 Cortex-M33 processor and example system components

Component	Cells
DAP	`tealcell_sync.v`
	`tealcell_sync_preset.v`
	`teal_cdc_comb_and2.v`
	`teal_cdc_comb_mux2.v`
	`teal_cdc_comb_or2.v`
	`teal_cdc_connect.v`
	`teal_cdc_send.v`
	`teal_cdc_send_reset.v`
TPIU	`tealcell_sync.v`
	`teal_cdc_comb_and2.v`
	`teal_cdc_comb_mux4.v`
	`teal_cdc_connect.v`

Table 6-2 Cortex-M33 processor and example system components (continued)

Component	Cells

Cortex-M33 processor	`tealcell_sync.v`
	`tealcell_and_gate.v`
	`tealcell_arch_clkgate.v`
	`tealcell_inter_clkgate.v`
	`tealcell_cdc_mux2.v`

ETM	`tealcell_inter_clkgate.v`
execution_tb_pmu	`teal_pmu_sync_reset.v` **Note** This cell is located in `logical/testbench/execution/verilog/models`

6.3.2 Architectural clock gating

The Cortex-M33 processor contains some clock gating cells to reduce the dynamic power dissipation by gating the clock to groups of registers within the design. These clock gates are called architectural clock gates to distinguish them from the clock gates that might be inferred by the synthesis tools during the implementation process. The architectural clock gating cells must be provided as modules named `tealcell_arch_clkgate`.

Architectural clock gating is optional because correct operation of the processor is not dependent on it. If architectural clock gating is not required, then you must provide an implementation of the `tealcell_arch_clkgate` module that directly connects the clock input and output ports together. The other ports are unused in this case.

If architectural clock gating is required, the `tealcell_arch_clkgate` module must directly instantiate a positive-edge clock gating cell from your target library. Correctly connect the clock input and outputs, clock enable, and scan-enable signals.

For reference, simulation and logical-equivalence checking purposes, `logical/models/cells/generic/tealcell_arch_clkgate.v` provides a configurable version of the `teal_clk_gate` module. Do not use this version for synthesis.

Chapter 7
Floorplan Guidelines

This chapter describes the floorplan that is used as a starting point for your design.

It contains the following sections:

7.1 About floorplanning

The processor is a tightly tuned design with a high density of paths in the critical range of the design. A good floorplan of the processor is crucial to ensure that the performance of the macrocell is not degraded.

The following figure is a process diagram that shows the top-level inputs, resources, outputs, and controls and constraints for floorplanning.

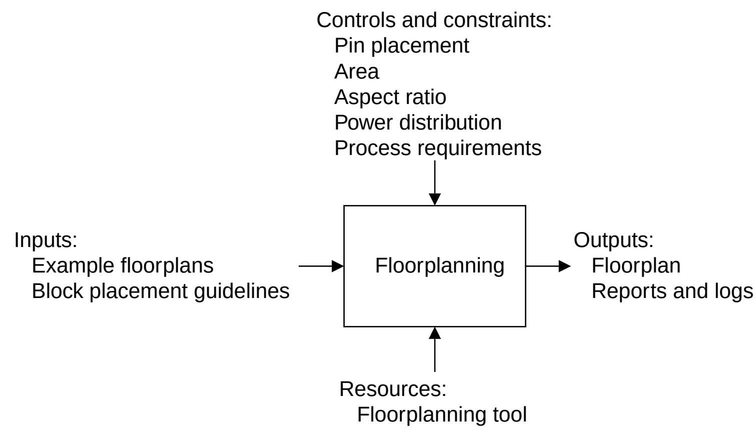


Figure 7-1 Floorplan process

For more information on floorplanning, see the documentation for your chosen Cortex-M33 Reference Implementation Flow.

7.2 Resource requirements for floorplans

This document assumes that you have suitable EDA tools and compute resources for floorplanning.

7.3 Controls and constraints for floorplans

There are certain controls and constraints that can influence floorplanning.

These include:

- Pin placement.
- Area.
- Aspect ratio.
- Power distribution.
- Process and library requirements.

7.4 Inputs for floorplans

Inputs are specific to your floorplanning tool, and they can include the following:

- Example floorplans.
- Block placement guideline.
- Pin placement.
- Power distribution.
- Placement blockages.

7.5 Considerations for floorplans

It is not expected that you have to perform hierarchical floorplanning of the processor.

Related references

Chapter 4 Functional Integration Guidelines on page 4-34.

7.6 Output from floorplans

Output files are specific to your floorplanning tool.

They might include:

- Logs.
- Floorplan with power grid and pin locations.
- Placement and route guides.

Chapter 8
Netlist Dynamic Verification

This chapter describes how to test the functionality of your implementation of the processor.

It contains the following section:

8.1 Netlist dynamic verification

You must use static equivalence checking tools to verify your post-synthesis and post-layout netlists. This is described in the Reference Methodology documentation from ARM.

Note

ARM requires you to use equivalence tools to verify your netlist. Equivalence checking provides a complete method for verifying your netlist and does not require the extensive simulation compute resources that other methods require.

In addition, you can use the execution testbench to perform dynamic verification, by simulating your netlist. See *Chapter 10 Execution Testbench* on page 10-104 for more information.

Chapter 9
Sign-off

This chapter describes the sign-off criteria.

In addition to your normal ASIC flow sign-off checks, you must satisfy certain verification criteria before you sign off your design.

It contains the following sections:

9.1 About sign-off

The following figure is a process diagram that shows the top-level inputs, resources, outputs, and controls and constraints for sign-off.

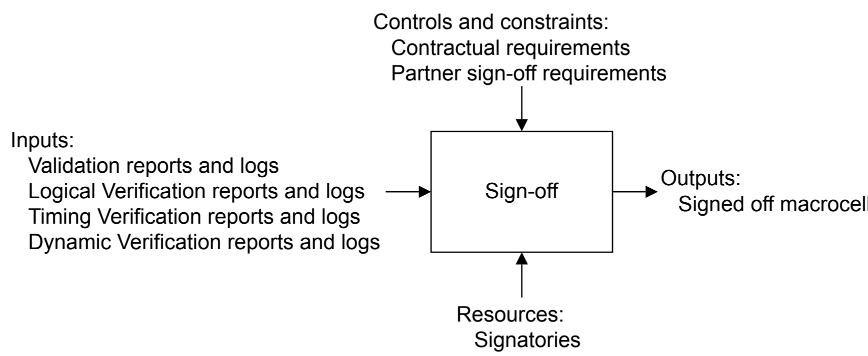


Figure 9-1 Sign-off process

9.2 Obligations for sign-off

Signatories must approve the sign-off of the design.

This must be in accordance with:

- The terms of the contract with ARM.
- Any other partner sign-off requirements.

9.3 Criteria for sign-off

The following sections describe the two types of requirement for sign-off.

9.3.1 Mandatory for sign-off

All ARM partners must fulfill the terms of their contract with ARM to complete sign-off.

Usually, you must complete the following implementation stages successfully for sign-off:

- Logical Equivalence Check. See the Reference Methodology documents that are supplied by your EDA tool vendor.
- Reports and logs from each of these stages are required for sign-off.
- A certain minimum set of deliverable outputs is required at the end of the implementation.

Note

You can change the timing constraints to suit your design provided it still meets all the mandatory criteria for sign-off.

9.3.2 Recommended for sign-off

The following table shows the recommended stages for sign-off.

Table 9-1 Recommended stages for sign-off

Sign-off stage	Notes
Design Rule Checking (DRC)	See the Reference Methodology documents that are supplied by ARM.
Layout versus Schematic (LVS)	
Characterization	
Timing verification through *Static Timing Analysis* (STA)	
Gate-level simulation in the execution testbench with a back-annotated netlist running at your target clock frequency	-

Related concepts

9.5 Completion of sign-off on page 9-103.

9.4 Steps for sign-off

To sign off the processor you must meet the criteria as they are described in each of the following stages in the design flow:

1. *9.4.1 RTL verification* on page 9-102.
2. *9.4.2 Post-synthesis and place-and-route* on page 9-102.
3. *9.4.3 Post place-and-route timing* on page 9-102.

Note

You must also ensure that you meet any additional verification or usage criteria that might be identified in the legal agreement between your company and ARM.

9.4.1 RTL verification

You must verify the RTL deliverables before you begin the synthesis stage by running the supplied execution testbench on the configured RTL. Running these tests demonstrates that the RTL has been successfully installed and configured.

9.4.2 Post-synthesis and place-and-route

You must verify the functionality of the final place-and-routed netlist before you sign off the macrocell. This verification requires you to prove logical equivalence between the validated processor RTL and the final place-and-routed netlist, using formal verification tools. See the Reference Methodology documents that are supplied by your EDA tool vendor.

9.4.3 Post place-and-route timing

You must verify the timing of the post-layout netlist before you sign off the netlist using STA. ARM also recommends that you run:

- All the functional vectors appropriate to your configured build.
- Back-annotated timing, as a final check.

9.5 Completion of sign-off

For successful completion of sign-off, you must have completed and verified ARM-related deliverables from the implementation process.

These include:

- GDS II output.
- Test vectors.
- Extracted timing model.
- Simulation model.
- All required reports and logs.

Chapter 10
Execution Testbench

This chapter describes the execution testbench for the Cortex-M33 processor.

It contains the following sections:

10.1 About the execution testbench

The execution testbench, execution_tb, is provided as a reference to enable integration of the Cortex-M33 processor into your system. It contains tests to check that you have completed the integration process correctly.

The execution testbench:

- Instantiates the supplied `TEALMCU` module. The `TEALMCU` is an example integration of the processor with low area debug and trace components. See *Appendix F TEALMCU* on page Appx-F-228.
- Supports configuration options. See *10.5 Configuring the execution testbench RTL* on page 10-112 for more information.

You can modify the execution testbench to suit your requirements, see *10.11 Modifying the execution testbench RTL for your SoC* on page 10-128, *10.15 Modifying the execution testbench tests* on page 10-143, and *10.1.1 Execution testbench limitations* on page 10-106 for more information.

The execution testbench tests work with all valid configuration options of the Cortex-M33 processor, DAP, TPIU, optional ETM-M33, and optional MTB-M33.

The execution testbench supports RTL, DSM, and netlist simulation.

Note

The execution testbench also supports *Unified Power Format* (UPF) power aware simulation. See *10.8.3 Running with UPF* on page 10-123 and *Chapter 12 Power Intent* on page 12-154 for more information.

The tests that are supplied with the execution testbench are written in C and are compliant with the *Cortex Microcontroller Software Interface Standard* (CMSIS) to aid code portability. You can modify the test code to work in your own system.

The execution testbench is based on a simple example microcontroller, `execution_tb_mcu`, that instantiates the execution testbench subsystem, `execution_tb_sys`, that in turn instantiates the `TEALMCU` level, ROM, RAM, a *Power Management Unit* (PMU), a system level ROM table, a reset controller, and some *General Purpose Input Output* (GPIO). If the configuration includes debug, a second instantiation of the processor subsystem, the Debug Driver generates debug stimulus in the testbench.

The following figure shows a high-level view of the execution testbench, where the example microcontroller `execution_tb_mcu` is the device under test.

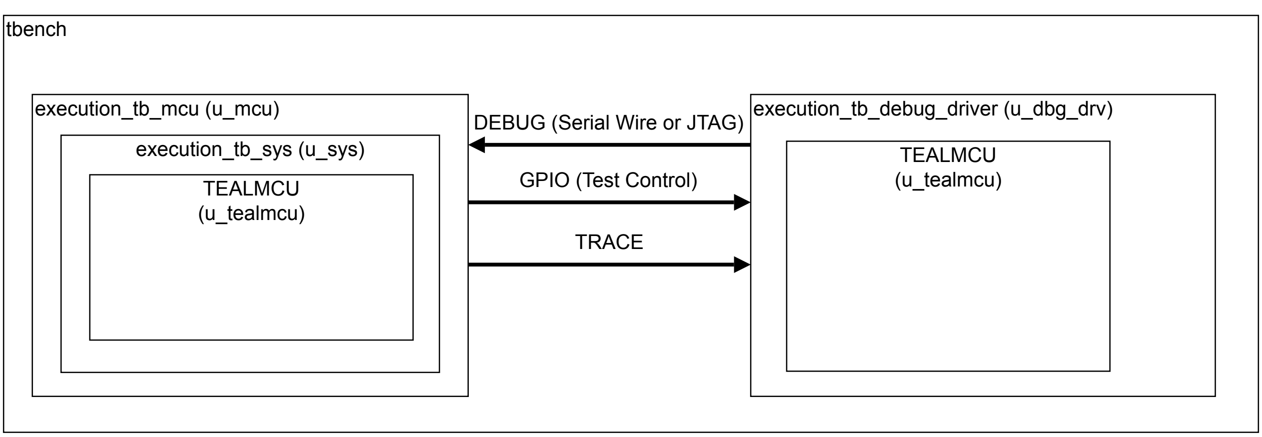


Figure 10-1 Cortex-M33 processor execution testbench overview

10.1.1 Execution testbench limitations

The execution testbench is a simple, representative example of a basic microcontroller. However, the execution testbench has some known limitations and is not a replacement for other testing.

The execution testbench tests do not exhaustively test the Cortex-M33 processor I/O because not all I/O pins have an effect that is visible to program code, and some pins are tied to static values within the `execution_tb_mcu` example system. This means that you must not use the passing of all execution testbench tests as the only sign-off criteria for successful processor integration.

Execution testbench limitations include the following:
- Some implementation parameters do not have a program-visible effect, so cannot be tested in the execution testbench. For example, `RAR`.
- Some tests are timing sensitive to test the effect of implementation parameters such as `WIC` and `WICLINES`.
- Some tests require the presence of one external interrupt. The default external interrupt default for the execution testbench is **IRQ0**.
- Signal **CPUWAIT** that can halt the processor after leaving reset, is not used.
- The execution testbench is designed to test a single processor subsystem such as a microcontroller. The execution testbench does not support testing at the `TEAL` level of hierarchy.
- The DFT signals **DFTCGEN** and **DFTRSTDISABLE[1:0]** are tied off.
- The Debug access control signals **DBGEN**,**NIDEN**, **SPIDEN**, and **SPNIDEN** are tied off.

See *Sign-off* for details of your sign-off obligations.

Related references

Chapter 9 Sign-off on page 9-98.
5.2 ATPG Test Interface on page 5-82.

10.2 Execution testbench flow

The execution testbench (`execution_tb`) is intended as a platform that enables you to develop a SoC incorporating the Cortex-M33 processor.

The following figure shows the execution testbench flow.

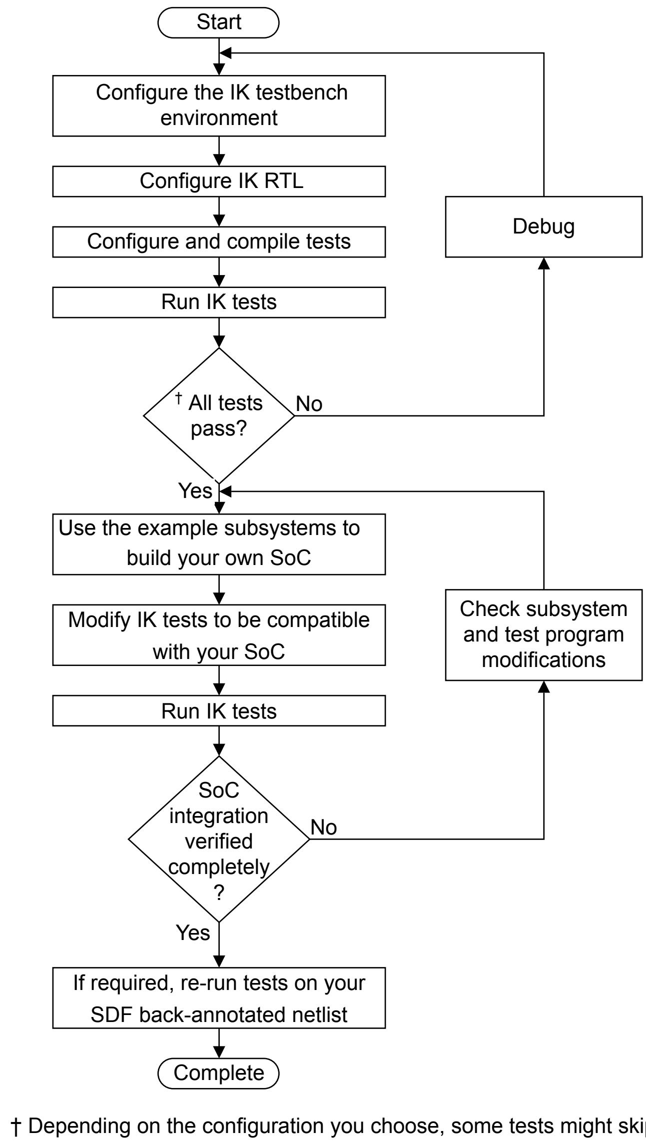


Figure 10-2 Execution testbench flow

10.3 Test overview

The following table shows the test programs in the `testbench/execution_tb/tests` directory.

Table 10-1 Test programs

Test program	Description
`hello_world`	The processor reads the CPUID register and writes to the GPIO registers to print a simple message. This must be the first test run. You can run this test without compiling it, as both the source code and binary executable versions are supplied. If you want to use the pre-compiled binary executable, you must copy it from the `pre_compiled` directory to the `test` directory before use.
`config_check`	This test verifies that the processor configuration matches the expected configuration values set in the `EXECTB_Config.h` file. This must be the second test run.
`coprocessor`	This test demonstrates the operation of the example coprocessor.
`debug`	The test checks the pins **LOCKUP**, **EDBGRQ**, **HALTED**, **DBGRESTART**, and **DBGRESTARTED**.
`dhrystone`	This test runs the Dhrystone benchmarking program. The default number of iterations is five. You can change the number of iterations by editing the `ITERATIONS` value in the `Makefile`. You can run this test without compiling it, as both the source code and binary executable versions are supplied. To use the pre-compiled binary executable, copy it from the `pre_compiled` directory to the `test` directory before use. **Note** The pre-compiled binary of the `dhrystone` test that is supplied is only for processor configurations that include the FPU. If your processor configuration does not include the FPU, you must configure the `Makefile` as described in *10.6.2 Test program compilation using the ARM or GCC compilers* on page 10-118 and compile the `dhrystone` test yourself.
`eppb`	This test demonstrates access to the small memory on the EPPB bus.
`etm_trace`	If the ETM is licensed and implemented, this test generates instruction trace using the ITM, ETM, and the TPIU. Checking is limited to validating the protocol of the generated trace.
`exclusive`	This test demonstrates the local and global exclusive monitors.
`idau`	This test demonstrates the operation of the IDAU.
`interrupt`	This test exercises **NMI**, **IRQ**, **TXEV**, and **RXEV**. The connection of up to 64 interrupts are tested.
`itm_trace`	This test generates trace using the ITM and the TPIU. Checking is limited to validating the protocol of the generated trace.
`maxpwr_cpu`	This tests the power consumption of the processor at sustained maximum power running integer operations. You can run this test without compiling it, as both the source code and binary executable versions are supplied. If you want to use the pre-compiled binary executable, you must copy it from the `pre_compiled` directory to the `test` directory before use. The pre-compiled code measures power in the default configuration with the MPU, SAU and DWT enabled. If any other configuration of the MPU, SAU, DWT, ETM and MTB is present in your design, you must reconfigure the `execution_tb` and recompile the test.
`mtb_trace`	If the MTB is licensed and implemented, this test demonstrates the operation of the MTB trace and accessing the external trace memory.
`non_secure`	This test demonstrates switching between Secure and Non-secure states. **Note** The code takes shortcut to demonstrate security switching and must not be used as an example of how to develop mixed Secure and Non-secure code.
`reset`	This test checks the **SYSRESETREQ** output and that accesses to the AHB default slave cause a fault.

Table 10-1 Test programs (continued)

<table>
<tr><th>Test program</th><th>Description</th></tr>
<tr><td><code>romtable</code></td><td>This test checks that it is possible for a debugger to autodetect the Cortex-M33 processor. The test uses the DAP to locate the architecturally defined ROM table to locate the processor. If you have included system-level ROM tables, the content of these is displayed, but not checked.
Note
You must set the <code>JEPID</code> and <code>PARTNUM</code> fields correctly in your system ROM table, otherwise this test fails. Modify the ROM table ID default parameter values at the <code>TEALMCU</code> level appropriately and set the execution testbench expected values in the <code>EXECTB_Config.h</code> file to match.
See F.2 Configuring the TEALMCU level on page Appx-F-230, and the execution testbench system ROM table configuration options in Table 10-3 Execution testbench defines on page 10-112. For more information on ROM table SoC specific IDs, see 4.21.1 CoreSight ROM tables on page 4-78.</td></tr>
<tr><td><code>saxpy_scalar</code></td><td>This test is used to measure maximum power with the floating-point unit. You can run this test without compiling it, as both the source code and binary executable versions are supplied. To use the pre-compiled binary executable, copy it from the <code>pre_compiled</code> directory to the test directory before use. The pre-compiled code measures power in the default configuration with the MPU, SAU and DWT enabled. If any other configuration of the MPU, SAU, DWT, ETM and MTB is present in your design, you must reconfigure the <code>execution_tb</code> and recompile the test.</td></tr>
<tr><td><code>sleep</code></td><td>This test exercises the sleep modes of the processor, and the SLEEPING and SLEEPDEEP signals.
The test uses an interrupt to wake the processor, and if the processor includes debug, the test also wakes the processor from sleep through the DAP. You can extend this test to verify state retention if that is implemented.
Note
• If the UPF file does not include state retention, the sleep test is not supported for power aware simulation.
• If state retention flip-flops are not used, the sleep test is not supported for power gated netlist simulation.</td></tr>
<tr><td><code>wfi</code></td><td>This test measures minimum power when the processor is awaiting an interrupt. You can run this test without compiling it, as both the source code and binary executable versions are supplied. To use the pre-compiled binary executable, copy it from the <code>pre_compiled</code> directory to the <code>test</code> directory before use.</td></tr>
</table>

10.4 Configuring the testbench

The testbench instantiates the `execution_tb_mcu` level. The testbench supports simulation of Verilog RTL source.

The following table shows the Verilog command files in the `logical/testbench/execution_tb/verilog/` directory.

Table 10-2 Verilog command files

File	Description
`execution_tb.vc`	Used for all simulations. Defines paths to testbench components.
`dsm.vc`	Used for DSM simulations. Defines the paths to the DSM testbench Verilog files.
`rtl.vc`	Used for RTL simulations. Defines the paths to the processor Verilog files. This file assumes that the CoreSight ETM and MTB is present in your Cortex-M33 processor configuration. If your configuration does not include the ETM or MTB, then you can modify the `rtl.vc` file to comment out the lines relating to the ETM or MTB.
`netlist.vc`	Used for netlist simulation. Defines the paths of the processor netlist and standard cell libraries. +defines to select unit delay or SDF simulation. It also defines controls for VCD generation, and controls power gate netlist simulation.

You can edit the `logical/testbench/execution_tb/make.cfg` file to configure the execution testbench. The configuration options allow you to enable or disable the following:

- 64-bit simulation.
- Assertions.
- DSM simulation.
- Netlist simulation.
- Netlist simulation with SDF.
- Tarmac trace.
- UPF simulation.
- Using the simulation GUI.
- Visualizer capture, MTI simulator only.
- Verdi capture, MTI and VCS simulators only.

Note

- If you change any parameters or anything that affects the processor or testbench, you must run `make clean` before running a test.
- If you are using a 32-bit machine to run the execution testbench, you must not enable 64-bit simulation.

10.4.1 Netlist considerations

Before simulating a netlist, you must modify the `netlist.vc` file.

The following Verilog defines affect netlist simulations:

- `ARM_UD_MODEL` selects the unit delay timing model in the gate library.
- `ARM_TEAL_EXECTB_SDF`, in the file `netlist.vc`, determines if delays specified by SDF are included in netlist simulations. Ensure this define is included if you want SDF delay annotation. If not, ensure this define is not included.
- `ARM_PG_ON` makes connections from the PMU power regulator control signals through to the processor
- `ARM_FPU_PG_ON`, `ARM_DEBUG_SPRG_ON`, `ARM_MTB_SPRG_ON` control connection of the power regulator signals for the FPU, DEBUG and MTB power domains
- `ARM_RET_ON` controls the connection of the retention control signals to the processor.
- `PMU_PWR_DOWN` allows the PMU to power down the various power domains.
- `ARM_TEAL_EXECTB_NETLIST_SDF`, in `tbench.v`, points to the SDF file from synthesis flow.
- `ARM_TEAL_EXECTB_NETLIST_SCOPE`, in `tbench.v`, points to the netlist instance.
- `ARM_INPUT_DELAY` in file `TEAL_input_delay.v` sets the input delay in the wrapper. An input delay of 1ns is applied to the input ports of the netlist. The Verilog wrapper, `TEAL_input_delay.v` is located in the `testbench/execution_tb/verilog` directory.

Before running a netlist simulation, you must ensure that you have either:

- Synthesized using the *Reset All Registers* (RAR) option.
- Alternatively, either ensure that all registers are initialized by depositing a value on flip-flop outputs, or modify your gate-level library so that flip-flops are initialized at the start of the simulation.

Related concepts

10.4 Configuring the testbench on page 10-110.

10.4.2 State retention power gating considerations

To run a UPF RTL simulation, you must define `ARM_PG_ON` in `logical/testbench/execution_tb/verilog/rtl.vc`.

You must set:

- `ARM_FPU_PG_ON`, `ARM_DEBUG_RG_ON`, `ARM_MTB_RG_ON` to control the connection of the power regulator signals for the FPU, DEBUG and MTB power domains individually.
- `ARM_RET_ON` to control the connection of the retention control signals to the processor.

Note

If you do not use UPF in RTL simulations, `ARM_PG_ON` must not be defined.

Related concepts

10.11.1 Testbench structure on page 10-128.

10.5 Configuring the execution testbench RTL

This section describes how you can configure the execution testbench, `execution_tb`, RTL.

10.5.1 Configuring the execution testbench

The system ROM table SoC-specific ID values must be configured correctly.

The execution testbench uses Verilog defines to control the generation of logic outside of the `TEALMCU` level and to capture system-specific values for parameters.

Note

Before running any tests, you must:

- Configure the system ROM table SoC-specific ID values correctly for your `JEPID`, `PARTNUM`, and revision in the `TEALMCU.v` file. See *F.2 Configuring the TEALMCU level* on page Appx-F-230 and *4.21 CoreSight system integration* on page 4-78.
- Set these values in the execution testbench ROM table configuration test defines, see *Table 10-11 Execution testbench system ROM table configuration options* on page 10-118.

The following table lists the defines in `logical/testbench/execution_tb/verilog/execution_tb_defs.v`. You must modify the values to match your configuration of the `TEALMCU` level.

Table 10-3 Execution testbench defines

Define	Description
`TEAL_EXECTB_CFGBIGEND`	Specifies the endianness of the Cortex-M33 processor in `execution_tb_mcu` and in the debug driver. **Note** If you are compiling tests using an ARM compiler, edit the `COMPILE_BE` variable in `execution_tb/tests/make.cfg` to match.
`TEAL_EXECTB_CFGSSTCALIB`	Specifies the Secure SysTick calibration.
`TEAL_EXECTB_CFGNSSTCALIB`	Specifies the Non-secure SysTick calibration.
`TEAL_EXECTB_CFGFPU`	If configured, enables support for hardware floating-point.
`TEAL_EXECTB_CFGDSP`	If configured, enables support for ARMv8-M DSP extensions.
`TEAL_EXECTB_INITSVTOR`	Specifies the Secure vector table initialization value.
`TEAL_EXECTB_INITNSVTOR`	Specifies the Non-secure vector table initialization value.
`TEAL_EXECTB_CFGSECEXT`	Enable support for ARMv8-M Security Extension, if configured.
`TEAL_EXECTB_MPUNSDISABLE`	Disable support for the Non-secure MPU, if configured.
`TEAL_EXECTB_MPUSDISABLE`	Disable support for the Secure MPU, if configured.
`TEAL_EXECTB_SAUDISABLE`	Disable support for the SAU, if configured.
`TEAL_EXECTB_INSTANCEID`	Specifies the Instance ID for the DAP.
`TEAL_EXECTB_MTB_BASE_ADR`	Specifies the MTB RAM memory base address.
`TEAL_EXECTB_MTBAWIDTH`	Specifies the MTB RAM memory address width between bits[32:5].
`TEAL_EXECTB_MCUROMADDR`	Specifies the location of the ROM table base address for the Cortex-M33 DAP. Changing this define does not modify the location of the system level ROM table in the system. You must only modify this define if you are also modifying the `execution_tb_mcu` memory map.
`TEAL_EXECTB_DPSEL`	Specifies the Debug Port Select.

Table 10-3 Execution testbench defines (continued)

Define	Description
`TEAL_EXECTB_TARGETID`	Specifies the target ID.
`TEAL_EXECTB_JEPID`	Specifies the JEP106 identification code.
`TEAL_EXECTB_JEPCONT`	Specifies the JEP106 continuation code.
`TEAL_EXECTB_PARTNUM`	Specifies the part number for the MCU.
`TEAL_EXECTB_CORERET`	Specifies that retention is supported in Core power domain.
`TEAL_EXECTB_FPURET`	Specifies that retention is supported in FPU power domain.
`TEAL_EXECTB_DBGRET`	Specifies that retention is supported in Debug power domain.
`TEAL_EXECTB_PDDEBUG`	Specifies the debug power domain is present.
`TEAL_EXECTB_PDFPU`	Specifies the FPU power domain is present.
`TEAL_EXECTB_PDMTB`	Specifies the MTB power domain present.
`TEAL_EXECTB_RET_AON`	Configures the PMU to maintain the retention power supplies always on.
`PMU_PWR_DOWN`	Configures the PMU to remove power from the processor when in sleeping.

Note

The execution testbench requires these defines to be set appropriately so that the GPIO, memories and the Cortex-M33 DAP are correctly configured to support the `TEALMCU` level. The instantiation of the `TEALMCU` level in `logical/testbench/execution_tb/verilog/example_sys/verilog/execution_tb_sys.v` uses these defines. They are also used to configure other execution testbench logic.

If you build a system that includes debug, and you include a CoreSight system level ROM table, you must pass your own JEP-106 ID, part number, and revision values into the instantiation of `teal_mcu_apb_rom_table.v`. See *4.21 CoreSight system integration* on page 4-78, execution testbench system ROM table configuration options in *Table 10-10 Execution testbench CoreSight ETM-M33 configuration options* on page 10-117.

10.6 Configuring and compiling tests

This section describes how to configure and compile the test programs before running a testbench simulation.

The tests have been developed and tested using ARM Development Studio 6 (DS-6), and GCC compilers running under Linux.

Note

See the *ARM® Cortex®-M33 MCU Release Note* for the ARM compiler versions that have been tested with the deliverables.

You might have to modify:
- The tests if you want to use an alternative compiler.
- The `Makefile`, or use an alternative system, if you want to use a different OS.

10.6.1 Execution testbench test configuration

The execution testbench tests check the configuration of the Verilog RTL on which they are running, against the expected values that are defined in the `EXECTB_Config.h` header file.

`EXECTB_Config.h` contains several `EXPECTED_*` defines having a variable part that is the same as the RTL configuration option. You must ensure that the configuration options match the configuration of your RTL, see *2.2 Configuration options* on page 2-27 for more information.

Note

If you use Keil® MDK-ARM to build your tests, you can use the Configuration Wizard to update the values in `EXECTB_Config.h`.

Execution testbench processor configuration options

The execution testbench processor configuration options are given in this section.

Table 10-4 Execution testbench processor configuration options

Name	Description
`EXPECTED_FPU`	Expected value of `FPU` parameter. See *Table 2-1 Cortex-M33 processor configuration options summary* on page 2-27. **Note** If you are compiling tests using an ARM or GCC compiler, edit the `COMPILE_FPU` variable in the `tests/make.cfg` in the tests directory to match.
`EXPECTED_DSP`	Expected DSP Extension (0,1). **Note** If you are compiling tests using an ARM or GCC compiler, edit the `COMPILE_DSP` variable in the `tests/make.cfg` in the tests directory to match.
`EXPECTED_SECEXT`	Expected Security Extension (0,1).
`EXPECTED_CPIF`	Expected External Coprocessor interface unit (0,1).
`EXPECTED_MPU_NS`	Expected value of Non-secure `MPU` regions parameter (0, 4, 8, 12, 16).
`EXPECTED_MPU_S`	Expected value of Secure `MPU` regions parameter (0, 4, 8, 12, 16).
`EXPECTED_SAU`	Expected number of `SAU` regions parameter (0, 4, 8).

Table 10-4 Execution testbench processor configuration options (continued)

Name	Description
EXPECTED_NUMIRQ	Expected value of NUMIRQ lines parameter <1-480>.
EXPECTED_IRQLVL	Expected value of IRQLVL Exception levels parameter <3-8>.
EXPECTED_IRQLATENCY	Expected split between high and low latency interrupts. See EXPECTED_IRQLATENCY parameter values table.
EXPECTED_IRQDIS	Expected value to disable support for individual interrupts. See EXPECTED_IRQDIS parameter values table.
EXPECTED_ITM	Expected CoreSight ITM configuration.
EXPECTED_MTB	Expected CoreSight MTB configuration.
EXPECTED_MTBAWIDTH	Expected MTB RAM address width.
EXPECTED_DBGLVL	Expected value of DBG parameter (0, 1, 2).
EXPECTED_WIC	Expected value of WIC parameter (0, 1).
EXPECTED_WICLINES	Expected value of WICLINES parameter <3-483>.
EXPECTED_CTI	Expected Cross Trigger Interface.
EXPECTED_INITSVTOR	Expected secure default initialization vector. See the *ARM®v8-M Architecture Reference Manual* for more information about the *Vector Table Offset Register* (VTOR).
EXPECTED_INITNSVTOR	Expected Non-secure default initialization vector. See the *ARM®v8-M Architecture Reference Manual* for more information about the *Vector Table Offset Register* (VTOR).
EXPECTED_RETENTION	Expected ability of the processor to support register value retention when the power is removed.
EXPECTED_MCUROMADDR	The expected value of the TEALDAP CoreSight Component pointer. Ensure that this configuration option matches the tied-off value of the corresponding signal. See *4.21.1 CoreSight ROM tables* on page 4-78 for information about how to determine this value for your design.

The following table shows the expected split between high and low latency interrupts parameter values.

Table 10-5 EXPECTED_IRQLATENCY parameter values

Parameter	Default value
EXPECTED_IRQLATENCY_479_448	0x00000000
EXPECTED_IRQLATENCY_447_416	0x00000000
EXPECTED_IRQLATENCY_415_384	0x00000000
EXPECTED_IRQLATENCY_383_352	0x00000000
EXPECTED_IRQLATENCY_351_320	0x00000000
EXPECTED_IRQLATENCY_319_288	0x00000000
EXPECTED_IRQLATENCY_287_256	0x00000000
EXPECTED_IRQLATENCY_255_224	0x00000000
EXPECTED_IRQLATENCY_223_192	0x00000000
EXPECTED_IRQLATENCY_191_160	0x00000000
EXPECTED_IRQLATENCY_159_128	0x00000000
EXPECTED_IRQLATENCY_127_96	0x00000000
EXPECTED_IRQLATENCY_95_64	0x00000000

Table 10-5 EXPECTED_IRQLATENCY parameter values (continued)

Parameter	Default value
EXPECTED_IRQLATENCY_63_32	0x00000000
EXPECTED_IRQLATENCY_31_0	0xFFFFFFFF

The following table shows the expected disable individual interrupts parameter values.

Table 10-6 EXPECTED_IRQDIS parameter values

Parameter	Default value
EXPECTED_IRQDIS_479_448	0x00000000
EXPECTED_IRQDIS_447_416	0x00000000
EXPECTED_IRQDIS_415_384	0x00000000
EXPECTED_IRQDIS_383_352	0x00000000
EXPECTED_IRQDIS_351_320	0x00000000
EXPECTED_IRQDIS_319_288	0x00000000
EXPECTED_IRQDIS_287_256	0x00000000
EXPECTED_IRQDIS_255_224	0x00000000
EXPECTED_IRQDIS_223_192	0x00000000
EXPECTED_IRQDIS_191_160	0x00000000
EXPECTED_IRQDIS_159_128	0x00000000
EXPECTED_IRQDIS_127_96	0x00000000
EXPECTED_IRQDIS_95_64	0x00000000
EXPECTED_IRQDIS_63_32	0x00000000
EXPECTED_IRQDIS_31_0	0x00000000

Related concepts

2.2 Configuration options on page 2-27.

Execution testbench processor tie-off options

The following table shows the execution testbench processor tie-off options.

Table 10-7 Execution testbench processor tie-off options

Name	Description
EXPECTED_BIGEND	Expected value of BE parameter. **Note** • The Cortex-M33 processor is in little-endian configuration. You must compile the execution testbench tests as little-endian. • If you are compiling tests using an ARM compiler, edit the COMPILE_BE variable in the tests/make.cfg in the tests directory to match.
EXPECTED_SSTCALIB	Secure SysTick calibration. The expected value of **CFGSSTCALIB[25:0]**. Ensure that this configuration option matches the tied-off value of the corresponding signal. For information about how to determine this value for your design, see *SysTick signals*.

Table 10-7 Execution testbench processor tie-off options (continued)

Name	Description
EXPECTED_NSSTCALIB	Non-secure SysTick calibration. The expected value of **CFGNSSTCALIB[25:0]**. Ensure that this configuration option matches the tied-off value of the corresponding signal. For information about how to determine this value for your design, see *SysTick signals* .
EXPECTED_CFGFPU	Expected Floating Point Unit Enable.
EXPECTED_CFGDSP	Expected DSP Extension Enable.
EXPECTED_CFGSECEXT	Expected Secure Extension Enable.
EXPECTED_MPUNSDISABLE	Expected value of **MPUNSDISABLE**.
EXPECTED_MPUSDISABLE	Expected value of **MPUSDISABLE**.
EXPECTED_SAUDISABLE	Expected value of **SAUDISABLE**.

Execution testbench DAP configuration options

The following table shows the execution testbench DAP configuration options.

Table 10-8 Execution testbench DAP configuration options

Name	Description
EXPECTED_TREVISION	Expected value of the TREVISION field of the `TARGETID` parameter.
EXPECTED_TPARTNO	Expected value of the TPARTNO field of the `TARGETID` parameter.
EXPECTED_TDESIGNER	Expected value of the TDESIGNER field of the `TARGETID` parameter.
EXPECTED_DPSEL	Expected Debug Port select

Execution testbench DAP tie-off options

The following table shows the execution testbench DAP tie-off options.

Table 10-9 Execution testbench DAP tie-off options

Name	Description
EXPECTED_INSTANCEID	Expected value of **INSTANCEID[3:0]**. Ensure that this configuration option matches the tied-off value of the corresponding signal. See *4.21.1 CoreSight ROM tables* on page 4-78.

Execution testbench CoreSight ETM-M33 configuration options

This section describes the execution testbench CoreSight ETM-M33 configuration options.

The following table shows the execution testbench CoreSight ETM-M33 configuration options. If you have not licensed the CoreSight ETM-M33, or do not include it in your configuration, you must set the ETM parameter to 0 in your RTL and in `EXECTB_Config.h`.

Table 10-10 Execution testbench CoreSight ETM-M33 configuration options

Name	Description
EXPECTED_ETM	Expected value of the `ETM` parameter

Execution testbench system ROM table

The following table shows the execution testbench system ROM table configuration options.

Table 10-11 Execution testbench system ROM table configuration options

Name	Description
`EXPECTED_CUST_JEPID`	Expected value of `teal_mcu_apb_rom_table.v` `JEPID` parameter. This value is your own JEDEC JEP-106 identity code value. See *4.21.1 CoreSight ROM tables* on page 4-78.
`EXPECTED_CUST_JEPCONT`	Expected value of `teal_mcu_apb_rom_table.v` `JEPCONTINUATION` parameter. This value is your own JEDEC JEP-106 continuation code value. See *4.21.1 CoreSight ROM tables* on page 4-78.
`EXPECTED_CUST_PART`	Expected value of `teal_mcu_apb_rom_table.v` `PARTNUMBER` parameter. This value is the part number value that enables you to identify your system. See *4.21.1 CoreSight ROM tables* on page 4-78.
`EXPECTED_CUST_REV`	Expected value of `teal_mcu_apb_rom_table.v` `REVISION` parameter. This value identifies the revision of your system, as defined by your part number value. See *4.21.1 CoreSight ROM tables* on page 4-78.

Execution testbench system ROM table tie-off options

The following table shows the execution testbench system ROM table tie-off options.

Table 10-12 Execution testbench system ROM table tie-off options

Name	Description
`EXPECTED_CUST_REVAND`	Expected value of `teal_mcu_apb_rom_table.v` **ECOREVNUM[3:0]**. Ensure that this configuration option matches the tied-off value of the corresponding signal. See *4.21.1 CoreSight ROM tables* on page 4-78.

Execution Testbench ECO and revision tie-off options

The following table shows the Execution Testbench processor tie-off options.

Table 10-13 Execution Testbench ECO and revision tie-off options

Name	Description
`EXPECTED_ECOREVNUM`	Expected value of **ECOREVNUM[51:0]**

Related concepts

2.2 Configuration options on page 2-27.

10.6.2 Test program compilation using the ARM or GCC compilers

To compile the test programs with the ARM or GCC compiler under Linux using the `make` command, you might have to modify the `Makefile` and `make.cfg` in the `testbench/execution_tb/tests` directory to suit your environment.

The following table lists the available options in the `make.cfg` file.

Table 10-14 make.cfg file options

Parameter	Default	Description
`COMPILE_BE`	0	Compile for a processor supporting big-endian.
`COMPILE_FPU`	1	Compile for a processor with an FPU.

Table 10-14 make.cfg file options (continued)

Parameter	Default	Description
`COMPILE_DSP`	1	Compile for a processor supporting DSP instructions.
`COMPILE_GCC`	0	Selects between ARM compiler 6 and GCC. **Note** The standard release of GCC supports little endian operation only. To run in big endian the gcc libraries must be recompiled.
`EXECTB_PRINTF`	0	Enable Dhrystone message printing.

You can change the compiler and linker tool name by changing the Makefile.

Test programs are compiled into the `tests` directory. The `make` command compiles:

- Binary `.elf` files for use with a debugger.
- Binary `.bin` files for memory initialization.
- ASCII `.inc` files that other C program files might include.
- ASCII `.rcf` (ARM ROM Code) files for use with ARM ROM models.
- ASCII `.disass` files contain the program in ARM assembly code.
- ASCII `.map` file links the instructions to the source subroutine.

To compile a test:

1. Change to the `testbench/execution_tb/tests` directory.
2. Use the `Makefile` in `testbench/execution_tb/tests` to compile the test programs. The command can be either:

`make <test_program>`	For individual tests.
`make all`	For all tests.

Where:

`<test_program>`
: Selects an individual test program to compile. See *Table 10-1 Test programs* on page 10-108 for a list of available tests.

`all`
: Compiles all the test programs into the current directory.

For example, to compile the test `hello_world.c`, type:

```
make hello_world
```

Note

- The ARM or GCC compiler must be on your path.
- If no individual test program is selected and the option `all` is not used with the `make` command, the default is to compile all the tests that are listed in *Table 10-1 Test programs* on page 10-108.
- To remove the compiled tests and prepare for a fresh compilation, type:

```
make clean
```

10.7 Configuration overview

The following table shows all the files that must be adjusted when making changes to the Teal configuration:

- The first column `TEAL_CONFIG.v` lists all the available Teal configuration parameters, see section 2.2 Configuration options.
- The next three columns list the parameters and defines in the execution testbench that must be changed to keep all the tests passing.
- `EXECTB_Config.h` sets the expected values for the `execution_tb` software tests and verifies they align with those set in `TEAL_CONFIG.v`.
- The defines in `execution_tb_defs.v` set the values applied to the teal configuration input signals see section 2.2 Configuration options.
- The `make.cfg` file in `testbench/execution_tb/tests` tells the compiler if the processor is configured to support FPU and DSP functionality, see *10.6.2 Test program compilation using the ARM or GCC compilers* on page 10-118.

Table 10-15 Teal configuration files

TEAL_CONFIG.v	EXECTB_Config.h	execution_tb_defs.v	make.cfg
FPU	EXPECTED_FPU, EXPECTED_CFGFPU	TEAL_EXECTB_CFGFPU TEAL_EXECTB_PDFPU TEAL_EXECTB_FPURET	COMPILE_FPU
DSP	EXPECTED_DSP, EXPECTED_CFGDSP	TEAL_EXECTB_CFGDSP	COMPILE_DSP
SECURE	EXPECTED_SECEXT, EXPECTED_CFGSECEXT	TEAL_EXECTB_CFGSECEXT	-
CPIF	EXPECTED_CPIF	-	-
MPU_NS	EXPECTED_MPU_NS, EXPECTED_MPUNSDISABLE	TEAL_EXECTB_MPUNSDISABLE	-
MPU_S	EXPECTED_MPU_S, EXPECTED_MPUSDISABLE	TEAL_EXECTB_MPUSDISABLE	-
SAU	EXPECTED_SAU, EXPECTED_SAUDISABLE	TEAL_EXECTB_SAUDISABLE	-
NUMIRQ	EXPECTED_NUMIRQ	-	-
IRQLVL	EXPECTED_IRQLVL	-	-
IRQLATENCY	EXPECTED_IRQLATENCY	-	-
IRQDIS	EXPECTED _IRQDIS	-	-
DBGLVL	EXPECTED_DBGLVL	-	-
ITM	EXPECTED_ITM	-	-
ETM	EXPECTED_ETM	-	-
MTB	EXPECTED_MTB	-	-
MTBAWIDTH	EXPECTED_MTBWIDTH	-	-
WIC	EXPECTED_WIC	-	-

Table 10-15 Teal configuration files (continued)

TEAL_CONFIG.v	EXECTB_Config.h	execution_tb_defs.v	make.cfg
WICLINES	EXPECTED_WICLINES	-	-
CTI	EXPECTED_CTI	-	-
RAR	-	-	-

10.8 Running the testbench

The `Makefile` in the `execution_tb` directory runs the simulation.

Type the following command from within the `execution_tb` directory to run one simulation:

```
make run SIM=<simulator> TESTNAME=<test> [PLUSARGS=”<plusargs_list>”]
```

To run all the tests type:

```
make all SIM=<simulator>
```

Where:

- `<simulator>` is one of:
  - — `mti` if you compiled the testbench using Mentor Questasim.
  - — `vcs` if you compiled the testbench using Synopsys VCS.
  - — `ius` if you compiled the testbench using Cadence IUS.
- `<test>` is the name of the test you compiled, see *Table 10-1 Test programs* on page 10-108.
- `<plusargs_list>` is a list of simulation plusargs.

Note

— The square braces around the `PLUSARGS` option indicates that it is optional. You do not type the square braces in the command. If you do not want to specify any `PLUSARGS`, do not use this option.

— The list of `PLUSARGS` must be enclosed within double quotes, otherwise the `Makefile` does not interpret them all as plusargs.

— You can configure the simulation options in the `make.cfg` or you can included them on the command line. For example, to run the `hello_world` test using the MTI simulation with tarmac trace enabled, use:

```
make run SIM=mti TESTNAME=hello_world TARMAC=yes
```

10.8.1 Running with a netlist

There are factors to be considered when running simulations with the `TEAL.v` replaced with a netlist.

To run simulations with the `TEAL.v` replaced with a netlist, ensure that:

- The `NETLIST` parameter in `make.cfg` is set.
- The file `logical/testbench/execution_tb/verilog/netlist.vc` points to the netlist at the required level.
- The instantiation of the netlist in the execution testbench does not take parameters. The parameters that are passed to the netlist instance are removed with define `ARM_TEAL_EXECTB_NETLIST`.

  Note

  The debug driver block also instantiates the netlist.

- The file `logical/testbench/execution_tb/verilog/netlist.vc` points to the appropriate technology libraries for netlist simulation.
- The rest of the execution testbench matches the configuration of the netlist. See *10.4 Configuring the testbench* on page 10-110.

If the netlist is run with SDF annotation, ensure that the appropriate Verilog defines are set correctly. See *10.4.1 Netlist considerations* on page 10-110.

Note

A netlist that is generated with the processor parameter `RAR` set to 0 has the potential to propagate Xs from uninitialized registers. This is caused by X-pessimism in gate-level simulation. When simulating with `RAR` set to 0, initialize all registers either by depositing a 0 or 1 value on the register outputs, or by modifying the flip-flop models in your gate-level library to initialize the models at the start of simulation.

Related concepts

10.4 Configuring the testbench on page 10-110.
10.4.1 Netlist considerations on page 10-110.

10.8.2 Running with a DSM

If you want run a DSM, you must edit the `execution_tb/make.cfg` file and the `execution_tb/verilog/dsm.vc` file. This will enable you to run simulations using a DSM instead of the TEAL.v supplied.

To run simulations with the TEAL.v replaced with a DSM:

1. In the `execution_tb/make.cfg` file ensure to:
   - Change the DSM parameter to `pli` or `dpi` as required.
   - Remove the comment from the `export DSM_MODEL_PATH` line and ensure this points to the location of the DSM model.
   - Set `SIM_64BIT` to align with the type of DSM generated.
   - Set `TARMAC` to `yes` if you require a tarmac log file.
2. In the `execution_tb/verilog/dsm.vc` file ensure to:
   - Uncomment the appropriate `-v` line.
   - Change the `<PATH to DSM>` to point to the location of the DSM model you want to use.
3. Ensure that you select the same configuration parameters in `TEAL_CONFIG.v` as the ones used when generating the DSM. The parameters used to build the DSM are available from the DSM release in `docs/TEAL_DSM_<name>_README.txt`.
4. Run make clean in the `execution_tb` directory when changing the DSM model.
5. Run make clean in the `execution_tb/tests` directory when the configuration has changed.

Related references

Chapter 13 DSM Generation on page 13-173.

10.8.3 Running with UPF

There are steps to follow to simulate the RTL or netlist with a Cortex-M33 processor constraints UPF file.

To simulate the RTL with a Cortex-M33 processor constraints UPF file:

1. Render the Cortex-M33 processor constraints UPF file, using the `RenderTeal_upf.pl` script with the following options:

   `./RenderTeal_upf.pl -debug 1 -fpu 1 -mtb 1`
2. Change to the execution testbench directory:

   `cd logical/testbench/execution_tb`
3. Configure the testbench to use UPF. In the `make.cfg` file, set:
   ```
   UPF :=yes
   ```
4. Compile the testbench and run tests as required. For example:
   ```
   make run SIM=mti TESTNAME=sleep
   ```

1. As an alternative to steps 3 and 4, you can compile and run tests with UPF as follows:
   ```
   make run SIM=mti TESTNAME=sleep UPF=yes
   ```

Related concepts

12.9 Rendering UPF constraints on page 12-171.

10.8.4 VCD generation

The execution testbench can generate a VCD file while running a simulation.

To enable this, ensure that:

- Verilog defines `ARM_TEAL_EXECTB_VCD` in `logical/testbench/execution_tb/verilog/netlist.vc`.
- Verilog defines `ARM_TEAL_EXECTB_VCD_START` and `TEAL_EXECTBVCD_STOP` in `logical/testbench/execution_tb/verilog/netlist.vc.`
- Verilog define `TEAL_EXECTB_VCD_FILE` in `logical/testbench/execution_tb/verilog/tbench.v`.

10.8.5 Simulation logs

Simulation log files are generated when a test program passes, fails, does not complete on time, or tries to test a non-existent feature of the Cortex-M33 processor.

When a test program passes in simulation, the following message appears in the simulation log:

```
** TEST PASSED OK **
```

When a test program fails in simulation, the following message appears in the simulation log:

```
** TEST FAILED **
```

When a test program does not complete within the time limit that is specified by `ARM_TEALIK_TIMEOUT_CYCLES`, the runaway simulation timer terminates the test and the following message appears in the simulation log:

```
** TEST KILLED **
```

When a test program tries to test a non-existent feature of the Cortex-M33 processor, it passes but prints the following message in the simulation log:

```
** TEST SKIPPED **
```

The simulation log files are generated in `testbench/execution_tb/logs`.

10.9 Measure power consumption

The Cortex-M33 power consumption can be measured using gate simulation.

Four execution testbench tests support power measurement:

- `dhrystone`.
- `maxpwr_cpu`.
- `saxpy_scalar`.
- `wfi`.

You can measure the power consumption of your Cortex-M33 processor implementation as follows:

1. Copy the test binary files from the `logical/testbench/execution_tb/tests/pre_compiled` directory to the `tests` directory, one level above. Alternatively, compile the required test, see *10.6 Configuring and compiling tests* on page 10-114.
2. Simulate the RTL design with tarmac generation enabled, see *10.8 Running the testbench* on page 10-122.
3. To locate the time to start and end power measurement, analyze the `tarmac` file. See *10.9.1 Locating VCD start and end points* on page 10-126.
4. To generate a VCD activity file between the power measurement points, run a netlist simulation. See *10.8.1 Running with a netlist* on page 10-122 and *10.8.4 VCD generation* on page 10-124.
5. To measure the power consumption, use the VCD file together with other data from the physical implementation flow.

10.9.1 Locating VCD start and end points

The power measurement points for `dhrystone`, `maxpwr_cpu`, `saxpy_scalar`, and `wfi` tests can be extracted from `tarmac` files.

You can find the measurement points for the `dhrystone` test as follows:

- `grep` the `tarmac` file for `MLA` or `MULS`.
- Use the timestamps of the last two grepped points as the start and end points respectively.

You can find the measurement points for the `maxpwr_cpu` and `saxpy_scalar` tests as follows:

- `grep` the `tarmac` file for `SEV`.
- Use the last two `SEV` instructions as the start and end points.

Note

To avoid boundary activity on the `maxpwr_cpu` and `saxpy_scalar` tests, ensure the measurement points set four clock cycles inside the `SEV` instructions.

You can find the measurement points for the `wfi` test as follows:

- `grep` the `tarmac` file for `WFI`.
- Set the start point to the time of the `WFI` instruction plus 100ns, equivalent to ten instructions.
- Set the end point to the start point plus 100ns, equivalent to ten instructions.

Related concepts

10.6 Configuring and compiling tests on page 10-114.
10.8 Running the testbench on page 10-122.
10.8.1 Running with a netlist on page 10-122.
10.8.4 VCD generation on page 10-124.

10.10 Debugging failing tests

All tests must pass, regardless of the configuration of the processor. If some tests are failing or being killed by the runaway simulation timer, debug the tests to determine the cause of the problem.

If the tests are running on the simulation testbench, you can debug the tests interactively using the functions available in your chosen simulator.

If the tests are running on hardware and a debugger is available, you can use the features that are provided by the debugger to identify where the test fails.

ARM recommends the following debug strategy:

Prioritize the failures

The simplest test is `hello_world`. If this test is among your failing tests, start debugging it first.

The `config_check` test is more complex, but it is the only test that checks the configuration of the Verilog RTL matches the expected values set in `EXECTB_Config.h`. If this test is failing, you must resolve the issues, because other tests assume that the `EXPECTED_*` values in `EXECTB_Config.h` are correct.

Check log files for errors or warnings

The test itself might indicate the cause of the failure, for example, a mismatch in expected and actual values for a parameter.

Check configuration

Check that the `EXPECTED_*` values in `EXECTB_Config.h` match the configuration of the processor. See *10.5.1 Configuring the execution testbench* on page 10-112

Enable message printing

By default, tests print progress and status messages to the simulation log using the simple character output device in the top-level testbench. You can disable this by commenting out the definition of `EXECTB_PRINTF` in `EXECTB_Config.h` before compiling the tests. You might want to do this to reduce the runtime of the tests.

If a test is failing, enable message printing by defining `EXECTB_PRINTF`, for example by uncommenting it in `EXECTB_Config.h`, and recompile and rerun the test to help determine the reason for failure.

You can also enable message printing from the debug driver module by defining `DEBUGDRIVER_PRINTF` in `EXECTB_Config.h` and recompiling the debug driver image. This can help you to debug an issue with a test that uses the debug driver, for example `config_check`, and `debug`.

Run on an unmodified version of the execution testbench to view the output that the tests produce.

Add your own debug messages

If message printing is enabled, you can insert more print messages into the code to help determine where the test is failing.

Compare executed instructions against the test code

If `tarmac` is used, RTL simulations produce two log files, <test_name>_tarmac0.log and <test_name>_tarmac1.log, of instructions that are executed on the processor in `execution_tb_mcu` and Debug Driver respectively. These can debug test failures by comparing the executed instructions with the assembly language of the test code. You can use the `*.disass` files that are generated when the code is compiled to help with debug. The `tarmac` files are placed in the logs directory and the name of the tests is appended to the front of the file name.

Note

See your compiler documentation if you are not using ARM or GCC compiler to compile your tests.

10.11 Modifying the execution testbench RTL for your SoC

This section describes the testbench structure and the Execution Testbench level, `execution_tb_mcu`, memory map. Read this section if you want to modify the Execution Testbench RTL for your own SoC requirements.

10.11.1 Testbench structure

The following figure shows the testbench structure and its instantiated execution testbench components.

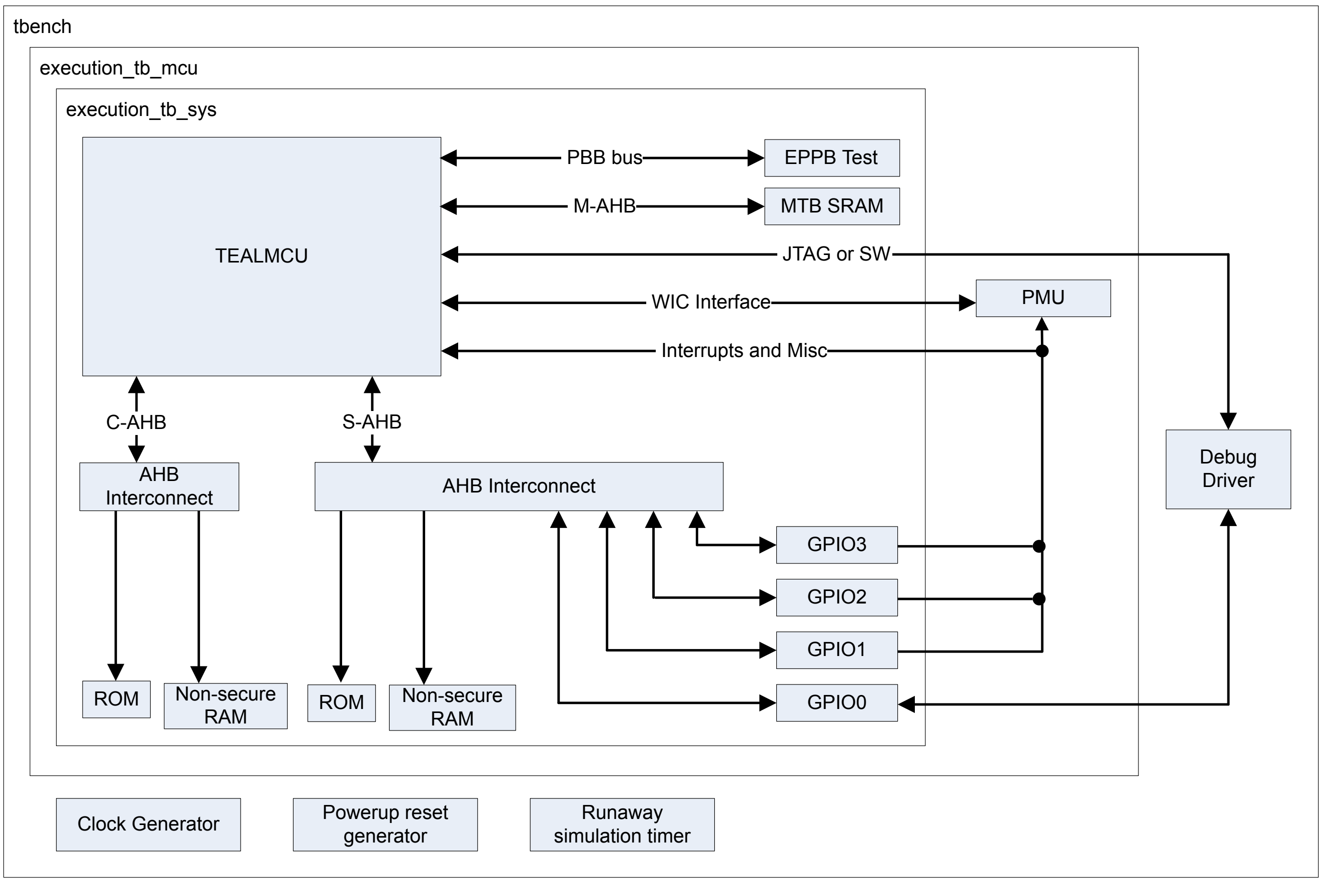


Figure 10-3 Execution testbench

The execution testbench contains these levels:

- TEALMCU level.
- execution_tb_sys level.
- execution_tb_mcu level.
- tbench level.

TEALMCU

This level instantiates the `TEAL`, the DAP, the TPIU, the system level ROM table, and an APB interconnect.

See *Appendix F TEALMCU* on page Appx-F-228 for more information on the `TEALMCU` and *1.1 About the processor* on page 1-16 for more information on `TEAL`.

The system level ROM table enables a debugger to uniquely identify the Cortex-M33 processor-based system, and enables debug components, such as the optional CoreSight ETM-M33, to be discovered automatically. See *4.21.1 CoreSight ROM tables* on page 4-78.

This example system ROM table has two entries:
- A pointer to the CoreSight ROM table.
- A pointer to the Cortex-M33 TPIU.

You must modify the instantiation of the module to use your own JEP-106 manufacturer ID value and a part number that identifies your system. See *Execution testbench system ROM table* on page 10-117.

execution_tb_sys level

The `execution_tb_sys` level contains various components, and allocates specific functions to GPIO signals.

The `execution_tb_sys` level contains:
- `TEALMCU` level.
- GPIO.
- AHB default slave.
- ROM controller.
- ROM.
- SRAM controller.
- SRAM.
- AHB bus interconnect.
- EPPB Test.
- MTB SRAM.
- Miscellaneous logic that is used for integration test purposes.

The `execution_tb_sys` level allocates specific functions to the following GPIO signals:

General Purpose Input Output 0

This GPIO has all 32 I/O pins that are routed to the testbench level. These signals:
- Indicate test completion and pass or fail status.
- Display messages using a simple character output device.
- Drive the SW or JTAG interface to the DAP using the debug driver block in the testbench.

General Purpose Input Output 1

This GPIO drives signals in the example `execution_tb_mcu`, for test purposes only. It drives power management unit signals and other processor signals.

General Purpose Input Outputs 2 and 3

This GPIO drives signals in the example `execution_tb_mcu`, for test purposes only. It acts as a source of interrupts for the WIC and processor. All GPIO interrupts drive **IRQ[0]** of the processor.

Related concepts

10.12.1 GPIO on page 10-133.

10.12.7 AHB default slave on page 10-134.

10.12.2 ROM controller on page 10-133.

10.12.3 ROM on page 10-133.

10.12.4 SRAM controller on page 10-133.

10.12.5 SRAM on page 10-133.

10.13 GPIO Integration on page 10-136.

10.13.1 GPIO 0 bit assignments on page 10-138.

10.13.2 GPIO 1 bit assignments on page 10-139.

10.13.3 GPIO 2 bit assignments on page 10-140.

10.13.4 GPIO 3 bit assignments on page 10-140.

execution_tb_mcu level

The `execution_tb_mcu` level is a simple example MCU.

It contains:
- *execution_tb_sys level* on page 10-129.
- *10.12.9 Power management unit* on page 10-134.

tbench level

The `tbench` level is the top-level testbench and contains:

- `execution_tb_mcu` level.
- Debug driver.
- Modules specific to `tbench` level:

Clock generator
This generates clocks for the `execution_tb_mcu` level.

Powerup reset generator
This generates Powerup reset.

Runaway simulation timer
This is used in simulation to end tests that have not completed within a specified cycle limit.

10.11.2 execution_tb_mcu memory map

The `execution_tb_mcu` memory map is characterized by 4MB of physical memory, three GPIOs, and an *External Private Peripheral Bus* (EPPB).

The following table shows the `execution_tb_mcu` memory map.

Table 10-16 Cortex-M33 memory map

Address range, inclusive	Region	Interface	Application
`0x00000000` - `0x000FFFFF`	Code	C-AHB	Memory Non-secure ROM
`0x00100000` - `0x001FFFFF`			Memory Secure RAM
`0x00200000` - `0x1FFFFFFF`			AHB default slave

Table 10-16 Cortex-M33 memory map (continued)

Address range, inclusive	Region	Interface	Application
`0x20000000` - `0x200FFFFF`	SRAM	S-AHB	Memory Non-secure RAM
`0x20100000` - `0x201FFFFF`			Memory Secure RAM
`0x20200000` - `0x3FFFFFFF`			AHB default slave
`0x40000000` - `0x400007FF`	Peripheral		GPIO0
`0x40000800` - `0x40000FFF`			GPIO1
`0x40001000` - `0x400017FF`			GPIO2
`0x40001800` - `0x40001FFF`			GPIO3
`0x40002000` - `0x5FFFFFFF`			AHB default slave
`0x50000000` - `0x5000003F`			IRQCHECK registers
`0x50000040` - `0x5FFFFFFF`			AHB default slave
`0x60000000` - `0x60000FFF`	External RAM		MTB RAM
`0x60001000` - `0x9FFFFFFF`			AHB default slave
`0xA0000000` - `0xDFFFFFFF`	External Device		-
`0xE0000000` - `0xE0000FFF`	PPB		ITM
`0xE0001000` - `0xE0001FFF`			DWT-Watchpoint
`0xE0002000` - `0xE0002FFF`			BPU-Breakpoint
`0xE0003000` - `0xE000DFFF`			Reserved
`0xE000E000` - `0xE000EFFF`			SCS-System control
`0xE000F000` - `0xE001EFFF`			Reserved
`0xE002E000` - `0xE002EFFF`			SCS - Non-secure alias
`0xE003E000` - `0xE003FFFF`			Reserved
`0xE0040000` - `0xE0040FFF`		EPPB	TPIU
`0xE0041000` - `0xE0041FFF`			ETM
`0xE0042000` - `0xE0042FFF`			CTI
`0xE0043000` - `0xE0043FFF`			MTB
`0xE0044000` - `0xE0044FFF`			EPPB test
`0xE0044000` - `0xE00FFFFF`			Reserved
`0xE00FE000` - `0xE00FEFFF`			MCU ROM default table
`0xE00FF000` - `0xE00FFFFF`			Processor ROM table
`0xE0100000` - `0xFFFFFFFF`	Vendor_SYS	S-AHB	AHB default slave

The `execution_tb_mcu` memory map contains:

- `execution_tb_mcu` instantiates 4MB of physical memory. It is split into four blocks of 1MB:
  - — The first 1MB, typically flash in an MCU, is read-only memory that holds the Secure code.
  - — The second 1MB, typically flash in an MCU, is SRAM memory that holds the Non-secure code.
  - — The third 1MB, typically SRAM, is where the Secure stack and heap are located.
  - — The fourth 1MB, typically SRAM, is where the Non-secure stack and heap are located.
- There are four GPIOs, each allocated 2KB of space. The GPIOs are connected to the S-AHB port in the execution testbench.
- The EPPB space of the Cortex-M33 processor is 1MB.

If any of these peripherals do not exist, the region maps to the APB default slave.

Note

The Processor and EPPB ROM tables are always present, but the other peripherals are optional.

- All remaining memory regions are mapped to the AHB default slave, including the reserved space. Any access to the default slave results in a fault.

10.11.3 Low-power implementation

To implement the low-power features supported by the Cortex-M33 processor, you might have to modify files in the execution testbench for your deliverables.

The following files are used by the execution testbench to implement the low-power features of the example system:

- `execution_tb_pmu.v`. The *Power Management Unit* (PMU), see *10.12.9 Power management unit* on page 10-134.
- `TEAL_constraints.upf`, `TEAL_configuration` and `testbench` processor implementation-dependent UPF files that are rendered using the `RenderTEAL_upf.pl` script, see *12.9 Rendering UPF constraints* on page 12-171.
- `execution_tb_mcu.v, execution_tb_sys.v` and `TEALMCU.v`. Example system structural files that connect the PMU to the level above the Cortex-M33 processor, see *10.11.1 Testbench structure* on page 10-128. The `execution_tb_mcu.v` file also contains power domain supply voltage control code that is required for power aware simulation.

You can modify all these files to suit your requirements. For example, if the MTB is not used in your Cortex-M33 processor configuration, ARM recommends that you modify these files accordingly.

For information about how to use the low-power modes, see *Chapter 11 Low-power Integration* on page 11-146 and *Chapter 12 Power Intent* on page 12-154.

10.12 Execution testbench components

This section describes the components that are supplied with the execution testbench and instantiated by the `TEALMCU.v` or `execution_tb_sys.v` levels.

Note

You can use these simple example components in your system and modify them to suit your requirements. In both cases, you are responsible for verifying the correct operation of the components in your system.

10.12.1 GPIO

The GPIO module, `logical/testbench/execution_tb/verilog/example_sys/execution_tb_ahb_gpio.v` is an example general-purpose I/O device. You can configure the GPIO pins individually as inputs or outputs.

You can configure the GPIO to generate an interrupt when specific input values change. See *Appendix A GPIO Programmers Model* on page Appx-A-182.

10.12.2 ROM controller

The ROM controller, `logical/testbench/execution_tb/verilog/example_sys/execution_tb_ahb_rom_bridge.v`, is an example AHB Read Only Memory bridge that guarantees zero wait-state responses to all AHB accesses.

10.12.3 ROM

The ROM model, `logical/testbench/execution_tb/verilog/example_sys/execution_tb_rom.v`, is an example behavioral Verilog model that supports image preloading from a file. The execution testbench test code image is loaded here.

Note

- `execution_tb_rom.v` can be replaced with a real ROM model.
- The `Makefile` flow generates `.rcf` format data suitable for ARM models.

10.12.4 SRAM controller

The SRAM controller model, `logical/testbench/execution_tb/verilog/example_sys/execution_tb_ahb_sram_bridge.v`, is an example AHB SRAM controller that guarantees zero wait-state responses to all AHB accesses by supporting write data buffer forwarding.

10.12.5 SRAM

The SRAM model, `logical/testbench/execution_tb/verilog/example_sys/execution_tb_sram.v`, is a synchronous SRAM.

10.12.6 AHB bus interconnect

The `logical/testbench/execution_tb/verilog/example_sys/execution_tb_ahb_interconnect.v` is an example module implementing AHB address decoding and multiplexing.

10.12.7 AHB default slave

The AHB default slave, `logical/testbench/execution_tb/verilog/example_sys/execution_tb_ahb_def_slv.v`, handles accesses to unused address locations in the system. The AHB default slave responds with an error each time it is accessed.

In the execution testbench, the 32-bit AHB buses are connected to AHB default slaves.

10.12.8 IRQCHECK registers

The IRQCHECK registers check that the `IRQLATENCY` and `IRQDIS` values align in both the `TEAL_CONFIG.v` and `EXECTB_Config.h` files.

10.12.9 Power management unit

The PMU, `logical/testbench/execution_tb/verilog/example_sys/execution_tb_pmu.v`, controls power domains, and it interfaces with the WIC and processor.

The PMU is an example system power controller that:

- Controls the following power domains:
  - The core power domain.
  - The debug power domain.
  - The MTB power domain.
  - The FPU in the processor.
- Interfaces with the WIC, to ensure that powerdown and wake-up behaviors are transparent to software.
- Interfaces with the processor to manage extended sleep and ensure clean powerdown.

The following table describes the primary outputs for an example power management unit.

Table 10-17 Example power management unit outputs

PMU signal	Description
CDBGPWRUPACK	The system is powered up and ready for debug.
WICENREQ	Request for WIC-based deep sleep.
COREQREQn	Core domain quiescence request signal.
FPUQREQn	FPU domain quiescence request signal.
DBGQREQn	Debug domain quiescence request signal.
MTBQREQn	MTB domain quiescence request signal.
nISOLATECORE	Isolation control for the core.
nRETNCORE	Retention control for the core.
nPWRUPRetnCORE	Power control for the core retention supply.
nPWRUPCORE_HAMMER	Power control for the core primary low impedance supply.
nPWRUPCORE_TRICKLE	Power control for the core primary current in-rush limited supply.
nISOLATEFPU	Isolation control for FPU.
nRETNFPU	Retention control for the FPU.
nPWRUPRetnFPU	Power control for FPU retention supply.
nPWRUPFPU_HAMMER	Power control for FPU primary low impedance supply.
nPWRUPFPU_TRICKLE	Power control for FPU primary current in-rush limited supply.
nISOLATEDEBUG	Isolation control for Debug.
nRETNDEBUG	Retention control for Debug.

Table 10-17 Example power management unit outputs (continued)

PMU signal	Description
nPWRUPRetnDEBUG	Power control for Debug retention supply.
nPWRUPDEBUG_HAMMER	Power control for Debug primary low impedance supply.
nPWRUPDEBUG_TRICKLE	Power control for Debug primary current in-rush limited supply.
nISOLATEMTB	Isolation control for MTB.
nPWRUPMTB_HAMMER	Power control for MTB primary low-impedance supply.
nPWRUPMTB_TRICKLE	Power control for MTB primary current in-rush limited supply.

10.12.10 Debug driver

The Debug driver module, `logical/testbench/execution_tb/verilog/execution_tb_debug_driver.v`, is provided to run tests on a Cortex-M33 processor-based subsystem. The `execution_tb_debug_driver.v` module is used in the execution testbench to provide SW or JTAG debug stimulus to `execution_tb_mcu`.

See *10.12.10 Debug driver* on page 10-135.

10.13 GPIO Integration

The execution testbench instantiates four GPIOs internally, that is, GPIO 0, 1, 2, and 3.

For more information about GPIO, see *Appendix A GPIO Programmers Model* on page Appx-A-182.

The following figure shows the connection of the GPIO interrupt lines.

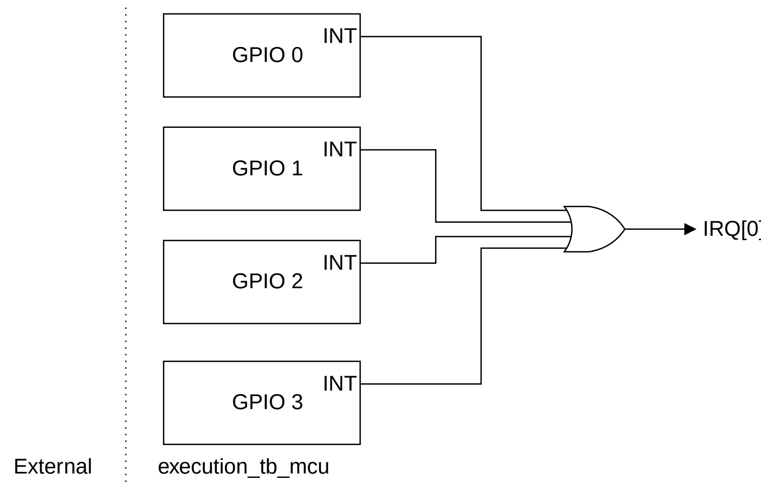


Figure 10-4 GPIO interrupt line connections

The following figure shows the GPIO 0 connections.

Note

IRQ[0] is the logical OR of the **INT** outputs from the four GPIOs. This enables the execution testbench interrupt tests to work even when the processor is configured to have only one external interrupt.

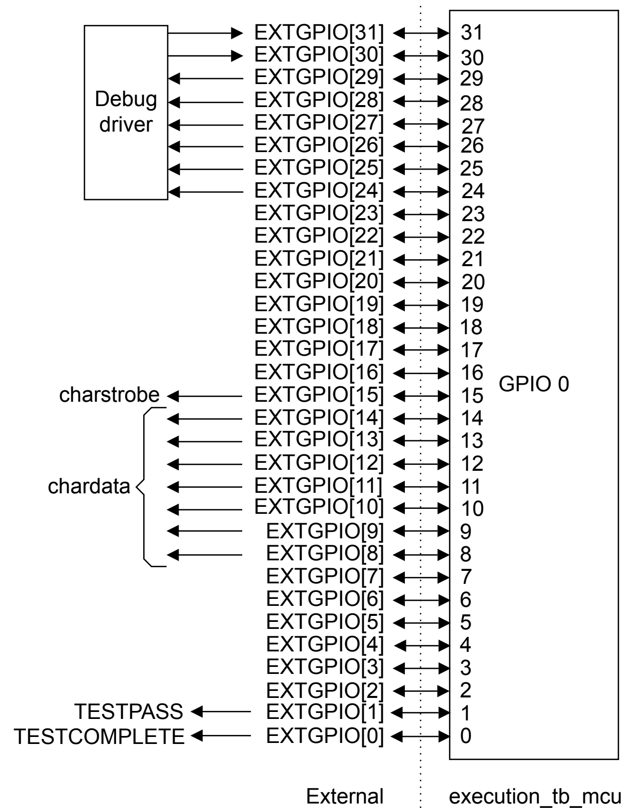


Figure 10-5 GPIO 0 connections

The following figure shows the GPIO 1 connections.

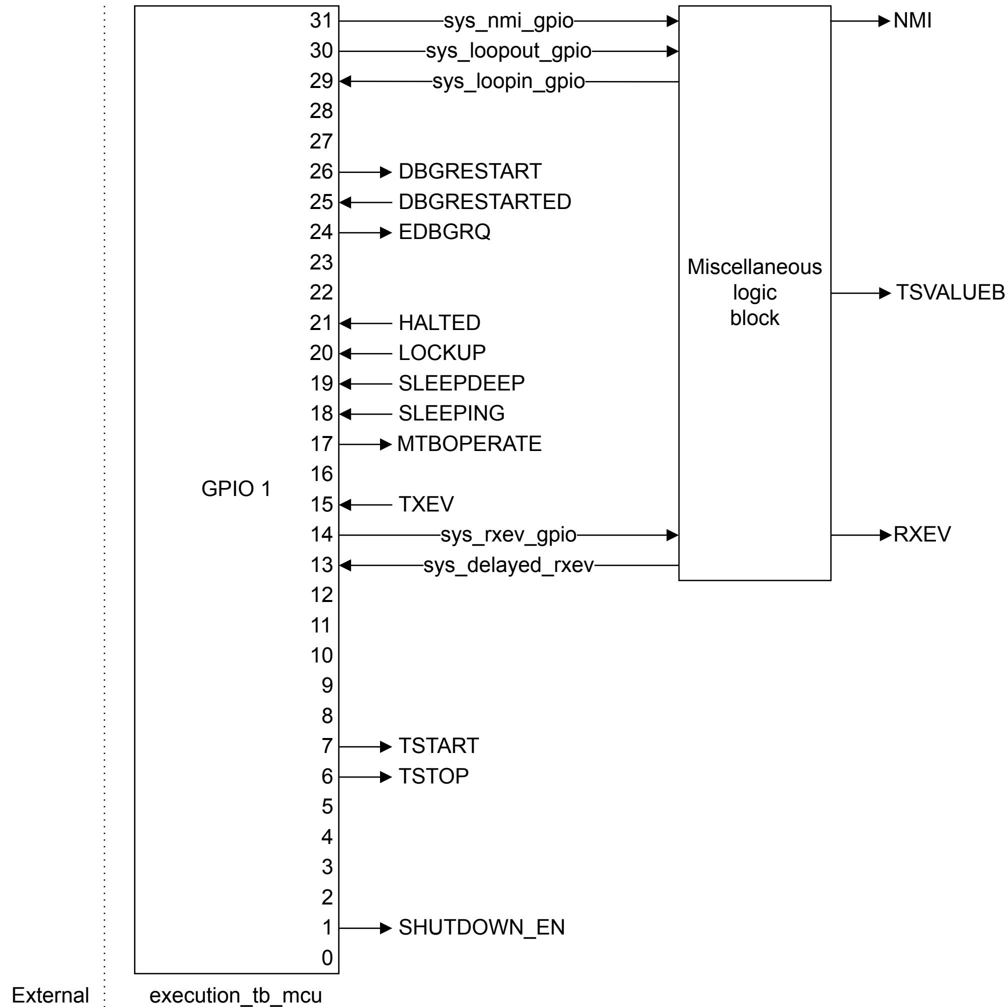
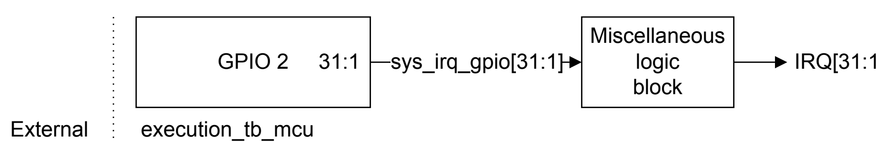


Figure 10-6 GPIO 1 connections

The following figure shows the GPIO 2 connections.



The following figure shows the GPIO 3 connections.

Figure 10-7 GPIO 2 connections

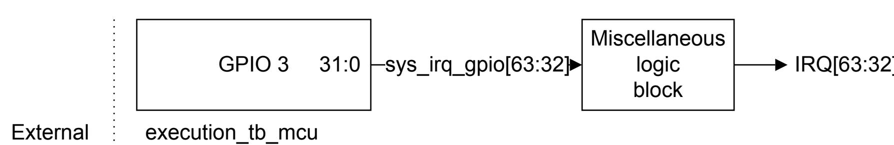


Figure 10-8 GPIO 3 connections

10.13.1 GPIO 0 bit assignments

GPIO 0 is connected to the external GPIO pins **EXTGPIO[31:0]**. GPIO 0 provides the I/O capabilities of the example MCU, with its signals routed to the top level of the MCU.

The **EXTGPIO** signals are used in the execution testbench to indicate test completion and pass or fail status. They also provide a simple character output device and control the debug driver block.

The following table shows the GPIO 0 bit assignments in the execution testbench.

Table 10-18 GPIO 0 bit assignments

Bit	Receive input from	Send output to	Connection information
[31]	**EXTGPIO[31]**	-	Running signal from Debug Driver
[30]	**EXTGPIO[30]**	-	Error signal from Debug Driver
[29]	-	**EXTGPIO[29]**	Function Strobe to Debug Driver
[28:24]	-	**EXTGPIO[28:24]**	Function Select to Debug Driver
[23:16]	-	-	Not used
[15]	-	**EXTGPIO[15]**	Connected to **charstrobe** in the top-level testbench
[14:8]	-	**EXTGPIO[14:8]**	Connected to **chardata** in the top-level testbench
[7:2]	-	-	Not used
[1]	-	**EXTGPIO[1]**	**TESTPASS** in the top-level testbench
[0]	-	**EXTGPIO[0]**	**TESTCOMPLETE** in the top-level testbench

10.13.2 GPIO 1 bit assignments

GPIO 1 is used internally to the example MCU to drive and monitor processor signals for testing purposes only.

These signals can drive, or can be driven by, other blocks in your SoC.

The following table shows the GPIO 1 bit assignments.

Table 10-19 GPIO 1 bit assignments

Bit	Receive input from	Send output to	Connection information
[31]	-	**NMI**	Drives the **NMI** pin of the Cortex-M33 processor, after a delay
[30]	-	Miscellaneous logic block	Drives **GPIOIN[29]**, after a delay
[29]	Miscellaneous logic block	-	Delayed version of **GPIOOUT[30]**
[28:27]	-	-	Not used
[26]	-	**DBGRESTART**	Drives the **DBGRESTART** pin of the Cortex-M33 processor
[25]	**DBGRESTARTED**	-	Connects to the **DBGRESTARTED** pin of the Cortex-M33 processor
[24]	-	**EDBGRQ**	Drives the **EDBGRQ** pin of the Cortex-M33 processor
[23:22]	-	-	Not used
[21]	**HALTED**	-	Connects to the **HALTED** pin of the Cortex-M33 processor
[20]	**LOCKUP**	-	Connects to the **LOCKUP** pin of the Cortex-M33 processor
[19]	**SLEEPDEEP**	-	Connects to the **SLEEPDEEP** pin of the Cortex-M33 processor
[18]	**SLEEPING**	-	Connects to the **SLEEPING** pin of the Cortex-M33 processor
[17]		**MTBOPERATE**	Connects to the **MTBOPERATE** pin of the Cortex-M33 processor
[16]	-	-	Not used
[15]	**TXEV**	-	Connects to the **TXEV** pin of the Cortex-M33 processor
[14]	-	**RXEV**	Drives the **RXEV** pin of the Cortex-M33 processor

Table 10-19 GPIO 1 bit assignments (continued)

Bit	Receive input from	Send output to	Connection information
[13]	Miscellaneous logic block	-	Delayed version of the **RXEV** pin of the Cortex-M33 processor
[12:2]	-	-	Not used
[1]	-	**SHUTDOWN**	Enable the PMU to shut down the processor
[0]	-	-	Not used

10.13.3 GPIO 2 bit assignments

GPIO 2 is used internally to the example MCU to drive **IRQ[31:1]** for testing purposes only.

These signals can be driven by other blocks in your SoC.

The following table shows the GPIO 2 bit assignments.

Table 10-20 GPIO 2 bit assignments

Bit	Receive input from	Send output to	Connection information
[31:1]	-	Miscellaneous logic block	Drives the corresponding **IRQ** pin of the Cortex-M33 processor, after a delay. For example, bit[31] drives **IRQ[31]**.
[0]	-	-	Not used.

10.13.4 GPIO 3 bit assignments

GPIO 3 is used internally to the example MCU to drive **IRQ[63:32]** for testing purposes only.

Other blocks in your SoC can drive these signals.

The following table shows the GPIO 3 bit assignments.

Table 10-21 GPIO 3 bit assignments

Bit	Receive input from	Send output to	Connection information
[31:0]	-	Miscellaneous logic block	Drives the corresponding **IRQ** pin of the Cortex-M33 processor, after a delay. For example, bit[31] drives **IRQ[63]**.

10.14 Debug driver

The debug driver block is a testbench component that controls the DAP using the JTAG or Serial Wire pins.

The debug driver contains an instantiation of the `TEALMCU` level, memories, and a GPIO. You can control the debug driver block using the pins **EXTGPIO[31:24]** of the MCU to initiate one of several predetermined operations. This arrangement enables testing of the execution testbench even when the processor in the MCU is sleeping.

The debug driver captures trace data from the processor. This is analyzed by the debug driver software to check the number of packets and packet data format.

The debug driver always executes the binary file `debugdriver.bin`, regardless of the test that is run on MCU. The `debugdriver.bin` binary must be built from the supplied source code. See *10.6.2 Test program compilation using the ARM or GCC compilers* on page 10-118 for information on how to compile and build the tests.

The following table lists the software source files that are used by the debug driver.

Table 10-22 Debug driver source files

File	Location	Description
`core_ARMv8MML.h`	`tests/CMSIS/Include/`	CMSIS file that defines the peripherals for the Cortex-M33 processor.
`core_cmFunc.h`	`tests/CMSIS/Include/`	CMSIS file that provides helper functions that access processor registers.
`debugdriver.h`	`tests/`	CMSIS device specific file that defines the peripherals for the `execution_tb_debugdriver` block.
`system_exectb_debugdriver.h`	`tests/Device/ARM/exectb_debugdriver/Source/ARM/`	CMSIS device vendor Header file that provides device-specific configuration for the `execution_tb_debugdriver` block.
`system_exectb_debugdriver.c`	`tests/Device/ARM/exectb_debugdriver/Source/ARM/`	CMSIS device vendor C file that provides device-specific configuration for the `execution_tb_debugdriver` block.
`boot_exectb_debugdriver.c`	`tests/Device/ARM/exectb_debugdriver/Source/ARM/`	This file provides the stack and heap initialization, vector table, default handlers and `_sys_exit` function that is used by `execution_tb_debugdriver`.
`retarget_exectb_debugdriver.c`	`tests/`	This file implements the functions necessary to retarget the C-library `printf()` function output to the `execution_tb_debugdriver` GPIO pins.
`debugdriver.h`	`tests/`	This header file contains various defines used by the debug driver block, including the GPIO pin allocations.
`debugdriver.c`	`tests/`	This is the main source code file for the debug driver block. It includes the routines for communicating with MCU through the GPIO and the routines to drive the MCU Serial Wire or JTAG interface.
`debugdriver_functions.h`	`tests/`	This header file contains a C `enum` representing the functions that the debug driver makes available to MCU.

The following figure shows the debug driver.

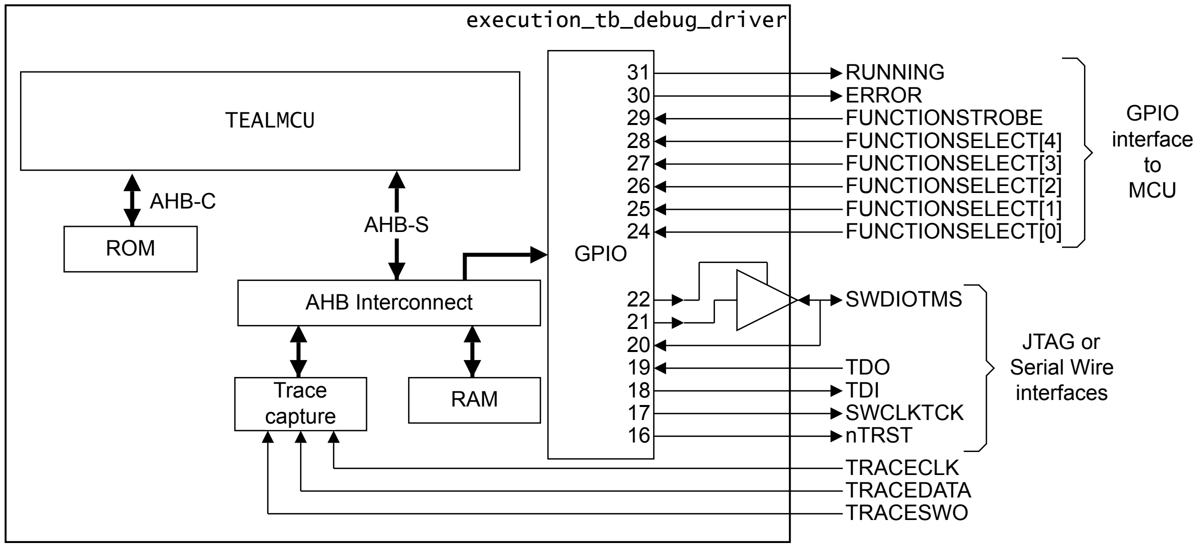


Figure 10-9 Debug driver

Note

- The timing wrapper in the debug driver is the same as that used in the MCU level.
- If any of the pins at the implementation level are modified, they must be suitably connected in the debug driver to preserve the intended behavior.

10.15 Modifying the execution testbench tests

The supplied test programs rely on the execution testbench GPIO to report test status, generate interrupts, and monitor status signals. The test programs are supplied in C source code with the execution testbench. The header of each source code file describes the test requirements of that code. You might have to modify these test programs to work in your SoC validation environment.

To run the test programs in a custom environment, modify the code to:

- Report test passed or failed.
- To generate external interrupts, use available peripherals.
- Define appropriate exception handlers for the system.

As supplied, the code uses some components external to the processor, to self-check some of the interface signals and to check the functionality of the processor. You must adapt the integration test programs to use the external components in your design for these tests. Omit one or more of the tests if your SoC validation environment:

- Does not use a particular interface.
- Does not support self-checking of one or more of the interface signals.

The tests supplied with the execution testbench are written in C and are compliant with the ARM Cortex Microcontroller Software Interface Standard (CMSIS). The CMSIS enables end users to write code that is portable across all ARM Cortex-M based microcontrollers.

The execution testbench includes a minimal subset of the CMSIS for the Cortex-M33 processor that is sufficient to support the supplied tests. See *http://www.arm.com/cmsis* for the full version of the CMSIS.

See the CMSIS documentation for information about how to provide your customer with the latest CMSIS library, and how to provide headers tailored to your Cortex-M33 processor-based device.

The following table shows the files in the `testbench/execution_tb/tests` and `testbench/execution_tb/tests/CMSIS` directories that constitute the CMSIS.

Table 10-23 CMSIS files

Filename	Location	Supplied by	Description
`boot_exectb_mcu.c`	`tests/Device/ARM/exectb_mcu/Source/ARM`	ARM	This file provides the stack and heap initialization, vector table, and default exception handlers. `boot_exectb_mcu.c` is provided as an example of a boot file written entirely in C. The CMSIS provides assembler example startup files that you can modify and use instead of `boot_exectb_mcu.c`.
`startup_exectb_mcu.S`	`tests/Device/ARM/exectb_mcu/Source/ARM`	ARM	This file provides the stack and heap initialization, vector table, and default exception handlers.`startup_exectb_mcu.S` is provided as an example of a boot file.
`ARMv8MML_SP.h`	`tests/CMSIS/Include`	ARM	Defines the core peripherals for the Cortex-M33 processor.
`core_cmfunc.h`	`tests/CMSIS/Include`	ARM	Defines the Cortex-M core register access functions.
`core_cminstr.h`	`tests/CMSIS/Include`	ARM	Defines the Cortex-M core instructions. That is, it defines functions that provide access to instructions that are not part of the C-language.
`core_cmsimd.h`	`tests/CMSIS/Include`	ARM	Defines compiler specific include files.
`cmsis_armclang.h`	`tests/CMSIS/Include`	ARM	ARM Compiler 6 include file.
`core_armv8mml.h`	`tests/CMSIS/Include`	ARM	Defines ARMv8MML Core Peripheral Access Layer.

Table 10-23 CMSIS files (continued)

Filename	Location	Supplied by	Description
cmsis_gcc	tests/CMSIS/ Include	ARM	GNU compiler include file.
system_ARMv8MML.h	tests/CMSIS/ Include	ARM	ARM Device system header.
exectb_mcu.h	tests/Device/ARM/ exectb_mcu/ Include	Device vendor	Device-specific file that defines the peripherals for the execution_tb_mcu example microcontroller device.
system_exectb_mcu.h	tests/Device/ARM/ exectb_mcu/ Include	Device vendor	Header file that provides device-specific configuration for the execution_tb_mcu example microcontroller device.
system_exectb_mcu.c	tests/Device/ARM/ exectb_mcu/ Include	Device vendor	C file that provides device-specific configuration for the execution_tb_mcu example microcontroller device.

The following table shows the test support files in the testbench/execution_tb/tests directory.

Table 10-24 Test support files

Filename	Description
config_check.c	This test verifies that the processor configuration matches the expected configuration values set in the EXECTB_Config.h file.
coprocessor.c	Exercises the demonstration FIR filter coprocessor module.
debug.c	This test checks DAP accesses and the pins **LOCKUP**, **EDBGRQ**, **HALTED**, **DBGRESTART**, and **DBGRESTARTED**.
debugdriver_functions.h	This header file contains a C enum representing the functions that the debug driver makes available to execution_tb_mcu.
debugdriver.c	This is the main source code file for the debug driver block. It includes the routines for communicating with execution_tb_mcu through the GPIO and the routines to drive the execution_tb_mcu SW or JTAG interface.
debugdriver.h	This header file contains various defines used by the debug driver block, including the GPIO pin allocations.
dhrystone_1.c	Dhrystone benchmark source code.
dhrystone_2.c	Dhrystone procedures and functions used by dhrystone_l.c.
dhrystone.h	This header file contains global definitions.
eppb.c	This test checks the operation of a simple memory on the EPPB bus interface.
idau.c	This test checks the operation of the IDAU security block.
saxpy_scalar.c	This test executes instructions to exercise Cortex-M33 maximum power consumption when executing floating-point instructions. Run this test on your netlist to get power values.
etm_trace.c	If ETM is implemented this test generates trace using the ITM, ETM and the Trace Port. Checking is limited to validating the protocol of the generated trace.
hello_world.c	This file provides some tests that the processor reads and checks its CPUID register and writes to the GPIO registers to print a simple message.
EXECTB_Config.h	This file includes some defines that you must edit to match the implemented configuration of the execution testbench.

Table 10-24 Test support files (continued)

Filename	Description
`EXECTB_tests.c`	This file provides the functions the execution testbench tests use to initialize the GPIOs and to communicate with the debug driver and `sys_exit()` function that updates the **TESTPASS** and **TESTCOMPLETE** signals when test code completes.
`EXECTB_tests.h`	This header file describes and defines the allocation of **GPIO** pins used by the `execution_tb_mcu` in the execution testbench. It also declares the function prototypes the execution testbench tests use to communicate with the debug driver.
`interrupt.c`	This test exercises **NMI**, **IRQ**, **TXEV**, and **RXEV**.
`itm_trace.c`	This test generates trace using the ITM and the Trace Port. Checking is limited to validating the protocol of the generated trace.
`Makefile`	This test enables you to build the execution testbench tests using the ARM compiler toolchain.
`maxpwr_cpu.c`	This test executes instructions that exercise the Cortex-M33 processor and maximize power consumption. Run this test on your netlist to get power values.
`mtb_trace`	This checks the program trace into the MTB memory.
`reset.c`	This test checks: 1. The access response to the AHB default slave. 2. The **SYSRESETREQ** pin resets the processor.
`retarget_exectb_mcu.c`	This implements the functions necessary to retarget the C-library `printf()` function output to the `execution_tb_mcu` **GPIO** pins.
`retarget_exectb_debugdriver.c`	This implements the functions necessary to retarget the C-library `printf()` function output to the `execution_tb_debug_driver` **GPIO** pins.
`romtable.c`	This test checks that it is possible for a debugger to autodetect the Cortex-M33 processor. The test uses the DAP to locate the architecturally defined ROM table to locate the processor. If you have included system level ROM tables, the content of these is displayed, but not checked.
`sleep.c`	This tests that the processor: • Enters sleep mode when **SLEEPING** and **SLEEPDEEP** outputs are asserted using the `execution_tb_mcu` **GPIO** pins. • Wakes from deep sleep mode by a debugger using the *Debug Halting Control and Status Register* (DHCSR). **Note** • If the UPF file does not include state retention, the sleep test is not supported for power aware simulation. • If state retention flip-flops are not used, the sleep test is not supported for power gated netlist simulation.
`wfi.c`	This test measures the minimum power when the processor is awaiting an interrupt.

Chapter 11
Low-power Integration

This chapter describes how to use the low-power features of the Cortex-M33 processor in your system.

It contains the following sections:

11.1 About low-power integration

Clock control, sleep signals, and implementing the supported sleep modes are described in this section.

For more information on these signals, see *4.2 Clocking and resets* on page 4-36, and *4.11.2 Power control and sleep interface* on page 4-65.

For an example of how to implement power gating in your processor, see *Chapter 12 Power Intent* on page 12-154.

For details of the architecturally defined low-power features of the processor, see the *Low-power modes* section in the *ARM® Cortex®-M33 Processor Technical Reference Manual*.

Related references

4.2 Clocking and resets on page 4-36.
4.11.2 Power control and sleep interface on page 4-65.
Chapter 12 Power Intent on page 12-154.

11.2 Processor operation in sleep mode

Sleep is a concept that applies only to software executing on the processor.

When the processor is in any of the supported sleep modes, it makes the following guarantees concerning software execution:

- The main execution pipeline is quiescent and no instruction fetches or software data accesses are initiated.
- There are no outstanding software transactions on any of the master interfaces and all software accesses committed before sleep mode entry have completed.

Debug accesses on D-AHB are independent of software execution and are therefore orthogonal to the processor being in sleep mode. This means that the processor can enter sleep mode while D-AHB accesses are ongoing and can remain in sleep mode while they are handled. The implications are:

- D-AHB accesses can cause transactions on the master interfaces while the processor is in sleep mode.
- D-AHB accesses can cause the internal processor state to be updated while the processor is in sleep mode.
- Debug activity in sleep mode enables gated clocks in the processor increasing the dynamic power used.

Therefore, the system is responsible for appropriate management of these interfaces when attempting to put a sleeping processor into a low-power state.

11.3 WIC Operation

The Cortex-M33 processor includes an optional WIC that can latch pending exceptions and detect wake-up conditions.

When implemented, the WIC can be:

Inactive
The internal clocks to the remainder of the processor can be clock gated. If the logic is implemented with state retention, the processor can be potentially powered down in a software transparent manner.

Active
The processor handshakes with the WIC to offload all prioritization information about exceptions before entering sleep mode.

For WIC-based operation, the system is required to establish the **WICENREQ** and **WICENACK** handshake with the processor and the SLEEPDEEP bit must be set in the SCR register before the processor enters sleep mode. While in WIC sleep mode, indicated by the Q-Channel **SLEEPING** and **SLEEPDEEP** signals, the events that can wake the processor are visible on the **WICSENSE** bus. When the WIC detects an appropriate event, it raises signal **WAKEUP** to wake the processor, and if powered down, must be powered-up. The **WAKEUP** signal is also considered in the Q-channel **COREQACTIVE** signal.

Note

- The **COREQACTIVE** signal requires **CLKIN** to be active for the **WAKEUP** event to be seen on the Q-Channel interface.
- The WIC can only be used to wake up a processor if the Core domain remains powered during sleep or if state retention is included in the implementation.

11.4 System requirements for low-power states

A low-power state is defined in this section as a state with one or more processor domains removed.

Implementing low-power states when the processor is in sleep mode imposes the following requirements on your system:

- Meeting the clocking requirements of the processor and associated system logic. The Cortex-M33 processor includes only a single clock, **CLKIN**, on the external interface. All the internal clocks are gated dynamically according to the current operating mode. **CLKIN** is used by the synchronization logic in the Q-channel interface and the WIC. This means if the Q-channel interface or the WIC is used in your design the clock can only be gated when the entire processor is powered down.
- Ensuring that no D-AHB accesses occur when in a low-power state. Any D-AHB accesses that occur while the core is powered down result in an AHB error. The *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2* specifies that an external debugger must, on physical connection to a device, request the system to turn on all power and clocks that are required to create and maintain a functional debug connection to the entire system. For Cortex-M33-based systems, this requires that the primary Core Power domain is active. The Debug and MTB power domains can be dynamically activated when required.

The CoreSight DAP provides the **CDBGPWRUPREQ** and **CDBGPWRUPACK**, and **CSYSPWRUPREQ** and **CSYSPWRUPACK** signals to implement this handshake between the debugger and the system PMU. For more details, see the *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2*.

There can be a requirement to negotiate the race condition on entry into a low-power state, where the processor can wake up between the cycles system commits to entering this state and the cycles the clock and power are turned off. To avoid race condition, the Q-channel interface handshake ensures that the Cortex-M33 processor is kept in sleep even in the presence of a wake-up condition. In this case, as a result of the wake-up condition, **COREQACTIVE** is asserted. For systems that are not using the Q-channel interface the **SLEEPHOLDREQn** and **SLEEPHOLDACKn** handshake is supported. For more information on this handshake, see *11.5 Sleep-hold interface* on page 11-151.

Related concepts

11.6 Supported sleep modes on page 11-152.
11.5 Sleep-hold interface on page 11-151.

Related references

4.2 Clocking and resets on page 4-36.

11.5 Sleep-hold interface

The sleep-hold interface helps the system to resolve the race condition between the following two events:

- The system commits to entering a low-power state.
- An event outside the direct control of the system causes the processor to wake up.

Failure to correctly negotiate this race condition might result in the processor:

- Resuming execution before the system enters a low-power state.
- Causing a loss of data or corruption of program flow.

The protocol for the sleep-hold handshake is defined as:

- **SLEEPHOLDREQn** assertion is only recognized when **SLEEPING** is HIGH. Therefore, ARM recommends that **SLEEPHOLDREQn** is driven LOW only when **SLEEPING** is HIGH.
- **SLEEPHOLDREQn** must remain asserted at least until either **SLEEPHOLDACKn** is asserted, or **SLEEPING** goes LOW.
- When **SLEEPHOLDACKn** is asserted, **SLEEPHOLDREQn** must remain asserted until entry into the low-power state is safely complete and for the entire duration of the low-power state. The processor guarantees that **SLEEPHOLDACKn** remains asserted and the processor remains in sleep state until **SLEEPHOLDREQn** is deasserted.
- On detection of a wake-up event, the system must wait until all required clocks and power have been safely restored before deasserting **SLEEPHOLDREQn**. Detection of a wake-up event depends on the sleep mode that is used, see *11.6 Supported sleep modes* on page 11-152.

Related concepts

11.6 Supported sleep modes on page 11-152.

11.6 Supported sleep modes

Different sleep modes allow the system to make different trade-offs between power saving and wake-up latency to support a wide range of usage models.

The supported sleep modes are:

- Standard sleep.
- Deep sleep.
- WIC sleep.

Note

- The system could implement multiple distinct low-power states within a given sleep mode. This is entirely system defined and invisible to the processor as long as the requirements for the relevant sleep mode are met.
- The **CLKIN** signal must run in all three sleep modes.
- The figures in this section do not show the additional logic that is required to exit sleep mode for a DAP powerup request.

11.6.1 Standard sleep

In this mode, the WIC is inactive and it is the NVIC that is responsible for monitoring incoming interrupts and waking up the processor.

The internal clock signals are gated to save power.

11.6.2 Deep sleep

The processor regards deep sleep mode to be identical to standard sleep mode.

This mode allows deeper levels of sleep that are entirely system defined. For example, deep sleep can be used to shut down a system PLL, and for switching to a low frequency clock, to achieve greater power savings at the expense of wake-up latency.

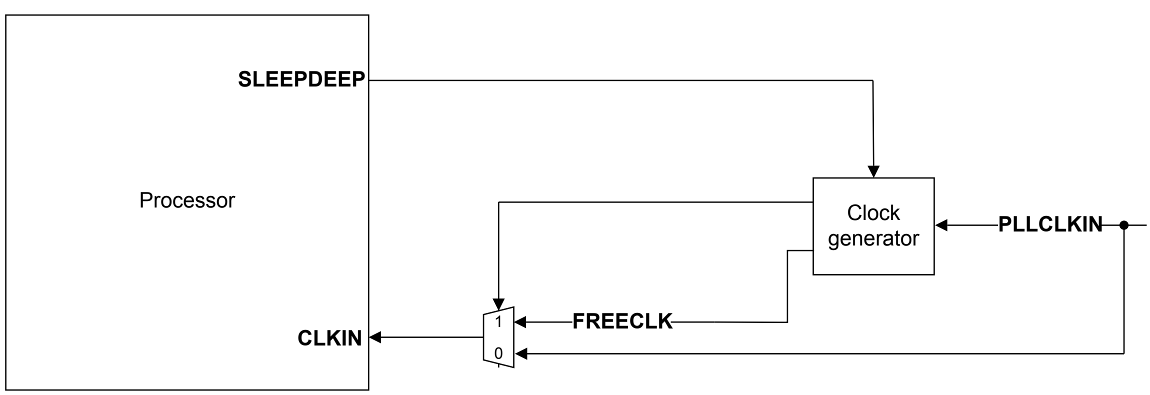


Figure 11-1 Example deep sleep clock control

11.6.3 WIC sleep

WIC sleep mode enables you to use the wake-up features in the processor after the processor core is powered down in WIC sleep mode. In WIC sleep mode, the WIC is active and is responsible for registering pending exceptions and detecting wake-up conditions. The NVIC is inactive.

In this case, **SLEEPDEEP** must be enabled in the *System Control Register* (SCR) and the system must request that the NVIC handshakes with the WIC to off-load all prioritization information about exceptions before entering sleep mode. This is performed using the **WICENREQ** and **WICENACK** handshake signals, see the following figure for more details. The **WICENREQ** and **WICENACK** handshake is normally performed sometime before the processor enters WIC sleep.

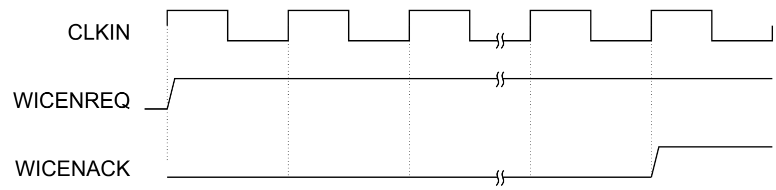


Figure 11-2 WICENREQ and WICENACK handshake signals

Software-transparent powerdown is only supported in WIC sleep by using state retention in the processor.

The following figure shows an example of WIC sleep clock control.

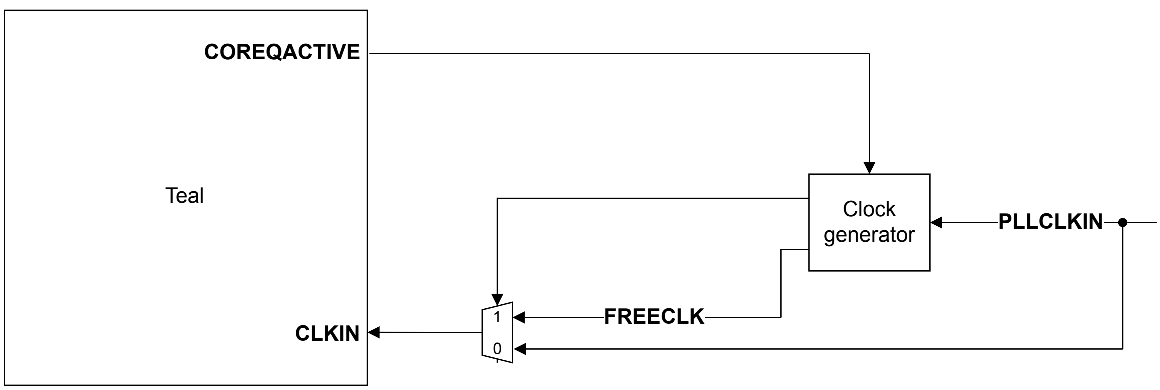


Figure 11-3 Example WIC sleep clock control

Related references

Chapter 12 Power Intent on page 12-154.

Chapter 12
Power Intent

This chapter describes the optional power gating features of the processor and how the logic can be divided into different power domains.

It contains the following sections:

12.1 About Cortex®-M33 processor power intent

This section describes the example power intent specification that is provided in the `logical/teal/power_intent/upf directory`.

This is an example of a fully featured power-gated design, but you can use a simpler design for your Cortex-M33 processor implementation. For example, you can combine power domains together but you must use valid power states as described in *12.3 Power states* on page 12-158.

The execution testbench provides an example of how you can implement power gating control in your system, including an example PMU and a sleep test.

Related concepts

12.3 Power states on page 12-158.

Related references

Chapter 10 Execution Testbench on page 10-104.

12.2 Power domains

The Cortex-M33 processor can be partitioned into several power domains as shown in the following figure.

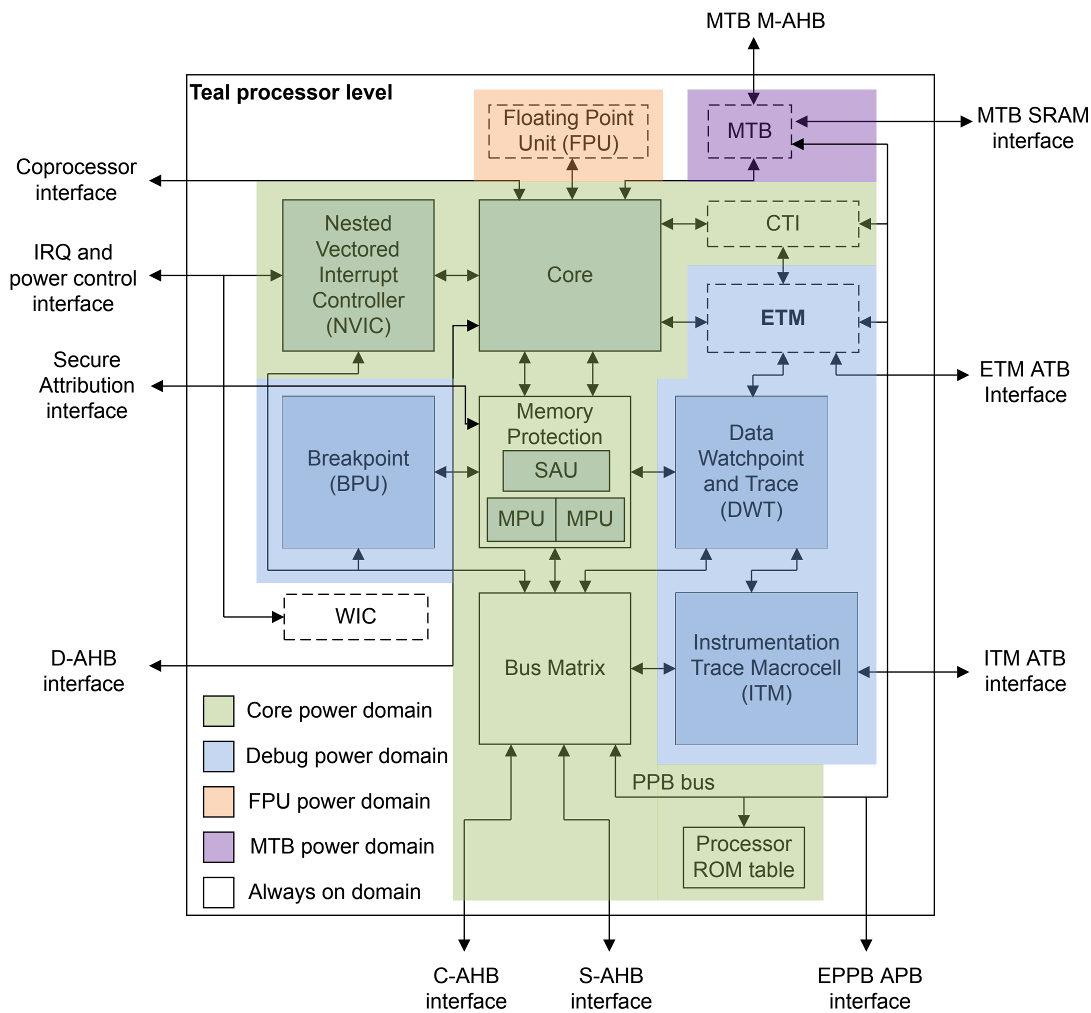


Figure 12-1 Cortex-M33 processor power domains

The five power domains that are used in this example are described in the following table.

Table 12-1 Power domain description

Power Domain	Description
Always-on	This includes the WIC, clock gates, and Q-Channel interface.
Core	This includes the Core, MPU, SAU, CTI, and NVIC.
FPU	This contains the FPU.
Debug	This includes the BPU, DWT, ITM, and ETM.
MTB	This includes the MTB.

Note

It is possible to merge one or more of the FPU, MTB, and Debug power domains into the Core power domain.

12.3 Power states

The power domains can be controlled to give different combinations of powered up and powered down domains. However, only some powered up and powered down domain combinations are valid and supported.

Table 12-2 Supported power states on page 12-158 shows the valid power states of the Cortex-M33 processor. It shows that:

1. FPU can only be powered when the core is powered up.
2. Debug can never be powered up without the core powered up.
3. MTB status is independent of other power modes, apart from shutdown mode.

Note

- If the Core domain includes retention logic and the debug domain does not the debug state is lost when software transparent power states are used.
- The MTB can operate as a SRAM interface using the M-AHB interface. For this purpose, a separate MTB power domain is included that enables access to the SRAM when the majority of the Cortex-M33 processor logic is powered down.

  The FPU can be put in retention when the core is powered to gain opportunistic power savings when no floating-point operations are being used in the current context.

These restrictions mean that clamps between these domains are only required in one direction apart from the Core/MTB domain interface.

Note

All transitions between power states must remain in the legal states that the following table shows.

Table 12-2 Supported power states

State	Always-on	Core	FPU	Debug	MTB
Run mode without FPU, trace, or debug	ON	ON	OFF	OFF	ON/OFF
Run mode with FPU but without trace or debug	ON	ON	ON/RET	OFF	ON/OFF
Run mode with debug and ETM but without FPU	ON	ON	OFF	ON/RET	ON/OFF
Run mode with FPU and with debug and ETM	ON	ON	ON/RET	ON/RET	ON/OFF
SW transparent power down with FPU	ON	RET	RET	OFF/RET	ON/OFF
SW transparent power down with FPU off	ON	RET	OFF	OFF/RET	ON/OFF
SW supported power down	ON	OFF	OFF	OFF	ON/OFF
MTB SRAM access when processor shutdown	ON	OFF	OFF	OFF	ON
Shutdown mode	OFF	OFF	OFF	OFF	OFF

The following table shows the Cortex-M33 processor power domains states and their behaviors.

Table 12-3 Power state description

Power state	Description
OFF	Block is power gated.
RET	Logic retention power only. **Note** MTB does not support retention mode.
ON	Block is active.

12.4 Q-Channel control

This section is based on the assumption that the Q-Channel logic is used to control the power domains in the Cortex-M33 processor. The alternative Sleep-hold interface is not considered.

For more information on the Sleep-hold interface, see *11.5 Sleep-hold interface* on page 11-151.

12.4.1 Responsibilities

All Q-Channel interfaces consist of the following signals:

- **QACTIVE**.
- **QREQn**.
- **QACCEPTn**.
- **QDENY**.

This gives the device the ability to indicate the need for power for Q-channel interfaces (**QACTIVE**) and also the ability to prevent power being removed (**QDENY**). Each domain can drive the signals differently and this is detailed in the subsequent sections.

In each Q-Channel state, the system power controller and Cortex-M33 processor have specific responsibility for power, clocking, and reset operation and is shown in the following table.

Table 12-4 Q-Channel states with controller and processor responsibilities

<table>
<tr><th>Q-Channel state</th><th>Controller responsibilities</th><th>Processor responsibilities</th></tr>
<tr><td>Q_STOPPED</td><td>When required, provide or remove:
• Domain power.
• Processor clock and power on reset.</td><td>Guarantee:
• Clocks remain gated.
• Reset remains static while the external reset is not asserted.</td></tr>
<tr><td>Q_EXIT</td><td rowspan="3">Guarantee:
• Domain power.
• Processor clock availability.
• Processor power on reset deasserted.</td><td>Guarantee:
• De-assertion of domain power on reset (if necessary).
• Enabling of domain clocks.</td></tr>
<tr><td>Q_RUN Q_DENIED
Q_CONTINUE</td><td>Guarantee:
• Domain power on reset deasserted.
• Domain clocks enabled.[b].</td></tr>
<tr><td>Q_REQUEST</td><td>If accepting the powerdown:
• Assert domain power on reset unless configured for retention.
• Disable domain clocks.</td></tr>
</table>

Many domains support state retention. Only full state retention is supported and this information must be communicated to the control logic to avoid incorrect assertion of reset. For more information, see *12.6.1 State retention support* on page 12-166

A domain is only guaranteed to be fully operational in the Q_RUN, Q_DENIED, and Q_CONTINUE states. Therefore, a domain only considers another domain as fully powered up when it is in one of these three states.

12.4.2 Dependencies

Multiple domains might want to be powered up or down at the same time. For example, transitioning to the shutdown state when all domains are powered up. A system power controller could achieve this by using different strategies.

[b] The processor might gate the clock off during the special retention case. For more information, see *12.4.4 Software transparent retention* on page 12-164

- Powerup or powerdown each domain fully in sequence, completing the required behavior on both sides of the Q-channel interface before proceeding to act on the next domain.
- Do each side of the Q-Channel interface at the same time.
  — Powerup: Update all relevant Q-Channels when all relevant domains have power.
  — Powerdown: Wait for all relevant Q-Channels to accept powerdown before removing any power.
- Do each as quickly as possible before proceeding as if there is no dependency between each domain.

Note

A system power controller that provides or removes power is generally separate from the Q-Channel handshake. All the approaches listed previously still require a system power controller to behave sensibly in providing power to the processor regardless of how the Cortex-M33 processor responds on the Q-Channels.

If the power domains are independent, a system power controller can power them up or down in any order. However, if the valid power states of the processor define a dependency between the domains, the system power controller must ensure to order the powerup or powerdown requests to the individual power domains. To reduce the overhead in the system power controller, the Cortex-M33 processor uses the valid power states to control the Q-Channel responses for dependent power domains.

The Cortex-M33 processor:

- Delays a powerup request for a domain if it is dependent on another being powered up first by stalling in the Q_EXIT state. In this case, the **QACTIVE** for the related domain is asserted.
- Denies a powerdown request for a domain if it is dependent on others being powered down first.

In the Cortex-M33 processor, the only dependencies are between the Core power domain and either the FPU or debug power domains.

The **QREQ** inputs are synchronized as soon as they enter the processor. If multiple **QREQn** changes occur at the same time or a short time apart, the order of the changes seen by the processor is unpredictable. The following table summarizes the Q-Channels behavior in all such cases. In the following table, D2 is a domain dependent on D1.

Table 12-5 Q-Channel behavior for dependent domains

Current state (D1/D2)		State after QREQn changes		
D1	**D2**	**D1 then D2**	**D2 then D1**	**Together**
Stopped	Stopped	Run/Run		
Run	Stopped	Stopped/Exit[c]	Denied/Run[d]	
Run	Run	Denied/Stopped[d]	Stopped/Stopped	
Run	Denied	Denied/Run		
Denied	Stopped	Run/Run		
Denied	Run	Run/Stopped		
Denied	Denied	Run/Run		

c In this case, the end state for D2 is Q_EXIT which requires the processor to respond. however, it does not respond till the PMU attempts to powerup D1 (allowing the dependent domain to be powered up). Core power domain, **QACTIVE** is asserted to ensure the PMUs do not lockup. This case is not straightforward and predictable. The end result might not be the intended behavior and the system power controller has the responsibility of resolving the situation.

d This case is not straightforward and predictable. The end result might not be the intended behavior and the system power controller has the responsibility of resolving the situation.

Note

The information in the table assumes that powerdown requests are only denied if a dependent domain requires power.

12.4.3 Domain behavior

This section describes the domain behavior.

Always-on domain

The always-on domain does not have a Q-Channel interface. It must be on for other domains to be on. Clock gating, reset generation, and Q-Channel control is in this domain so no other domain can be powered up without it.

Core power domain

The following table shows the Q-channel behavior for the Core power domain.

Table 12-6 Core power domain Q-Channel behavior

Core power domain Q-Channel behavior	
QACTIVE asserted	When powered up or powered down in retention: • Not in WIC sleep, and a trigger request on **CTICHIN**.[e] • In WIC sleep and a wakeup request occur. When powered up: • Not sleeping. • Sleeping with a wakeup request. • Debug access in progress (D-AHB). When powered down: • Q-Channel request to powerup Debug or FPU power domain.
Powerup request	Always accepted.
Powerdown request	Denied if any of the following are true: • Core power domain **QACTIVE** is asserted. • The FPU or Debug domains are powered up. Otherwise, accepted.

Note

Accesses to the D-AHB slave interface while the Core power domain is powered down returns a bus error.

FPU power domain

The following table shows the Q-Channel behavior of the FPU power domain.

[e] When the CTI is included in the Cortex-M33 processor.

Table 12-7 FPU power domain Q-Channel behavior

FPU power domain Q-Channel behavior	
QACTIVE asserted	At any time, FPU enabled through the CPACR. When powered up: • Outstanding FPU context indicated by CPPWR.SU10 and retention not supported in the FPU. • FPU is executing an instruction. **Note** **QACTIVE** is deasserted if the processor is in WIC sleep and the Core and FPU or Debug power domains are configured for retention unless the FPU is executing an instruction.
Powerup request	Stalled (in Q_EXIT) if the Core power domain is not powered up. Otherwise, stalled (in Q_EXIT).
Powerdown request	Denied if the FPU power domain **QACTIVE** is asserted. Otherwise accepted.

Debug power domain

The following table shows the Q-channel behavior of the debug power domain.

Table 12-8 Debug power domain Q-Channel behavior

Debug power domain Q-Channel behavior	
QACTIVE asserted	At any time: • PPB access to a slave in the debug power domain. — The access is stalled until powerup has completed. • The DEMCR.TRCENA bit is set. When powered up: • BPU is enabled. • ETM is enabled. • ITM, DWT, or ETM are active (have buffered trace data). **Note** **QACTIVE** is deasserted if the processor is in WIC sleep and the Core and FPU or Debug power domains are configured for retention unless the Debug domain has buffered trace.
Powerup request	Accepted if the Core is powered up. Otherwise, stalled (in Q_EXIT).
Powerdown request	Denied if the Debug power domain **QACTIVE** is asserted. Otherwise, accepted.

MTB domain

The following table shows the Q-channel behavior for the MTB power domain.

Table 12-9 MTB power domain Q-Channel behavior

MTB power domain Q-Channel behavior	
QACTIVE asserted	At any time: • PPB access to the MTB. When powered up: • AHB access to the MTB. — Accesses while the domain is powered down generate a bus error. • MTB MASTER.EN bit is 1. • MTB has buffered trace.
Powerup request	Always accepted.
Powerdown request	Denied if the MTB **QACTIVE** is asserted. Otherwise, accepted.

Note

Accesses to the M-AHB MTB slave interface while powered down returns a bus error.

12.4.4 Software transparent retention

The normal Q-Channel behavior is to treat each domain independently with a few constraints around domains with dependencies.

For more information, see *12.4.3 Domain behavior* on page 12-162. The Cortex-M33 processor supports one exception to this behavior to provide support for software transparent powerdown.

This behavior is visible when attempting to power the Core domain down into retention. Typically, the Debug and FPU power domains both need to be powered down first. This would require actively disabling both domains to allow their **QACTIVE** signals to be deasserted and a powerdown request to be accepted. However, if the FPU and Debug power domains support state retention, the Cortex-M33 processor removes the requirement to explicitly disable the FPU and Debug domains. The behavior of this function is as follows:

- The processor enters sleep mode and the Core power domain deasserts **QACTIVE**.
- The processor detects that the Core power domain and any powered up dependent domain are configured to be powered down into retention.
- Assuming the dependent power domains are not active (they can still be enabled), then the processor deasserts the **QACTIVE** signals for these power domains.
- Powerdown occurs in a normal manner (dependent domain first) with the following exceptions:
  - The clocks for all affected domains are gated at the same point to keep the state synchronized.
  - Any Debug access on D-AHB returns a bus error as if the domain was powered off. This makes the process non-transparent to some degree, but the debug case is considered acceptable as it is not the typical use case.
- Powerup occurs in a normal manner (dependent domains last). The only exception is the clocks for all affected power domains are gated until all domains have been powered up.

12.5 Clock gating

The Cortex-M33 processor has a number of internally generated clocks which must be gated when the corresponding domain is powered down. The following figure shows a simplified version of the clock gating structure.

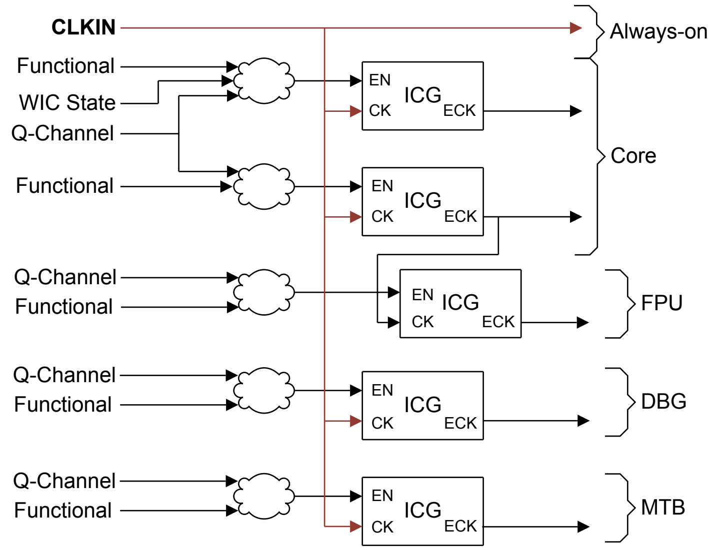


Figure 12-2 Teal internal clock generation

Each clock gate is controlled by a combination of the current Q-Channel state and any functional enables from the domain. In the case of the Core domain, the WIC state (from the always-on domain) is also used. The Q-Channel state ensures the clocks are properly enabled and disabled during the powerup and powerdown sequences.

Note

These are just the clock gates in the always-on domain. All other domains use their own clock gates where necessary to reduce power.

12.5.1 System level clock gating

When the main processor clock is gated, accesses cannot be generated on the C-AHB or S-AHB buses. This implies that it is possible to clock gate components immediately downstream of the processor to save power. If the processor generates an AHB access after waking up, the downstream slave must accept the address. Therefore, if the slave is clock gated when the processor is clock gated, it must also be ungated at the same time as the processor. To support this, the Cortex-M33 processor includes an output signal **CORECLKEN** to synchronize the system level clock gating with the processor.

12.6 Reset generation

The Cortex-M33 processor includes logic to generate the appropriate internal reset to each domain when powered up. The following diagram shows the reset generation structure.

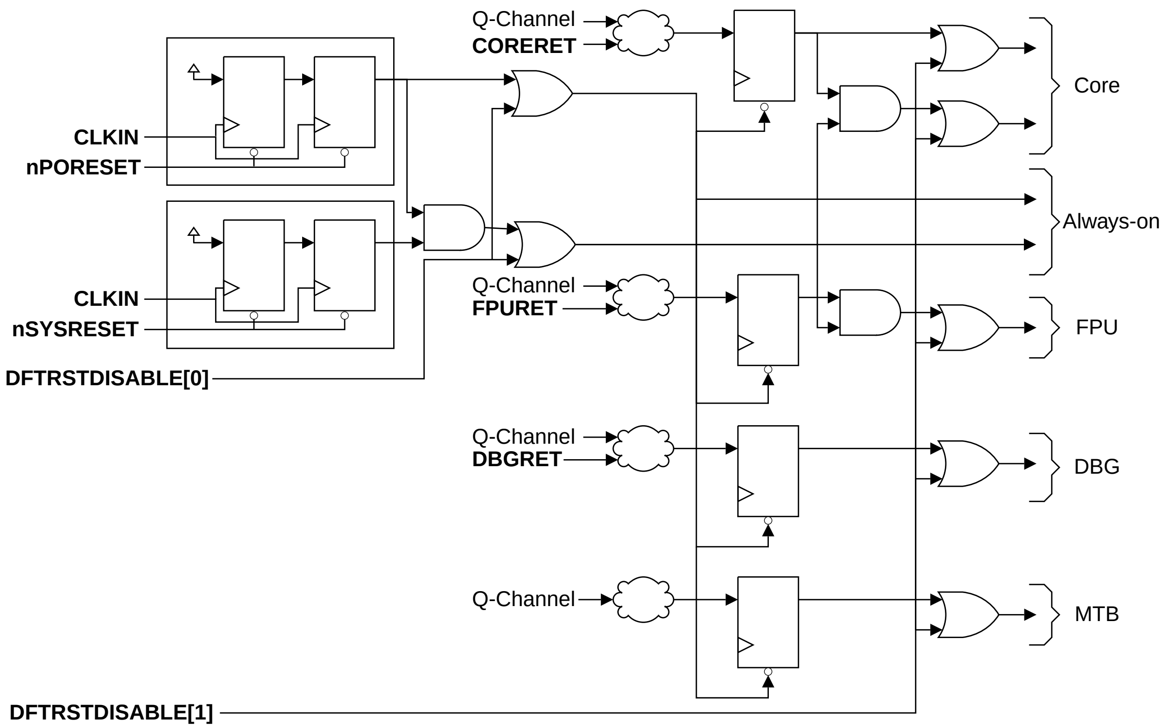


Figure 12-3 Cortex-M33 processor reset generation structure

Note

- The external resets are put through reset synchronizers to ensure synchronous deassertion.
- The system reset is combined with the power on reset to ensure it is asserted with power on reset.
- All reset signals can be forced HIGH (disabled) for DFT purposes. The two levels of reset have different override controls.

12.6.1 State retention support

The Cortex-M33 processor supports state retention on the Core, Debug, and FPU power domains. A power domain that supports state retention should only be reset when it has been powered off and not put into a retention state. The Cortex-M33 processor includes the input signals **CORERET**, **FPURET**, and **DBGRET** to support this requirement.

During powerdown, the domain reset is applied unless the associated **RET** signal input is asserted. The powerdown state for the domain ("retention" or "off") is remembered by the processor and is used for two purposes.

- To ensure reset is not asserted when powering up the domain from retention.
- To enforce correct dependencies between domains. For example, to prevent Core domain powerdown when FPU or Debug power domains are in retention.

Note

While this retention input can be statically assigned, it can also be driven from a register allowing the external system to choose the powerdown state in a more dynamic way.

12.7 Control sequences

There are specific control sequences to be used for the powering up and powering down of a domain from the system power controller.

12.7.1 Without state retention

The following figure shows the powerup and powerdown sequences expected for a domain without state retention. It assumes that powerdown requests are accepted by the processor. For completeness, the internally generated power on reset, **nPORESET**<PD>, and clock, **CLK**<PD> for the domain are also included in the figure.

When a controller wants to powerup a domain it applies power (T_u1), removes isolation (T_u2) and signals this to the processor by raising **QREQn** (T_u3). The processor then deasserts the reset (T_u4), and signals that the domain is fully up to the controller by raising **QACCEPTn** (T_u5). The domain clock is active while **QACCEPTn** is HIGH.

Powerdown is the opposite of powering up except the controller still initiates the power state change and the processor acknowledges when clock and resets have been dealt with.

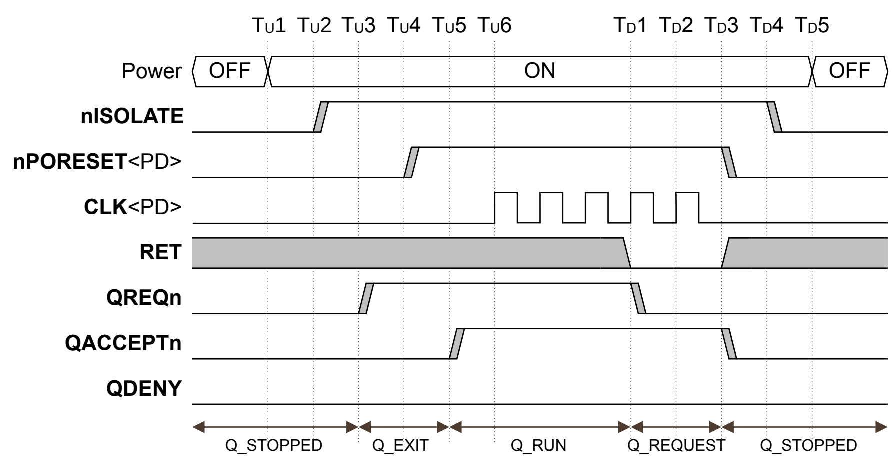


Figure 12-4 Power control sequence without retention

12.7.2 With state retention

The following figure shows the powerup and powerdown sequences expected for a domain with state retention. The figure assumes that powerdown requests are accepted by the processor and that the domain has previously been powered on and then powered down into retention. For completeness, the internally generated power on reset, **nPORESET**<PD> and clock, **CLK**<PD> for the domain are also included in the figure.

The initial powerup sequence is the same as that shown in *12.7.1 Without state retention* on page 12-168.

When a controller wants to powerup a domain that was powered down into retention, it applies power (T_u1), restores the flop values (T_u2), removes isolation (T_u3), and signals this to the processor by raising **QREQn** (T_u4). The processor then signals that the domain is fully up by raising **QACCEPTn** (T_u6). The domain clock is active while **QACCEPTn** is HIGH.

Powerdown is the exact reverse with the exception that the controller still initiates the power state change and the processor acknowledges once it has dealt with the clocks. The controller can then proceed to isolate the domain put the flops into retention and remove the power.

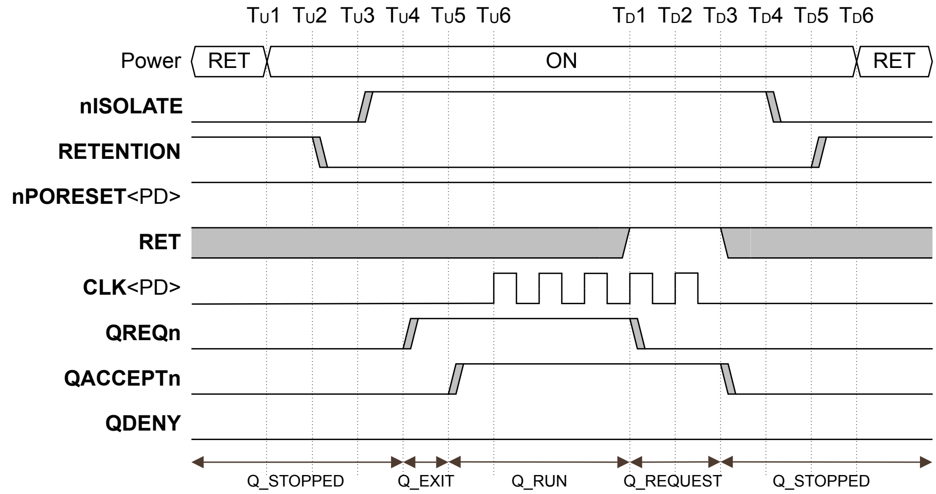


Figure 12-5 Power control sequence with retention

12.7.3 Software transparent retention

The following figure shows the power control sequence expected for a domain with transparent retention.

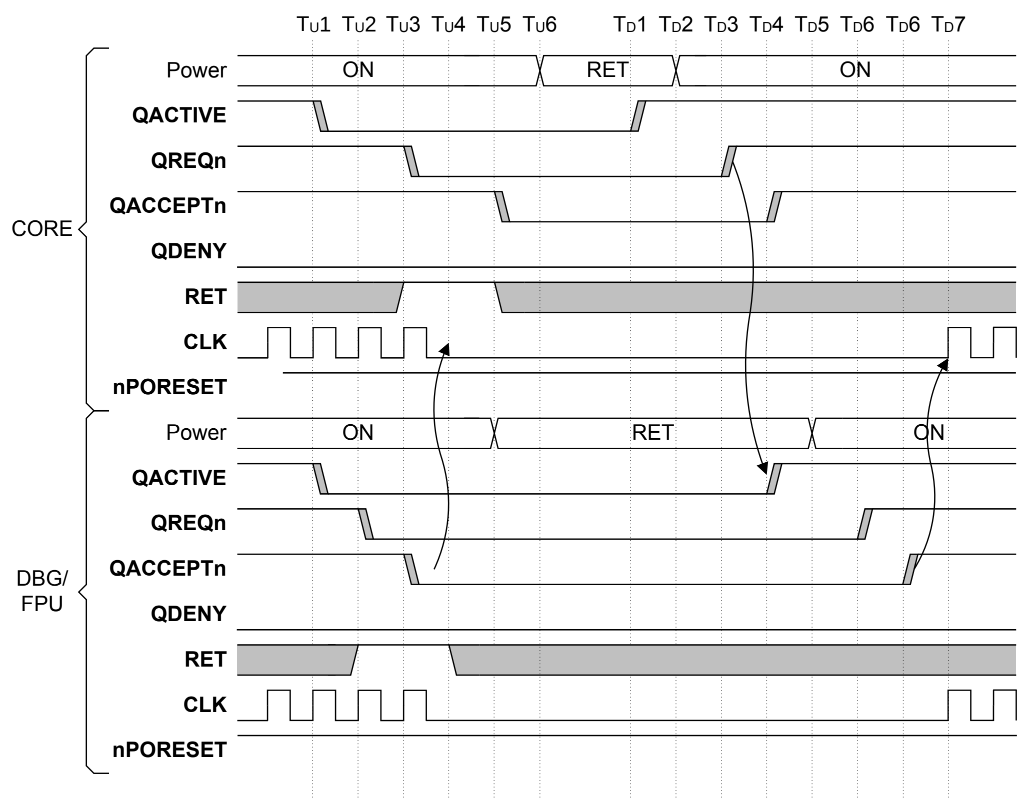


Figure 12-6 Power control sequence transparent retention

12.8 Power intent specification

The *Unified Power Format* (UPF) files are provided in the `logical/teal/power_intent/upf` directory.

It contains the following UPF files:

- `TEAL_constraints_unconfigured.upf`
- `TEAL_constraints_ret_unconfigured.upf`
- `TEAL_configuration_unconfigured.upf`
- `TEAL_configuration_ret_unconfigured.upf`
- `exclude.upf`.
- `testbench_unconfigured.upf`.
- `testbench_ret_unconfigured.upf`.

The UPF files must be rendered to generate files that are specific for your processor configuration. See *12.9 Rendering UPF constraints* on page 12-171 for more information.

You can use both your configured constraints UPF files together with the delivered RTL to validate the power intent specification of your SoC. This is done either by:

- Statically using rule checking tools.
- Dynamically using power-aware simulation tools. See *10.8.3 Running with UPF* on page 10-123 for information on how to run a power-aware simulation in the execution testbench. This uses the rendered `testbench.upf` file in directory `logical/teal/power_intent/upf` which is created when the UPF is rendered.

Note

The UPF files are IEEE 1801-2009 compliant.

Related concepts

10.8.3 Running with UPF on page 10-123.

12.9 Rendering UPF constraints

This section describes how to render the Cortex-M33 processor constraints UPF file using the `RenderTEAL_upf.pl` script:

1. Locate the UPF files and the `RenderTEAL_upf.pl` script in the directory `logical/teal/power_intent/upf`.
2. Render the UPF files using the `RenderTEAL_upf.pl` script with appropriate command-line options for your processor configuration. See *Table 12-10 Render script command-line options* on page 12-171.
3. Locate the generated UPF constraints file for your configuration in `logical/teal/power_intent/upf/TEAL_constraints.upf` and `TEAL_configuration.upf`.

Table 12-10 Render script command-line options

Option	Description
`-debug`	Configuration for PDDEBUG: **0** PDDEBUG not present. **1** PDDEBUG is present.
`-fpu`	Configuration for PDFPU: **0** PDFPU not present. **1** PDFPU is present.
`-mtb`	Configuration for PDMTB: **0** PDMTB not present. **1** PDMTB is present.
`-no_ret`	Configuration for retention supplies: **0** Retention supplies present. **1** Retention supplies absent.
`-ret_aon`	Configuration for retention supplies: **0** Retention supplies switchable. This is the default value. **1** Retention supplies always on. **Note** If you want to avoid extra power switches in the implementation flow, you should select retention supplies always on.
`-help`	Configuration help message.

12.10 Power domain clamping values

The `TEAL_constraints_unconfigured.upf` and `TEAL_constraints_ret_unconfigured.upf` files describe the output signals of the power domains that must be clamped HIGH. All outputs that are not clamped HIGH must be clamped LOW.

Chapter 13
DSM Generation

This chapter describes how to generate a *Design Simulation Model* (DSM).

It contains the following sections:

13.1 About DSM generation

A DSM is a two-state, obfuscated, cycle accurate, simulation model. A DSM is derived directly from the RTL and fully matches the cycle timing and behavior of the RTL. The programmers model signals are exported to a wrapper below the top-level but no other internal signals are visible within the model.

DSMs support the major Verilog and SystemVerilog simulators, Mentor QuestaSim, Synopsys VCS, and Cadence IUS. The DSM generation flow uses the Verilator open-source Verilog simulator that converts the Verilog RTL to C. A testbench and simulation script is also provided with a DSM so that you can test the installation of a model.

Simulation speed, when using a DSM with the SystemVerilog DPI interface, is six to seven times slower than the RTL. Using a DSM with the Verilog PLI interface is also slower. There is some variation in performance between the supported simulators.

You can optionally generate a DSM of your configured Cortex-M33 processor for in-house use or to provide to your customers.

A script is provided to allow you to generate and test your Cortex-M33 DSM. Generation of both 32-bit and 64-bit Linux DSMs is supported.

The DSM generation script uses the execution testbench to validate the model.

13.2 Prerequisites

The DSM generation flow requires the following prerequisites:

- A computer with at least 8GB of memory.

  Download and build the Verilator simulator, see the *ARM Cortex®-M33 MCU Release Note* for the Verilator version supported:

  ```
  $ cd <temporary directory>$ wget http://www.veripool.org/ftp/verilator-<version>.tgz$ tar
  zxf verilator-<version>.tgz$ cd verilator-<version>$ ./configure --prefix=$HOME$ make -
  j4$ make install
  ```

  This puts the required verilator binaries and included files into the following directories:

  ```
  ~/bin/
  ```

  and

  ```
  ~/share directories
  ```
- See the *ARM Cortex®-M33 MCU Release Note* for details of the GCC, Python and Perl versions that are required by the DSM generation flow.
- The DSM models have been tested with the simulator versions supported by the Cortex-M33 processor deliverables, as described in the *ARM Cortex®-M33 MCU Release Note*.

Related references

Appendix E Tarmac Tracing on page Appx-E-222.

13.3 Building and testing a DSM model

To generate, test, and package a DSM model of your Cortex-M33 processor configuration:

1. Configure your Cortex-M33 processor as described in *Chapter 2 Configuration Guidelines* on page 2-25.
2. Run the execution testbench test programs on the RTL and check they pass, as described in *Chapter 10 Execution Testbench* on page 10-104.
3. Build and test the DSM model using the generation script `BuildTEAL_DSM.pl`, for example:

   ```
   $ cd logical/teal/dsm
   ```

   ```
   $ ./BuildTEAL_DSM.pl -vendor=<your company name> -config=<name of your
   configuration>
   ```

   The command line options used in the example generate a DSM named `TEAL_DSM_<name of your configuration>`.

   It is tested with all six simulators and C-language interface combinations. The final model is then packaged into a tar file in the release directory.
4. Review the DSM deliverable readme file `TEAL_DSM_<name of your configuration>_README.txt` in the `dsm/logs/<config>_<32|64>bit_unlic` directory.
   Check that:
   - The configuration parameters are as expected.
   - All the required simulator and C-language interface combinations have been tested and all tests have passed.

Related references

Chapter 2 Configuration Guidelines on page 2-25.
Chapter 10 Execution Testbench on page 10-104.

13.4 DSM generation script command line options

The build options are controlled with the BuildTEAL_DSM.pl script command line options described in the following table. You must run the script without modification.

You must specify a vendor name and a configuration name, see the following table.

Table 13-1 DSM script command-line options

Command-line options	Description	Notes
vendor=<*your company name*>	Specifies a vendor name to generate a DSM model	-
config=<*config name*>	Specifies your configuration name, for example CONFIG1	A configuration name must be specified to generate a DSM model

13.4.1 Simulation options

The simulators that validate your model are shown in the following table.

Table 13-2 DSM simulation options

Command-line options	Description	Notes
-sim=mti	A mentor MTI simulator using Verilog PLI	• Simulations run faster using DPI. • If no -sim option is specified, all simulation options are tested. • A PLI model can be used with Verilog and SystemVerilog simulators. • DPI models can only be used with SystemVerilog simulators. • You must validate your DSM on all simulators that you want to support.
-sim=vcs	A synopsys VCS simulator using Verilog PLI	
-sim=ius	A cadence IUS simulator using Verilog PLI	
-sim=mti-sv	A mentor MTI simulator using SystemVerilog DPI	
-sim=vcs-sv	A synopsys VCS simulator using SystemVerilog DPI	
-sim=ius-sv	A cadence IUS simulator using SystemVerilog DPI	

13.4.2 Operating system architecture

There are two operating system architectures.

The following table describes the operating system architectures available.

Table 13-3 DSM operating system architectures

Command line options	Description	Notes
-osarch=64	64-bit Linux	• Default is 64-bit. • A 32-bit DSM can be used with a 32-bit simulator running on 64-bit Linux. • A 32-bit DSM cannot be used with a 64-bit simulator. • A 64-bit DSM can only be used with a 64-bit simulator running on 64-bit Linux. • A 32-bit DSM can be generated on 64-bit Linux. • 32-bit DSMs simulate faster than 64-bit DSMs.
-osarch=32	32-bit Linux	

13.5 Command-line examples

Command-line examples for use with 64-bit Linux and 32-bit Linux simulators.

64-bit Linux

`./BuildTEAL_DSM.pl -vendor=<your company name> -config=CONF1 –sim=mti-sv –sim=vcs-sv`.

Generates a DSM for your Cortex-M33 processor configuration that is specified in `logical/teal/verilog/TEAL_CONFIG.v` that supports the default OS architecture, 64-bit Linux.

It is tested with the `MTI` and `VCS` SystemVerilog simulators using the DPI interface.

32-bit Linux

`./BuildTEAL_DSM.pl -vendor=<your company name> -config=CONF2 –osarch=32`

Generates a DSM for your Cortex-M33 processor configuration that is specified in `logical/teal/verilog/TEAL_CONFIG.v` that supports the 32-bit Linux architecture.

It is tested with all six simulators and C-language interface combinations.

13.6 If DSM generation fails

Possible causes and solutions for the failure of DSM model generation are:

- Model not generated.

  If the message

  `-I- BuildTEAL_DSM.pl: DSM generation complete`

  is not displayed, the DSM model has not been generated.

  Possible solutions:
  — Check that the machine you are using has at least 8GB of memory.
  — If you are using a job submission system, increase the run time limits of the job.
- `config_check` test fails.

  Check that the processor configuration parameter settings in

  `logical/teal/verilog/TEAL_CONFIG.v`

  match the execution testbench configuration settings in

  `logical/testbench/execution_tb/tests/EXECTB_Config.h`
- Simulation fails:
  — If you are using a job submission system, check your job run time limits are sufficient.
  — Check the simulation and compile log files that are stored in `logical/teal/dsm/logs`
  — If the simulation fails to complete, view the log files for the last simulator option that is located in `logical/testbench/execution_tb/logs`
  — Check that all the execution testbench tests have passed on the RTL.

13.7 Generation directory structure

The DSM generation script and associated files are stored in the directory structure that is shown in the following figure.

The log files from the DSM generation processes are stored in the `logs` directory.

The final DSM deliverable files are in a tar file that is stored in the `release` directory.

The `BuildTEAL_DSM.pl` script creates the directory structure that is shown in the following figure.

```
└ teal/
   └ dsm/                              ——— Contains the DSM generation script
      ├ build/                         ——— Temporary files generated during the build
      ├ release/                       ——— Contains the final tar file
      ├ release_test/                  ——— Extracted tar files, used for testing
      └ logs/
         └ <config>_<32|64>bit_unlic/  ——— DSM generation log files
            └ sim_logs/                ——— Separate sub-directories for mti, vcs, ius, and vector replay
```

Figure 13-1 Generation directory structure

13.8 Deliverable directory structure

The `BuildTEAL_DSM.pl` script packages the DSM into a tar file ready for release that has the structure that is shown in the following figure.

```
teal
 └simulation_models/
   └linux_<32|64>bit_unlic/
     └TEAL_DSM_<RTL revision>/
       ├dsm/
       │ ├TEAL_DSM.v
       │ ├TEAL_DSM.a
       │ ├TEAL_DSM.so
       │ ├TEAL_DSM.sv
       │ ├TEAL_DSM.tab
       │ ├TEAL_DSM_<config>.so
       │ ├teal_tarmac_decode
       │ ├teal_tarmac.sv
       │ ├teal_tarmac_capture.sv
       │ └teal_tarmac_dpi.so
       ├testing_<config>/
       │ ├TEAL_DSM_TESTBENCH.sh
       │ └TEAL_DSM_TESTBENCH.v
       └docs/
         ├TEAL_DSM_<config>_README.txt
         ├tarmac-format.txt
         └DUI0302C_design_simulation_model_ug.pdf
```

Figure 13-2 Deliverable directory structure

Appendix A
GPIO Programmers Model

This appendix describes the GPIO programmers model.

It contains the following section:

A.1 About the GPIO programmers model

The GPIO is a general-purpose I/O device.

It has the following properties:

- Three registers.
- 32 input or output lines with programmable direction.
- Word and halfword read and write access.
- Address-masked byte write to facilitate quick bit set and clear operations.
- Address-masked byte read to facilitate quick bit test operations.
- Maskable interrupt generation that is based on input value change.

This section describes the GPIO programmers model. It contains the following sections:

- *A.1.1 Data Register - GPIODATA* on page Appx-A-183.
- *A.1.2 Direction Register - GPIODIR* on page Appx-A-184.
- *A.1.3 Interrupt Enable Register - GPIOIE* on page Appx-A-185.

The following table lists the GPIO registers.

Table A-1 GPIO registers

Address offset	Name	Type	Description
`0x00000000-0x000003FF`	**GPIODATA**	Read/Write	Reads current value of **GPIOIN** pins, or sets value that is driven onto **GPIOUT** pins.
`0x00000400`	**GPIODIR**	Read/Write	Configures the direction of the I/O. The value is driven on **GPIOEN.** Setting a bit defines that bit as an output.
`0x00000410`	**GPIOIE**	Read/Write	Configures the interrupt on input change feature.

See *Table 10-19 GPIO 1 bit assignments* on page 10-139 for details of the connections to **GPIOIN** and **GPIOOUT**.

A.1.1 Data Register - GPIODATA

Use this register to read the value of the **GPIOIN** pins when the corresponding **GPIODIR** is `0`.

Use this register to drive the value onto the **GPIOOUT** pins when the corresponding **GPIODIR** is `1`.

Note

Reading **GPIODATA** when **GPIOEN** is `1` returns the value that is seen on the **GPIOIN** pin.

The register address, access type, and reset state are:

Address
: `GPIO_BASE to GPIO_BASE + 0x000003FF`.

Access
: Read/write.

Reset state
: `0x00000000`.

Byte accesses to the Data Register use the bits **HADDRS[9:2]** as a mask. This enables you to perform various bit set and bit clear operations efficiently because it avoids the requirement for a read-modify-write to the **GPIODATA** register.

Bit set example

To set bit[9] of GPIO 0 Data Register, perform a byte write access to address `0x40000009` with **HWDATAS** set to `0x200` and **HSIZES** set to `0b000`.

The mask that is extracted from the address is `0x02` and applied to the second byte of the GPIO 0 Data Register.

This sets bit[9] but preserves all other bits.

Bit clear example

To clear bit[9] of GPIO 0 Data Register, perform a byte write access to address `0x40000009` with **HWDATAS** set to `0x0` and **HSIZES** set to `0b000`.

The mask that is extracted from the address is `0x02` and applied to the second byte of the GPIO 0 Data Register.

This clears bit[9] but preserves all other bits.

Bit read example

To read bit[9] of GPIO 0 Data Register, perform a byte read access to address `0x40000009` with **HSIZES** set to `0b000`.

The mask that is extracted from the address is `0x02` and applied to the second byte of the GPIO 0 Data register.

HRDATAS contains the value of bit[9], all other bits are zeros.

The following figure shows the structure of the GPIO Data Register.

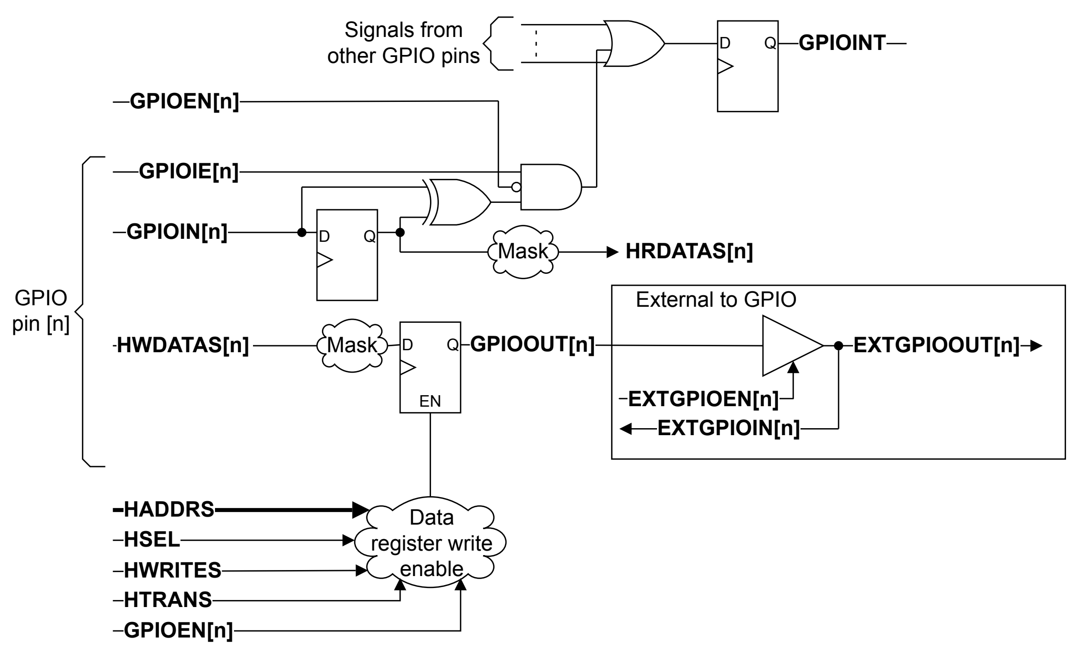


Figure A-1 GPIO Data Register

The external signals **EXTGPIOOUT**, **EXTGPIOIN**, and **EXTGPIOEN** map to **GPIOOUT**, **GPIOIN**, and **GPIOEN** respectively.

You can use the **GPIOINT** output pin of the GPIO as an interrupt source.

A.1.2 Direction Register - GPIODIR

Use this register to configure the direction of the **Data Register** as either `input` or `output`. This connects to the **GPIOEN** pin and is used as a tristate enable. Set bits in this register to `0b1` when you want to use the bit as an output.

The register address, access type, and reset state are:

Address
`GPIO_BASE + 0x00000400`.
Access
Read/write.
Reset state
`0x00000000`.

A.1.3 Interrupt Enable Register - GPIOIE

Use this register to enable input signal changes on **GPIOIN** to trigger an interrupt through the **GPIOINT** output.

You can set **GPIOIE[n]** to enable changes on **GPIOIN[n]** to pulse the **GPIOINT** pin.

The register address, access type, and reset state are:

Address
`GPIO_BASE + 0x00000410`.
Access
Read/write.
Reset state
`0x00000000.`

Appendix B
CoreSight SoC

This appendix describes the location of the CoreSight SoC-400 configuration files for the Cortex-M33 processor, and describes the CoreSight SoC-400 trace comparison tools.

It contains the following sections:

B.1 Location of the Cortex®-M33 processor CoreSight SoC-400 support files

The Cortex-M33 processor deliverables provide a header file for use with the CoreSight SoC-400 product, as described in the following table.

Table B-1 Location of CoreSight SoC-400 header file

Filename	Description	Location
`cssoc_teal.h`	Header file containing the description of the Cortex-M33 processor CoreSight components and associated function calls to support the integration of a Cortex-M33 processor in a CoreSight system.	`logcial/testbench/integration_cssoc`

In addition, a provided trace comparison tool compares the trace output with a trace reference file. The location of this tool and the reference files are listed in the following table.

Table B-2 Location of ETM trace comparison tool

Filename	Description	Location
`teal_ik_etm_compare.pl`	Trace comparison tool to compare a trace stream from the Cortex-M33 processor with a trace reference trace.	`logical/testbench/shared/tools/bin`
`cssoc_teal_trace_i.ref`[f]	A CoreSight SoC-400 Cortex-M33 trace reference file for the ETM. The configuration parameter `ETM` is set to `1`.	`logical/testbench/integration_cssoc`
`teal_ik_itm_compare.pl`	Trace comparison tool for the ITM trace stream from the Cortex-M33 processor.	`logical/testbench/shared/tools/bin`
`cssoc_teal_itm_trace.ref`	A CoreSight SoC-400 Teal trace reference file for the ITM. The configuration parameter ITM is set to 1.	`logical/testbench/integration_cssoc`

Related concepts

2.2 Configuration options on page 2-27.

[f] Use this reference file with the `teal_ik_compare` tool to check for correct instruction generation in your system when using the CoreSight SoC trace test.

B.2 Comparing captured ETM trace to the trace reference file

The ETM Trace log file generated by the CoreSight SoC-400 tests must be post-processed to verify the data captured at each trace sink matches the reference data the trace source is expected to generate.

A trace comparison tool, `teal_ik_etm_compare.pl`, is provided to check your ETM trace log file. The comparison tool:

- Reads in the ATB trace log file, that contains ETMv4 trace data that has been captured in a trace sink, such as a TPIU.
- Decodes the trace to basic ETMv4 trace elements such as atoms and addresses.
- Checks it is ETMv4 compliant.
- Compares this with a reference file if supplied.

The trace comparison flow supports verification of a predefined sequence of program instructions. For information about the instruction sequence and reference files, see *B.1 Location of the Cortex®-M33 processor CoreSight SoC-400 support files* on page Appx-B-187.

See the *ARM CoreSight SoC-400 User Guide* for information of the `trace` test and trace post-processing, including extra tools to capture and extract ATB logs from your simulation.

To compare captured trace to a trace reference file, run the command:

```
teal_ik_etm_compare.pl -atb <log_file> -atid <id> -data_trace -ref <ref file> -output
<output file>
```

Where:

- `<log_file>` is the ATB log file.
- `<id>` is the trace id of the ETM trace stream.
- `<ref file>` is the file name of the golden reference file.
- `<output file>` is the file name for the decoded trace items output.

To list all the available options, run the command:

```
teal_ik_etm_compare.pl -h
```

For example, to perform ETM trace comparison run the command:

```
teal_ik_etm_compare.pl -atb log_ID2.atb -atid 2 -ref cssoc_teal_trace_i.ref
```

The following example shows the output from `teal_ik_etm_compare.pl` for instruction trace.

Output from teal_ik_etm_compare.pl, instruction trace

```
teal_ik_etm_compare.pl
------------------
Opening atb file : log_ID4.atb
Opening ref file : cssoc_teal_trace_i.ref
Output           : log.out
atb_logger1 :    0 :        0ns (0x00000000): End of file : 9
reference   :    0 :        0ns (0x00000000): End of file : 10
```

B.3 Comparing captured ITM trace to the trace reference file

The ITM trace log file generated by the CoreSight SoC-400 tests must be post-processed to verify the data captured at each trace sink matches the reference data the trace source is expected to generate.

A trace comparison tool, `teal_ik_itm_compare.pl`, is provided to check your ITM trace log file. The comparison tool:

- Reads in the ATB trace log file, that contains ITM trace data that has been captured in a trace sink, such as a TPIU.
- Decodes the trace to basic ITM trace elements.
- Checks it is ITM compliant.
- Compares this with a reference file if supplied.

The trace comparison flow supports verification of a predefined sequence of program instructions. For information about the instruction sequence and reference files, see *B.1 Location of the Cortex®-M33 processor CoreSight SoC-400 support files* on page Appx-B-187.

See the *ARM CoreSight SoC-400 User Guide* for information of the `trace` test and trace post-processing, including extra tools to capture and extract ATB logs from your simulation.

To compare captured trace to a trace reference file, run the command:

`teal_ik_itm_compare.pl -atb <log_file> -atid <id> -data_trace -ref <ref file> -output <output file>`

Where:

- `<log_file>` is the ATB log file.
- `<id>` is the trace id of the ITM trace stream.
- `<ref file>` is the file name of the golden reference file.
- `<output file>` is the file name for the decoded trace items output.

The following example shows the output from `teal_ik_itm_compare.pl` for instruction trace.

Output from teal_ik_itm_compare.pl, instruction trace

```
teal_ik_itm_compare.pl
----------------------
rm: cannot remove `diff_itm.log': No such file or directory
Opening atb file : log_ID2.atb
Extract ATID     : 2
Opening ref file : cssoc_teal_itm_trace.ref

              ***TEST PASSED OK***
```

Appendix C
DAP and TPIU signals and implementation constraints

This appendix describes the DAP and TPIU signals and implementation constraints.

It contains the following sections:

C.1 Debug Access Port signals

Descriptions of the *Debug Access Port* (DAP) signals and how you must connect the signals in your SoC design.

Table C-1 TEALDAP signals

Name	Type	Clock domain	Description	Connection Information
SWCLKTCK	Input	-	SW/JTAG clock.	**SWCLKTCK** is typically driven by an external debugger and is completely asynchronous to the other clocks in the system.
DPRESETn	Input	**SWCLKTCK**	DP synchronous reset active LOW.	-
DCLK	Input	-	AP clock.	**DCLK** must always be driven while a debugger is connected. It can be gated when no debugger is connected. You can use the **CDBGPWRUPACK** input to the TEALMCU level to detect the presence of an external debugger requesting a connection.
APRESETn	Input	**DCLK**	AP synchronous reset active LOW.	-
nTRST	Input	**SWCLKTCK**	JTAG test logic reset signal.	**nTRST** can be tied HIGH when a synchronous JTAG reset is provided through the TMS pin. If implemented, **nTRST** must not be synchronized to **SWCLKTCK**. Tie **nTRST** LOW if your implementation does not contain JTAG-DP.
TDI	Input	**SWCLKTCK**	JTAG data in.	If your implementation contains a JTAG-DP, connect to input pad for **TDI**.
TDO	Output	**SWCLKTCK**	JTAG data out.	Connect to output pad for **TDO**. Optionally, you can use a tristate pad.
nTDOEN	Output	**SWCLKTCK**	JTAG **TDO** output enable.	Connect to optional tristate pad for **TDO**.
SWDITMS	Input	**SWCLKTCK**	SW data input and JTAG TMS.	If your implementation contains both a JTAG-DP and SW-DP, then, connect to the tristate pad for **SWDITMS**. If your implementation contains only an SW-DP, then connect to the tristate pad for **SWDO**. If your implementation contains only a JTAG-DP, then connect to input pad for **TMS**.
SWDO	Output	**SWCLKTCK**	SWdata output.	If your implementation contains both JTAG-DP and SW-DP, then connect to tristate pad for **SWDITMS**. If your implementation contains only an SW-DP, then connect to the tristate pad for **SWDO**.
SWDOEN	Output	**SWCLKTCK**	SW data output enable.	
SWDETECT	Output	**SWCLKTCK**	SW line reset detect.	Tie HIGH for one cycle of **SWCLKTCK** if your implementation contains an SW-DP and a Serial Wire line reset sequence is attempted while the TEALDAP is not in Dormant state. Optionally, connect to your own logic. For example, to disable a JTAG test device when a Serial Wire TEALDAP is implemented without multi-drop.

Table C-1 TEALDAP signals (continued)

Name	Type	Clock domain	Description	Connection Information
SWSEL	Output	**SWCLKTCK**	SW protocol active signal.	**SWSEL** and **JTAGSEL** can be combined to indicate whether JTAG, SW, or Dormant mode is active. • If **SWSEL** and **JTAGSEL** are LOW, then Dormant mode is active. • If **SWSEL** is LOW and **JTAGSEL** is HIGH, then JTAG mode is active. • **SWSEL** is HIGH and **JTAGSEL** is LOW, then SW mode is active. This information can be used to: • Disable other TAP controllers when not in JTAG mode. • Use spare pins when not in JTAG mode.
JTAGSEL	Output	**SWCLKTCK**	JTAG protocol active signal.	
HALTED	Input	**DCLK**	Processor halted.	Connect to the **HALTED** output of your Cortex-M33 processor.
CDBGPWRUPREQ	Output	None	Debug powerup request.	Connect to your power management unit.
CDBGPWRUPACK	Input	None	Debug powerup acknowledge.	
DEVICEEN	Input	**DCLK**	Debug enabled by system.	Tie this signal HIGH not to use debug authentication to authenticate debugger access to devices on the SLV bus. Tie this signal LOW to permanently disable debugger access. Connect this signal to your own logic, such as that connected to the Cortex-M33 processor debug enables, to dynamically enable and disable debugger access.
SLVADDR[31:0]	Output	**DCLK**	AHB address.	Connect to the **HADDRD** signal of your Cortex-M33 processor.
SLVWDATA[31:0]	Output	**DCLK**	AHB write data.	Connect to the **HWDATAD** signal of your Cortex-M33 processor.
SLVTRANS[1:0]	Output	**DCLK**	AHB transfer valid.	Connect to the **HTRANSD** signal of your Cortex-M33 processor.
SLVPROT[6:0]	Output	**DCLK**	AHB transaction protection.	Connect to the **HPROTD** signal of your Cortex-M33 processor.
SLVWRITE	Output	**DCLK**	AHB write/not read.	Connect to the **HWRITED** signal of your Cortex-M33 processor.
SLVSIZE[1:0]	Output	**DCLK**	AHB access size.	Connect to the **HSIZED** signal of your Cortex-M33 processor.
SLVNONSEC	Output	**DCLK**	AHB transaction security.	Connect to the **HNONSECD** signal of your Cortex-M33 processor.
SLVRDATA[31:0]	Input	**DCLK**	AHB read data.	Connect to the **HRDATAD** signal of your Cortex-M33 processor.
SLVREADY	Input	**DCLK**	AHB ready.	Connect to the **HREADYD** signal of the Cortex-M33 processor.
SLVRESP	Input	**DCLK**	AHB response.	Connect to the **HRESPD** signal of your Cortex-M33 processor.

Table C-1 TEALDAP signals (continued)

Name	Type	Clock domain	Description	Connection Information
BASEADDR[31:0]	Input	**DCLK**	AP ROM table base.	If you have an MCU level ROM table, tie to the base address of your MCU level ROM table. Otherwise, tie to the base address of the top level ROM table.
TARGETID[31:0]	Input	None	Target ID for SW multidrop selection.[g]	Tie to the value of your TARGETID register. For more information, see *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2*.
INSTANCEID[3:0]	Input	None	**DLPIDR[31:28]** for SW multidrop.[g]	If your implementation contains SW-DP, this signal configures the Target Instance field in the Data Link Protocol Identification Register. You must tie this to a value chosen to ensure that all Serial Wire multi-drop devices connected to the same interface are uniquely identifiable. For more information on *Target Selection Register* (TARGETSEL), see *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2*. If your implementation does not contain SW-DP, tie this input to `0b0000`.
ECOREVNUM[7:0]	Input	None	Engineering change order revision numbering: **[7:4]** DP Revision.[g] **[3:0]** AP Revision.[g]	Tie all bits LOW. This signal must be brought up to the top level on your SoC design to prevent synthesis tools optimizing out the logic this signal drives. ARM provides instructions on how to tie this signal in the event of an ECO change.

[g] This is a static signal that must not be changed after reset.

C.2 DAP implementation constraints

The following information describes the DAP implementation constraints that ARM recommends during integration.

The DAP uses CDC techniques that allow buses to cross the clock domains without synchronizers. The launch flops for these buses are guaranteed to be stable for two capture clock cycles at the point they are sampled in the capture clock domain. Therefore, ARM recommends a maximum delay of at most two cycles of the capture clock on paths across the clock domain from the following registers:

- `u_dap_top.u_dap_dp.u_dap_dp_cdc.gen_reg_dp_data*.u_reg_dp_data.iregdo`
- `u_dap_top.u_dap_dp.u_dap_dp_cdc.gen_reg_dp_regaddr*.u_reg_dp_regaddr.iregdo`
- `u_dap_top.u_dap_dp.u_dap_dp_cdc.u_reg_dp_rnw.iregdo`
- `u_dap_top.u_dap_ap.u_dap_ap_cdc.gen_reg_ap_data*.u_reg_dp_data.iregdo`
- `u_dap_top.u_dap_ap.u_dap_ap_cdc.u_reg_ap_err.iregdo`

For all other paths crossing the clock domain, ARM recommends a maximum delay of at most two cycles of the fastest clock.

C.3 Trace Port Interface Unit signals

Descriptions of the *Trace Port Interface Unit* (TPIU) signals and how you must connect the signals in your SoC design.

Table C-2 Trace Port Interface Unit signals

Name	Type	Clock domain	Description	Connection information
ATCLK	Input	ATCLK	ATB and APB clock.	-
ATCLKEN	Input	ATCLK	ATB and APB clock enable.	
TRACECLKIN	Input	TRACECLKIN	TRACECLKIN clock.	
RESETn	Input	ATCLK	Active LOW reset.	
TRESETn	Input	TRACECLKIN	TRACECLKIN asynchronous reset.	
PWRITE	Input	ATCLK	Direction.	Connect to the EPPB interface of the processor.
PENABLE	Input	ATCLK	Enable.	
PSEL	Input	ATCLK	Select.	
PADDR[11:2]	Input	ATCLK	Address.	
PWDATA[12:0]	Input	ATCLK	Write data.	
PRDATA[31:0]	Output	ATCLK	Read Data.	
ATDATA1S[7:0]	Input	ATCLK	ATB Interface 1 ATB data.	Connect to the ETM or ITM interface of the processor.
ATID1S[6:0]	Input	ATCLK	ATB Interface 1 ID for TPIU.	
ATREADY1S	Output	ATCLK	ATB Interface 1 ATB ready.	
ATVALID1S	Input	ATCLK	ATB Interface 1 ATB valid.	
AFREADY1S	Input	ATCLK	ATB Interface 1 ATB flush.	
AFVALID1S	Output	ATCLK	ATB Interface 1 ATB valid.	
ATDATA2S[7:0]	Input	ATCLK	ATB Interface 2 ATB data.	
ATID2S[6:0]	Input	ATCLK	ATB Interface 2 ID for TPIU	
ATREADY2S	Output	ATCLK	ATB Interface 2 ATB ready.	
ATVALID2S	Input	ATCLK	ATB Interface 2 ATB valid.	
AFREADY2S	Input	ATCLK	ATB Interface 2 ATB flush.	
AFVALID2S	Output	ATCLK	ATB Interface 2 ATB valid.	
SYNCREQ1S	Output	ATCLK	ATB Interface 1 synchronization request.	
SYNCREQ2S	Output	ATCLK	ATB Interface 2 synchronization request.	
TRACECLK	Output	TRACECLKIN	Exported trace port clock.	Connect to a trace port analyzer.
TRACEDATA[3:0]	Output	TRACECLKIN	Trace port data.	
TRACESWO	Output	TRACECLKIN	Serial Wire Viewer data	
SWOACTIVE	Output	ATCLK	SWO mode selected.	Use to multiplex the Serial Wire Viewer data, **TRACESWO**.
TPIUACTV	Output	ATCLK	TPIU data active.	Connect to the **TPIUACTV** pin of the processor.

Table C-2 Trace Port Interface Unit signals (continued)

Name	Type	Clock domain	Description	Connection information
TPIUBAUD	Output	ATCLK	Unsynchronized TPIU baud indicator	Connect to the **TPIUBAUD** pin of the processor.
DSYNC	Input	ATCLK	DWT synchronization request.	Connect to the **DSYNC** pin of the processor.
ECOREVNUM[3:0]	Input	ATCLK	ECO revision number.	Tie all bits LOW. This signal must be brought up to the top level on your SoC design to prevent synthesis tools optimizing out the logic this signal drives. ARM provides instructions on how to tie this signal in the event of an ECO change.
ETMTRIGOUT	Input	ATCLK	ETM Trigger event output bit[0]. Indicates a trigger packet in the trace stream.	Connect to the **ETMTRIGOUT** pin of the processor.
MAXPORTSIZE[1:0]	Input	ATCLK	Indicates the number of pins available for the TracePort mode.	Tie off to indicate the size of the trace port available on the device.
TRACEPORTSIZE	Output	ATCLK	Indicates the current parallel trace port size: `0b00` 1-bit `0b00` 2-bit `0b00` 4-bit **Note** Only valid when **SWOACTIVE** is deasserted LOW.	Use in combination with **SWOACTIVE** to multiplex the **TRACEDATA** trace data port.

C.4 TPIU implementation constraints

The following information describes the TPIU implementation constraints that ARM recommends during integration.

The TPIU contains an asynchronous FIFO to reduce the probability of metastability issues. ARM recommends a maximum delay of one cycle of the fastest clock on paths across the clock domain from the following registers:

- `u_tpiu_top.u_tpiu_atb_fifo1.write_pointer_gray_async`
- `u_tpiu_top.gen_atb_fifo2.u_tpiu_atb_fifo2.write_pointer_gray_async`
- `u_tpiu_top.u_tpiu_trace_fifo1.read_pointer_gray_async`
- `u_tpiu_top.gen_trace_fifo2.u_tpiu_trace_fifo2.write_pointer_gray_async`

The TPIU uses CDC techniques that allow buses to cross the clock domains without synchronizers. The launch flops for these buses are guaranteed to be stable for two capture clock cycles at the point they are sampled in the capture clock domain. Therefore, ARM recommends a maximum delay of at most two cycles of the capture clock on paths across the clock domain from the following registers:

- `u_tpiu_top.u_tpiu_atb_fifo1.fifo_data*`
- `u_tpiu_top.gen_atb_fifo2.u_tpiu_atb_fifo2.fifo_data*`
- `u_tpiu_top.u_tpiu_atb_apb_if.reg_trans_wdata`
- `u_tpiu_top.u_tpiu_atb_apb_if.apb_addr_reg`
- `u_tpiu_top.u_tpiu_atb_apb_if.apb_read_reg`
- `u_tpiu_top.u_tpiu_trace_apb_if.reg_trans_rdata`

For all other paths crossing the clock domain, ARM recommends a maximum delay of at most two cycles of the fastest clock.

Appendix D
Signal Timing Constraints

This appendix describes the timing constraints on the processor signals.

It contains the following sections:

D.1 About signal timing constraints

The timing constraints for signals are classified according to the percentage of the **CLKIN** clock period that is available for external logic:

- The signal direction is defined from the point of view of the processor.
- For inputs, this is the delay between the last register and the input port.
- For outputs, this is the delay between the output port and the first register.

——— **Note** ———

Some of the input signals to the processor:

- Require multicycle constraints.
- Can be asynchronous to **CLKIN** and are synchronized inside the processor.

———

D.2 Clock and reset signal timing constraints

The following table shows the clock and reset signal timing constraints.

Table D-1 Clock and reset signal timing constraints

Name	Timing constraint	Direction
CORECLKEN	60%	Output
SSTCLKEN	60%	Input
NSSTCLKEN	60%	Input
nPORESET	30%[h]	Input
nSYSRESET	30%[h]	Input

[h] The reset signals are asynchronous to **CLKIN** and synchronized inside the Cortex-M33 processor.

D.3 Configuration and initialization signal timing constraints

The following table shows the configuration and initialization signal timing constraints.

Table D-2 Configuration and initialization signal timing constraints

Name	Timing constraint	Direction
CFGSSTCALIB	10%	Input
CFGNSSTCALIB	10%	Input
CFGBIGEND	10%[i]	Input
CFGFPU	10%[i]	Input
CFGDSP	10%[i]	Input
CFGSECEXT	10%[i]	Input
MPUNSDISABLE	10%	Input
MPUSDISABLE	10%	Input
SAUDISABLE	10%	Input
INITSVTOR	10%	Input
INITNSVTOR	10%	Input

[i] Configuration signals require a multicycle constraint of a three cycle setup and two cycle hold.

D.4 C-AHB interface timing constraints

The following table shows the C-AHB interface timing constraints.

Table D-3 C-AHB interface timing constraints

Name	Timing constraint	Direction
HTRANSC[1:0]	50%	Output
HBURSTC[2:0]	60%	Output
HADDRC[31:0]	60%	Output
HWRITEC	60%	Output
HSIZEC[2:0]	60%	Output
HWDATAC[31:0]	60%	Output
HPROTC[6:0]	60%	Output
HNONSECC	40%	Output
HREADYC	60%	Input
HRDATAC[31:0]	60%	Input
HRESPC	60%	Input
HMASTERC	60%	Output
HEXCLC	60%	Output
HEXOKAYC	60%	Input
HHINTC[2:0]	60%	Output
HINNERC[4:0]	60%	Output

D.5 S-AHB interface timing constraints

The following table shows the S-AHB interface timing constraints.

Table D-4 S-AHB interface timing constraints

Name	Timing constraint	Direction
HTRANSS[1:0]	50%	Output
HBURSTS[2:0]	60%	Output
HADDRS[31:0]	60%	Output
HWRITES	60%	Output
HSIZES[2:0]	60%	Output
HWDATAS[31:0]	60%	Output
HPROTS[6:0]	60%	Output
HNONSECS	40%	Output
HREADYS	60%	Input
HRDATAS[31:0]	60%	Input
HRESPS	60%	Input
HMASTERS	60%	Output
HEXCLS	60%	Output
HEXOKAYS	60%	Input
HHINTS[2:0]	60%	Output
HINNERS[4:0]	60%	Output

D.6 D-AHB interface timing constraints

The following table shows the D-AHB interface timing constraints.

Table D-5 D-AHB interface timing constraints

Name	Timing constraint	Direction
HTRANSD[1:0]	60%	Input
HBURSTD[2:0]	60%	Input
HADDRD[31:0]	60%	Input
HWRITED	60%	Input
HSIZED[2:0]	60%	Input
HWDATAD[31:0]	60%	Input
HPROTD[6:0]	60%	Input
HNONSECD	60%	Input
HREADYD	60%	Output
HRDATAD[31:0]	60%	Output
HRESPD	60%	Output

D.7 Instruction execution control and hint signal timing constraints

The following table shows the instruction execution control and hint signal timing constraints.

Table D-6 Instruction execution control and hints

Signal name	Timing constraint	Direction
CPUWAIT	70%	Input
CODEHINT[2:0]	20%	Output
IFLUSH	20%	Input
CURRNS	50%	Output

D.8 External peripheral interface EPPB timing constraints

The following table shows the External peripheral interface EPPB timing constraints.

Table D-7 External peripheral interface EPPB

Name	Timing constraint	Direction
PSEL	60%	Output
PENABLE	60%	Output
PPROT[2:0]	60%	Output
PWRITE	60%	Output
PADDR[19:2]	60%	Output
PWDATA[31:0]	60%	Output
PSTRB[3:0]	60%	Output
PREADY	60%	Input
PSLVERR	60%	Input
PRDATA[31:0]	60%	Input
PADDR31	60%	Output

D.9 Coprocessor interface timing constraints

The following table shows the coprocessor interface timing constraints.

Table D-8 Coprocessor interface timing constraints

Name	Timing constraint	Direction
CPENABLED[7:0]	80%	Output
CPPWRSU[7:0]	80%	Output
CPSPRESENT[7:0]	10%[j]	Input
CPNSPRESENT[7:0]	10%[j]	Input
CPNUM[2:0]	70%	Output
CPCDP	70%	Output
CPMCR	70%	Output
CPMRC	70%	Output
CPSIZE	70%	Output
CPREGS[11:0]	70%	Output
CPOPC[8:0]	70%	Output
CPPRIV	70%	Output
CPNSATTR	70%	Output
CPVALID	50%	Output
CPREADY	60%	Input
CPERROR	60%	Input
CPWDATA[63:0]	60%	Output
CPRDATA[63:0]	60%	Input

[j] Requires a multicycle constraint of a three cycle setup and a two cycle hold.

D.10 Debug signals timing constraints

The following table shows the debug signals timing constraints.

Table D-9 Debug signals timing constraints

Name	Timing constraint	Direction
DBGEN	50%	Input
SPIDEN	30%	Input
NIDEN	30%	Input
SPNIDEN	30%	Input
EDBGRQ	50%	Input
HALTED	50%	Output
DBGRESTART	50%	Input
DBGRESTARTED	50%	Output

D.11 Q-Channel interface timing constraints

The following table shows the Q-Channel interface timing constraints.

Table D-10 Q-Channel interface timing constraints

Name	Timing constraint	Direction
COREQREQn	10%[k]	Input
COREQACCEPTn	90%	Output
COREQDENY	90%	Output
COREQACTIVE	90%	Output
FPUQREQn	10%[k]	Input
FPUQACCEPTn	90%	Output
FPUQDENY	90%	Output
FPUQACTIVE	90%	Output
DBGQREQn	10%[k]	Input
DBGQACCEPTn	90%	Output
DBGQDENY	90%	Output
DBGQACTIVE	90%	Output
MTBQREQn	10%[k]	Input
MTBQACCEPTn	90%	Output
MTBQDENY	90%	Output
MTBQACTIVE	90%	Output
CORERET	70%	Input
FPURET	70%	Input
DBGRET	70%	Input

[k] The QREQn input signals are asynchronous to **CLKIN** and synchronized inside the Cortex-M33 processor.

D.12 Sleep control compatibility interface timing constraints

The following table shows the sleep control compatibility interface timing constraints.

Table D-11 Compatibility interface timing constraints

Name	Timing constraint	Direction
SLEEPING	70%	Output
SLEEPDEEP	70%	Output
SLEEPHOLDREQn	50%	Input
SLEEPHOLDACKn	50%	Output
WAKEUP	50%	Output

D.13 ITM interface timing constraints

The following table shows the ITM interface timing constraints.

Table D-12 ITM interface timing constraints

Name	Timing constraint	Direction
ATVALIDI	70%	Output
ATIDI[6:0]	70%	Output
ATDATAI[7:0]	70%	Output
AFREADYI	70%	Output
AFVALIDI	70%	Input
ATREADYI	70%	Input

D.14 ETM interface timing constraints

The following table shows the ETM interface timing constraints.

Table D-13 ETM interface timing constraints

Name	Timing constraint	Direction
ATVALIDE	70%	Output
ATIDE[6:0]	70%	Output
ATDATAE[7:0]	70%	Output
AFREADYE	70%	Output
AFVALIDE	70%	Input
ATREADYE	70%	Input

D.15 Trace synchronization and trigger timing constraints

The following table shows the trace synchronization and trigger signals timing constraints.

Table D-14 Trace synchronization and trigger signals timing constraints

Name	Timing constraint	Direction
SYNCREQE	50%	Input
SYNCREQI	50%	Input
ETMTRIGOUT	60%	Output
DSYNC	60%	Output
TPIUACTV	80%	Input
TPIUBAUD	10%	Input

D.16 MTB interface timing constraints

The following table shows the MTB interface timing constraints.

Table D-15 MTB interface timing constraints

Name	Timing constraint	Direction
MTBSRAMBASE[31:5]	40%	Input
HSELM	40%	Input
HNONSECM	40%	Input
HTRANSM[1:0]	40%	Input
HBURSTM[2:0]	40%	Input
HADDRM[31:0]	40%	Input
HWRITEM	40%	Input
HSIZEM[2:0]	40%	Input
HWDATAM[31:0]	40%	Input
HPROTM[6:0]	40%	Input
HREADYM	40%	Input
HREADYOUTM	40%	Output
HRDATAM[31:0]	40%	Output
HRESPM	60%	Output
RAMCS	30%	Output
RAMAD[29:0]	30%	Output
RAMRD[31:0]	40%	Input
RAMWD[31:0]	30%	Output
RAMWE[3:0]	30%	Output

D.17 Cross Trigger Interface timing constraints

The following table shows the Cross Trigger Interface timing constraints.

Table D-16 Cross Trigger Interface timing constraints

Signal name	Timing constraint	Direction
CTICHIN[3:0]	60%	Input
CTICHOUT[3:0]	50%	Output
CTIRQ[1:0]	50%	Output

D.18 Interrupts and events signal timing constraints

The following table shows the Interrupts and events signal timing constraints.

Table D-17 Interrupts and events signal timing constraints

Name	Timing constraint	Direction
IRQ[479:0]	70%	Input
NMI	70%	Input
CURRPRI[7:0]	30%	Output
INTNUM[8:0]	70%	Output
TXEV	70%	Output
RXEV	70%	Input
LOCKUP	50%	Output

D.19 WIC interface timing constraints

The following table shows the WIC interface timing constraints.

Table D-18 WIC interface timing constraints

Name	Timing constraint	Direction
WICENREQ	50%	Input
WICENACK	50%	Output
WICSENSE[482:0]	50%	Output

D.20 Implementation Defined Attribution Unit interface timing constraints

The following table shows the *Implementation Defined Attribution Unit*(IDAU) interface timing constraints.

Table D-19 IDAU interface timing constraints

Signal name	Timing constraint	Direction
IDAUADDRA[26:0]	70%	Output
IDAUADDRB[26:0]	70%	Output
IDAUNSA	40%	Input
IDAUNSCA	40%	Input
IDAUNSB	40%	Input
IDAUNSCB	40%	Input
IDAUIDA[7:0]	60%	Input
IDAUIDB[7:0]	60%	Input
IDAUIDVA	60%	Input
IDAUIDVB	60%	Input
IDAUNCHKA	40%	Input
IDAUNCHKB	40%	Input

D.21 Miscellaneous signals timing constraints

The following table shows the miscellaneous signals timing constraints.

Table D-20 Miscellaneous signals timing constraints

Name	Timing constraint	Direction
TSVALUEB[63:0]	50%	Input
TSCLKCHANGE	50%	Input
SYSRESETREQ	70%	Output
TRCENA	70%	Output
ECOREVNUM[35:0]	10%	Input
LOCKSVTAIRCR	20%	Input
LOCKNSVTOR	20%	Input
LOCKSMPU	20%	Input
LOCKNSMPU	20%	Input
LOCKSAU	20%	Input

D.22 FPU signal timing constraints

The following table shows the FPU signal timing constraints.

Table D-21 FPU signal timing constraints

Name	Timing constraint	Direction
FPIXC	50%	Output
FPIDC		
FPOFC		
FPUFC		
FPDZC		
FPIOC		

D.23 DFT interface timing constraints

The following table shows the DFT interface timing constraints.

Table D-22 DFT interface timing constraints

Name	Timing constraint	Direction
DFTRSTDISABLE[1:0]	20%[1]	Input
DFTCGEN		

[1] DFT signals require a multicycle constraint of a two cycle setup and a one cycle hold.

Appendix E
Tarmac Tracing

This appendix describes how to control generation of a `tarmac` trace file during a simulation that traces program execution of a Cortex-M33 processor.

It contains the following sections:

E.1 About tarmac trace

A `tarmac` trace file is a trace of the Cortex-M33 processor program execution captured during simulation.

The output file is a normal text file that you can view in a text editor. In addition to the RTL, the tarmac trace flow uses:

- A simulator capture module library, `teal_tarmac_dpi.so`. The capture module has 32-bit and 64-bit versions to match the simulator you use. The output of the capture module can be saved for later processing or passed directly to the decoder process using a Unix pipe.
- A decoder process that produces the `tarmac` trace file from the capture module raw stream, `teal_tarmac_decode`. The decoder process has both 32-bit and 64-bit versions that you can use with 32-bit and 64-bit simulators respectively.

These binaries have been tested on RHE5 and RHE6 Linux distributions. The signal collection mechanism relies on the SystemVerilog DPI feature, so `tarmac` requires support for this feature in your simulator.

For details of the `tarmac` trace file format, see the *tarmac-format.txt* document in `teal/logical/testbench/shared/tarmac`.

E.2 Setting up the resource requirements for tarmac trace

ARM supplies tests and an execution testbench as a deliverable resource.

This testbench can be configured to create a tarmac trace log, see *Chapter 10 Execution Testbench* on page 10-104.

Related references

Chapter 10 Execution Testbench on page 10-104.

E.3 Running simulation

The execution testbench can be used as a reference to demonstrate the tarmac flow.

When integrating the tarmac flow into your own simulations ensure to:

1. Add logical/testbench/shared/tarmac/verilog directory to the include paths. Within this directory, include the file teal_tarmac.sv in the design compilation. See logical/testbench/execution_tb/verilog/rtl.vc.
2. Define the TEAL_TARMAC and TEAL_TARMAC_DPI Verilog macros.
3. The simulator loads either the 32-bit or the 64-bit teal_tarmac_decode and teal_tarmac_dpi.so files. The libraries are found in logical/testbench/shared/tarmac/linux or logical/testbench/shared/tarmac/linux64 directories. Use:
   a. The 32-bit library for 32-bit simulations.
   b. The 64-bit library for 64-bit simulations.
4. The option for simulation or compilation depends on the simulator tool you use:
   - For ModelSim and IUS use the simulation command:
     ```
     -sv_lib logical/testbench/shared/tarmac/linux<64>/lib/teal_tarmac_dpi.so
     ```
   - For VCS use the compilation command:
     ```
     -sverilog logical/testbench/shared/tarmac/linux<64>/lib/teal_tarmac_dpi.so
     ```
5. To make the tarmac decoder available to the simulator module, add logical/testbench/shared/tarmac/linux<*64*>/bin to your shell PATH environment variable.
6. The tarmac files generated are, by default named, teal_tarmac.<*path*>.log where <path> is a sanitized version of the hierarchical path of the Cortex-M33 processor in your design.

E.4 Tarmac variables

Tarmac generation is controlled using environment variables whose names are `TEAL_TARMAC_<VAR>`, and whose values hold a colon-separated list of pattern=value pairs:

- Pattern is a string (accepting wildcards) to be matched against the targeted instance path (by default the pattern is * if not present). The first matching pattern is taken as the value for that instance.
- Value is the value that the variable takes if pattern matches. It can itself contain the value of another variable, `<VAR2>`, by using the syntax `@VAR2@`. If necessary, the characters `=`, `:`, and `\` can be escaped using `\` so that they can appear in pattern or value strings. For example:
  — `dump.@PATH@.evs` always matches and contains the value of the `<PATH>` variable in its value.
  — `*u_mcu*=dump.0.evs:*u_dbg_drv*=dump.1.evs` has separate values for `u_mcu` and `u_dbg_drv`.

To execute and decode the raw stream, set the environment variables in *`TEAL_TARMAC_<VAR>`*, where *`<VAR>`* can be:

`DECODE_TYPE`
Defines which type of decoder to use.

`PARSER`
Defines how the arguments to the decoder are parsed.

`ARGS`
Defines the arguments.

The following commands are default value examples:

- `TEAL_TARMAC_DECODE_TYPE="teal_tarmac_decode"`
- `TEAL_TARMAC_DECODE_PARSER="cmdline"`
- `TEAL_TARMAC_DECODE_ARGS="-f tarmac.@PATH@.log"`

The `<PATH>` environment variable is built in and has a value corresponding to the sanitized hierarchical path of the processor instance.

Note

- Although the tarmac trace flow uses several built-in variables, you can create other variables for your own use within these variable values. For example, to create the variable `CPUID`, set the environment variable `TEAL_TARMAC_CPUID` to the value you require.

E.5 Controlling tarmac generation

The tarmac environment variable `TEAL_TARMAC_ENABLE` controls whether the raw stream is processed for a particular processor.

The value can be `always`, `never`, or a range `A-B` where `A` is the start time and `B` is the stop time. For example:

- To disable tracing for all Cortex-M33 processor instances, set `TEAL_TARMAC_ENABLE` to `never`.
- To enable tracing always for `u_mcu`, but disable tracing for all other Cortex-M33 processor instances, set `TEAL_TARMAC_ENABLE` to `*u_mcu*=always:never`.
- To trace between a simulation time of 1000 and 10000, set `TEAL_TARMAC_ENABLE` to `1000-10000`.

Time units are the same as those in the tarmac trace.

The raw stream from the capture module is post-processed by the tarmac variable `DECODE_TYPE`. The raw stream can be captured to a file for deferred processing using the tarmac variable `DUMP` that has the value of a filename. The decoder can process this file by running:

```
teal_tarmac_decode -i <dump file> -f tarmac.log
```

The default is to post-process immediately. However, you can disable the default by setting `DECODE_TYPE` to an empty value for that Cortex-M33 processor instance.

Use tarmac variables to set the time unit and scale for simulation to ensure that the `tarmac` trace file contains the correct timing information. The environment variable `TEAL_TARMAC_TIME_UNIT` contains one of the following values: `s`, `ms`, `us`, `ns`, `ps`, `fs`, `clk`, `tic`. The variable `TIME_SCALE` is a scale factor and can use scientific notation in the form aE-b. For example:

- `TEAL_TARMAC_TIME_UNIT=ns`.
- `TEAL_TARMAC_TIME_SCALE=5E-1`.

Sets the timestamp to be 0.5 times the simulation time in nanoseconds.

Appendix F
TEALMCU

This appendix describes the `TEALMCU` module.

It contains the following sections:

F.1 About TEALMCU

The `TEALMCU` is an example integration of the processor with the low area debug and trace components. It is used in the Execution Testbench and you can adapt it for your own requirements.

If you require a fully featured debug and trace implementation, ARM recommends using the CoreSight SoC-400 product, see *ARM® CoreSight™ SoC-400 Technical Reference Manual*.

The `TEALMCU` contains:

- The Cortex-M33 processor, see *1.1 About the processor* on page 1-16.
- The Cortex-M33 DAP, see *ARM® Cortex®-M33 Processor Technical Reference Manual*.
- The Cortex-M33 *Trace Port Interface Unit* (TPIU), see *ARM® Cortex®-M33 Processor Technical Reference Manual*
- A CoreSight system ROM table that identifies the CoreSight components in `TEALMCU`.

The following figure shows a block diagram of the `TEALMCU` module.

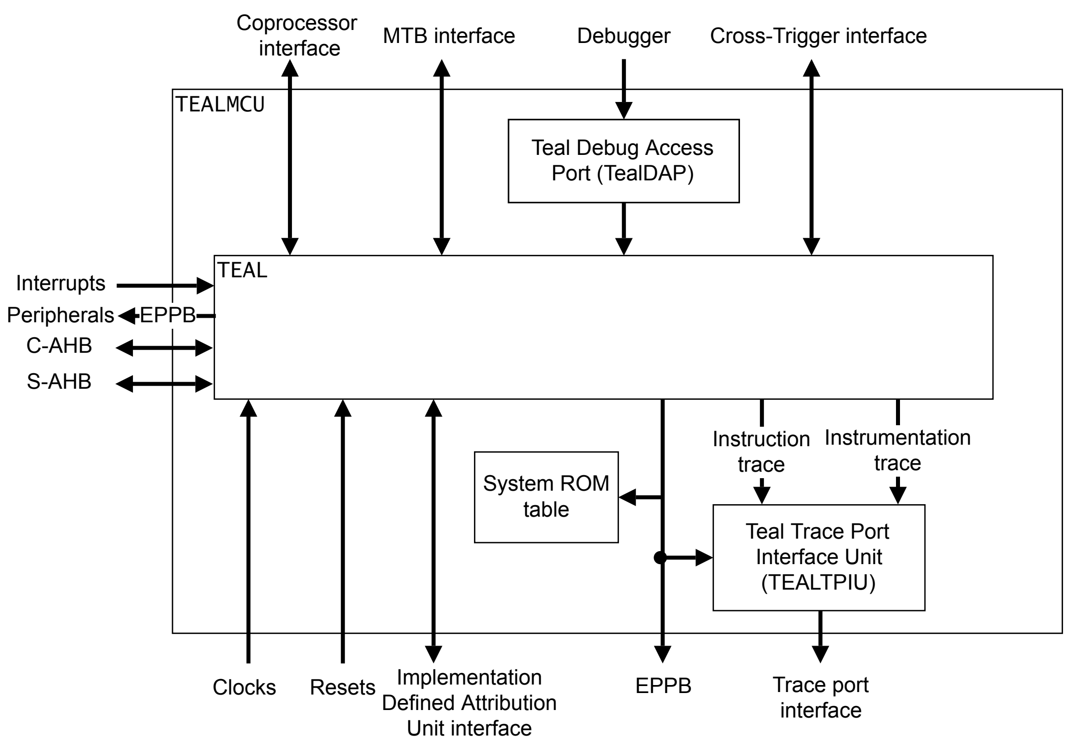


Figure F-1 TEALMCU block diagram

Note

You are permitted to modify only the `TEALMCU.v` file. Do not modify any other files unless this document instructs you to do so, or it is specified in your contract.

F.2 Configuring the TEALMCU level

Most of the configuration options for this module are specified in the `TEAL_CONFIG.v` file.

Parameters configure the remaining options in the module itself.

To configure the `TEALMCU`, either set the parameters on the instantiation of this module in your system or modify the default values of the parameters in the `TEAL_CONFIG.v` and `TEALMCU.v` files.

The following table shows the additional module specific options to configure the `TEALMCU`.

Table F-1 TEALMCU additional configuration options

Parameter	Default value	Supported values	Description
`MCUROMADDR`	`0xE00FE000`	-	Configures the ROM table base address that is read from the DAP during debug sessions.
`DPSEL`	0	0, 1, 2	Debug port selects **0** JTAG. **1** Serial Wire (SW). **2** Switches between SW and JTAG.
`JEPID`	0	-	This 7-bit value is your JEDEC JEP-106 identity code value. It is used in the System ROM table PID1 and PID2 registers.
`JEPCONT`	0	-	This 4-bit value is your JEDEC JEP-106 continuation code value. It is used in the System ROM table PID4 register.
`TARGETID`	`0x00000001`	-	This is a unique identifier for your SoC design. See the *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2* for details. It is used by the TEALDAP and has the bit assignments: **[31:28]** TREVISION. **[27:12]** TPARTNO. **[11:1]** TDESIGNER. **[0]** 1.
`PARTNUM`	0	-	This 12-bit value is a part number that identifies your system. It is used in the System ROM table PID0 and PID1 registers.

You must set the `JEPID`, `JEPCONT`, `PARTNUM`, and `TARGETID` parameter values to uniquely identify your SoC design. See the *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2* for details.

`MCUROMADDR` is the address of the CoreSight ROM table in the MCU. The DAP points to this ROM table first when discovering CoreSight components. The MCU ROM table points to the TPIU and the Cortex-M33 processor ROM table. The Cortex-M33 processor ROM table points to the CoreSight components inside the processor, see the following figure for more information. If the Cortex-M33 processor is configured without both the ITM and ETM, then the MCU ROM and TPIU are not instantiated. In this case, the DAP points directly at the Cortex-M33 processor ROM table.

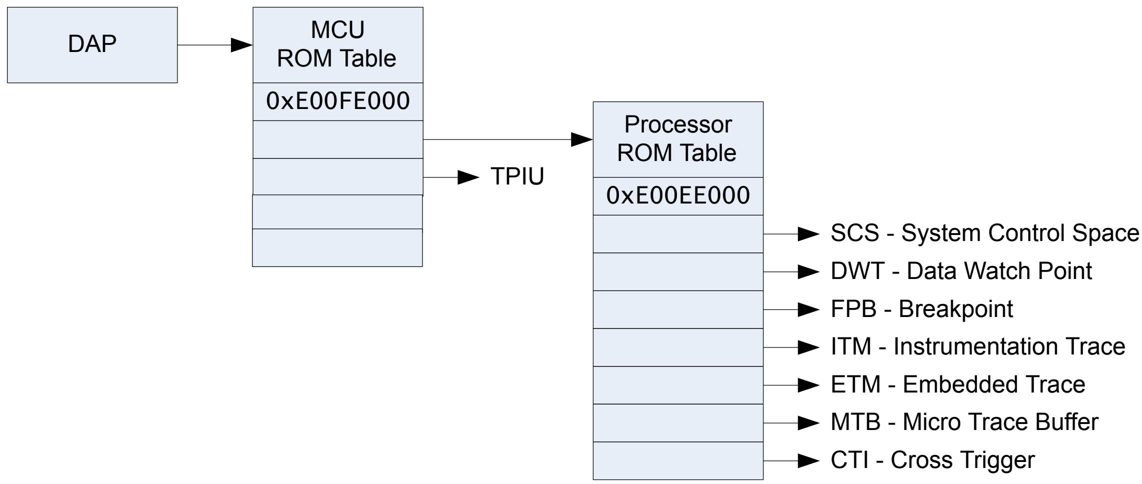


Figure F-2 CoreSight ROM tables

Related concepts

2.2 Configuration options on page 2-27.

F.3 TEALMCU port list

All the port signals that the `TEALMCU` module has are described in this section.

The following table shows the clock and clock enable signals for the `TEALMCU`.

Table F-2 Clock and clock enable signals

Signal name	Direction	Description	Connection information
CLKIN	Input	Primary processor clock. This is gated internally for functional units when required depending on the operating mode of the processor.	**CLKIN** must always be running, unless the processor is powered down.
CORECLKEN	Output	Core clock enable. Indicates whether the Core domain clock is gated or enabled.	**CORECLKEN** must be used to control the system clock that is associated with the AHB bus to ensure that it is always able to accept requests on C-AHB and S-AHB from the processor.
SSTCLKEN	Input	Synchronous enable that is used with **CLKIN** to derive the secure system SysTick clock.	If an asynchronous system clock is used, **SSTCLKEN** must be generated using an appropriate synchronizer circuit followed by an edge detector.
NSSTCLKEN	Input	Synchronous enable that is used with **CLKIN** to derive the Non-secure system SysTick clock.	If an asynchronous system clock is used, **NSSTCLKEN** must be generated using an appropriate synchronizer circuit followed by an edge detector.

The following table shows the reset signals for the `TEALMCU`.

Table F-3 Reset signals

Signal name	Direction	Description	Connection information
nPORESET	Input	Powerup reset. Resets entire processor.	Must be asserted on powerup. This signal is synchronized with **CLKIN** inside the processor. This means **nPORESET** can be asserted and deasserted asynchronously in the system.
nSYSRESET	Input	The **nSYSRESET** signal resets the processor, except the debug logic, D-AHB interface, CTI, MTB, and ETM.	**nSYSRESET** can be asserted without **nPORESET**. It is not necessary to assert **nSYSRESET** on powerup. This signal is synchronized with **CLKIN** inside the processor. This means **nSYSRESET** can be asserted and deasserted asynchronously in the system.
SYSRESETREQ	Output	Request to assert **nSYSRESET**.	Connect to **nSYSRESET** control logic.

The following table shows the static configuration signals for the `TEALMCU`. The static configuration signals that can only be changed at power up reset.

Table F-4 Static configuration signals

Signal name	Direction	Description	Connection information
CFGBIGEND	Input	Static endianness selection for data accesses. **0** Little-endian data. **1** BE8 big-endian data. This signal affects all data side memory interfaces identically except for the PPB region, which is always little-endian. Instructions fetches are always performed as little-endian.	Tie these signals in accordance with your requirements or processor configuration.
CFGSSTCALIB[25:0]	Input	Secure SysTick calibration configuration. **CFGSSTCALIB[23:0]** TENMS **CFGSSTCALIB[24]** SKEW **CFGSSTCALIB[25]** NOREF	
CFGNSSTCALIB[25:0]	Input	Non-secure SysTick calibration configuration. **CFGNSSTCALIB[23:0]** TENMS **CFGNSSTCALIB[24]** SKEW **CFGNSSTCALIB[25]** NOREF	
CFGFPU	Input	If configured, enables support for hardware floating-point	
CFGDSP	Input	If configured, enables support for ARMv8-M DSP extension.	
CFGSECEXT	Input	If configured, enables support for ARMv8-M Security Extension.	
MPUNSDISABLE	Input	If configured, disables support for the Non-secure MPU.	
MPUSDISABLE	Input	If configured, disables support for the Secure MPU.	
SAUDISABLE	Input	If configured, disables support for the SAU.	

The following table shows the reset configuration pins for the `TEALMCU`.

Signal name	Direction	Description	Connection information
INITSVTOR[31:7]	Input	Signals that set the vector base address bit field in the Secure Vector Table Offset Register, VTOR_S.TBLOFF[31:7], out of reset.	Tie these signals in accordance with your requirements or processor configuration.
INITNSVTOR[31:7]	Input	Signals that set the vector base address bit field in the Non-secure Vector Table Offset Register, VTOR_NS.TBLOFF[31:7], out of reset.	

The following table shows the Instruction execution control and hint for the `TEALMCU`.

Table F-5 Instruction execution control signals

Signal name	Direction	Description	Connection information
CPUWAIT	Input	Stall the core out of reset. The **CPUWAIT** signal, when HIGH out of reset, forces the core into a quiescent state. The core boot-up sequence and instruction execution is delayed until this signal is driven LOW. During this time, the processor does not perform any memory accesses. Debugger accesses continue when this signal is HIGH. **CPUWAIT** has no effect if driven HIGH when the processor is running.	-
CODEHINT[2:0]	Output	Prefetch hints. **CODEHINT[2]** When HIGH, indicates that the next instruction fetch transaction is not going to be sequential. For example, the next instruction is to be an unconditional branch or there is an interrupt pending. **CODEHINT[1]** When HIGH, indicates that the following instruction to be executed is a currently unresolved conditional backward branch. **CODEHINT[0]** When HIGH, indicates that the following instruction to be executed is a currently unresolved conditional forward branch. The system can use these signals to control instruction prefetching. The system must always manage these signals as speculative hints. The processor does not guarantee that all non-sequential instruction fetches indicate in advance on **CODEHINT**.	-
IFLUSH	Input	Flush instructions fetched on C-AHB or S-AHB on the previous **HREADY** cycle. **IFLUSH** is held HIGH for one clock cycle. If the previous AHB transaction on either C-AHB or S-AHB was not an instruction fetch, this signal must be LOW. If the previous AHB instruction fetch transaction resulted in an error indicated by **HRESP**, this signal must be LOW.	-
CURRNS	Output	Current Security state of the Cortex-M33 processor: **HIGH** Processor is in Secure state. **LOW** Processor is in Non-secure state. If the Cortex-M33 processor is not configured for ARMv8-M Security Extension support, the **CURRNS** signal is LOW.	-

The following table shows the C-AHB interface signals for the TEALMCU.

Table F-6 C-AHB interface signals

Signal name	Direction	Description	Connection Information
HADDRC[31:0]	Output	Transfer address.	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HBURSTC[2:0]	Output	Transfer burst length.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HEXCLC	Output	Exclusive request. Address phase control signal that indicates whether an access is because of an `LDREX` or `STREX` instruction: **0** Non-exclusive, standard, transaction. **1** Exclusive transaction.	To support exclusive transfers on the C-AHB interface, connect this signal to a global exclusive monitor, otherwise leave it unconnected.
HEXOKAYC	Input	Exclusive response. Data phase signal that is sampled on **HREADYC** that indicates whether the exclusive request was granted or not: **0** Exclusive access has failed. **1** Exclusive access is successful.	To support exclusive transfers to shared memory on the C-AHB interface, connect this signal to a global exclusive monitor, otherwise tie it LOW.
HHINTC[2:0]	Output	Hint signal. This signal is synchronous to **HTRANSC[1:0]**. **HHINTC[0]** Indicates that a read access is to an exception vector. **HHINTC[1]** Indicates that a PC relative load instruction has generated a read access. **HHINTC[2]** Indicates whether the read or write access was associated with Handler or Thread mode. 1 indicates access is a Handler mode fetch or read/write in Handler mode. 0 indicates access is a Thread mode fetch or read/write in Thread mode.	-
HINNERC[4:0]	Output	Inner memory attributes using the same format as **HPROTC[6:2]**.	-
HMASTERC	Output	Initiator of the Transfer. **0** Processor. **1** Debugger.	-
HNONSECC	Output	Security level, asserted to indicate a Non-secure transfer.	-
HPROTC[6:0]	Output	Protection and outer memory attributes.	-
HRDATAC[31:0]	Input	Read data.	-
HREADYC	Input	Slave ready.	-
HRESPC	Input	Slave response.	-
HSIZEC[2:0]	Output	Transfer size.	-
HTRANSC[1:0]	Output	Transfer type.	-

Table F-6 C-AHB interface signals (continued)

Signal name	Direction	Description	Connection Information
HWDATAC[31:0]	Output	Write data.	-
HWRITEC	Output	Write transfer.	-

The following table shows the S-AHB interface signals for the `TEALMCU`.

Table F-7 S-AHB interface signals

Signal name	Direction	Description	Connection Information
HADDRS[31:0]	Output	Transfer address.	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HBURSTS[2:0]	Output	Transfer burst length.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HEXCLS	Output	Exclusive request. Address phase control signal that indicates whether an access is because of an `STREX` or `LDREX` instruction: **0** Non-exclusive, standard, transaction. **1** Exclusive transaction.	To support exclusive transfers on the S-AHB interface, connect this signal to the global exclusive monitor, otherwise leave it unconnected.
HEXOKAYS	Input	Exclusive response. Data phase signal sampled on **HREADYS** that indicates whether the exclusive request was granted or not: **0** Exclusive access has failed. **1** Exclusive access is successful.	To support exclusive transfers to shared memory on the S-AHB interface, connect this signal to a global exclusive monitor, otherwise tie it LOW.
HHINTS[2:0]	Output	Hint signal. This signal is synchronous to **HTRANSS**. **HHINTS[0]** Indicates that a read access is to an exception vector. **HHINTS[1]** Indicates a PC relative load instruction generates a read access. **HHINTC[2]** Indicates whether the read or write access was associated with Handler or Thread mode. 1 indicates access is a Handler mode fetch or read/write in Handler mode. 0 indicates access is a Thread mode fetch or read/write in Thread mode.	Connect masters through the bus infrastructure.
HINNERS[4:0]	Output	Inner memory attributes using the same format as **HPROTS[6:2]**.	-
HMASTERS	Output	Initiator of the access: **0** Processor. **1** Debugger.	-
HNONSECS	Output	Security level, asserted to indicate a Non-secure transfer.	-
HPROTS[6:0]	Output	Protection and outer memory attributes.	-

Table F-7 S-AHB interface signals (continued)

Signal name	Direction	Description	Connection Information
HRDATAS[31:0]	Input	Read data.	-
HREADYS	Input	Slave ready.	-
HRESPS	Input	Slave response.	-
HSIZES[2:0]	Output	Transfer size.	-
HTRANSS[1:0]	Output	Transfer type.	-
HWDATAS[31:0]	Output	Write data.	-
HWRITES	Output	Write transfer.	-

The following table shows the EPPB signals for the `TEALMCU`.

Table F-8 EPPB signals

Signal name	Direction	Description	Connection information
PSEL	Output	APB device select. Indicates that a data transfer is requested.	Connect to your system CoreSight components as required. **Note** ARM recommends that all non-debug peripherals are integrated on the S-AHB interface.
PENABLE	Output	APB control signal. Strobe to time all accesses. Indicates the access phase of an APB transfer.	
PPROT[2:0]	Output	Transfer privilege and security level.	
PWRITE	Output	APB transfer direction. Write not read.	
PADDR[19:2]	Output	APB 18-bit Address bus. Only the bits that are relevant to the External Private Peripheral Bus are driven.	
PADDR31	Output	Initiator of the transfer. This signal is driven HIGH when the DAP is the requesting master. It is driven LOW when the processor is the requesting master.	
PWDATA[31:0]	Output	APB 32-bit write data bus.	
PREADY	Input	APB slave ready signal. This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.	
PSLVERR	Input	APB slave error signal. This signal is driven HIGH if the currently accessed APB device cannot handle the requested transfer.	
PRDATA[31:0]	Input	APB 32-bit read data bus.	

The following table shows the external coprocessor interface signals for the TEALMCU.

Table F-9 External coprocessor Interface signals

Signal name	Direction	Description	Connection information
CPENABLED[7:0]	Output	Indicates which coprocessor is enabled in the: • *Coprocessor Access Control Register* (CPACR) associated with the security state of the processor. • *Non-Secure Access Control Register* (NSACR) register if the processor is executing in Non-secure state. **Note** The CPACR is banked when the implementation includes the ARMv8-M Security Extension. See *CPENABLED* on page 4-56.	Connect to the external coprocessors.
CPPWRSU[7:0]	Output	Indicates which coprocessors are permitted to become UNKNOWN. See *CPPWRSU* on page 4-57.	
CPSPRESENT[7:0]	Input	Indicates which Secure coprocessors are present in the system. See *CPSPRESENT, CPNSPRESENT* on page 4-57.	
CPNSPRESENT[7:0]	Input	Indicates which Non-secure coprocessors are present in the system. See *CPSPRESENT, CPNSPRESENT* on page 4-57.	
CPCDP	Output	Coprocessor command operation. See *CPCDP, CPMCR, CPMRC* on page 4-57.	
CPMCR	Output	Coprocessor register transfer from processor operation. See *CPCDP, CPMCR, CPMRC* on page 4-57.	
CPMRC	Output	Coprocessor register transfer to processor operation. See *CPCDP, CPMCR, CPMRC* on page 4-57.	
CPSIZE	Output	Coprocessor size operation. See *CPSIZE* on page 4-57.	
CPNUM[2:0]	Output	Coprocessor number request. See *CPNUM* on page 4-57.	
CPREGS[11:0]	Output	Operation register fields. See *CPREGS* on page 4-58.	
CPOPC[8:0]	Output	Operation opcode fields. See *CPOPC* on page 4-58.	
CPPRIV	Output	Indicates operation privilege. See *CPPRIV, CPNSATTR* on page 4-58.	
CPNSATTR	Output	Indicates operation security state. See *CPPRIV, CPNSATTR* on page 4-58.	
CPVALID	Output	Indicates whether the coprocessor operation is valid. See *CPVALID* on page 4-59.	
CPREADY	Input	Indicates whether the coprocessor is stalled or ready. See *CPREADY* on page 4-59.	
CPERROR	Input	Indicates that the coprocessor is not present or the instruction is not supported. See *CPERROR* on page 4-59.	
CPWDATA[63:0]	Output	The coprocessor write data bus. See *Data signals* on page 4-59.	
CPRDATA[63:0]	Input	The coprocessor read data bus. See *Data signals* on page 4-59.	

The following table shows the debug signals for the `TEALMCU`.

Table F-10 Debug signals

Signal name	Direction	Description	Connection information
HALTED	Output	In halting mode debug. **HALTED** remains asserted while the processor is in debug.	Connect to external CoreSight CTI when an internal CTI is not implemented.
DBGRESTART	Input	Request for synchronized exit from halt mode. Forms a handshake with **DBGRESTARTED**. If multiprocessor debug support is not required, **DBGRESTART** must be tied LOW.	
DBGRESTARTED	Output	Handshake for **DBGRESTART**.	
EDBGRQ	Input	External debug request. A debug agent in the system asserts this signal to request that the processor enters Debug state.	
DBGEN	Input	Invasive debug enable.When LOW, disables all halt-mode and invasive debug features.	Tie-off or connect to debug authentication module.
NIDEN	Input	Non-invasive debug enable. When LOW, disables all trace and non-invasive debug features.	Either tie HIGH or connect to debug authentication module.
SPIDEN	Input	Secure invasive debug enable. When LOW, disables all halt mode and invasive debug features when the processor is in Secure state.	Tie-off or connect to debug authentication module.
SPNIDEN	Input	Secure non-invasive debug enable. Controls access to non-invasive debug features when the processor is in secure state and **SPIDEN** is LOW.	Tie-off or connect to debug authentication module.
CTICHIN[3:0]	Input	CTI channel input.	Either connect to **CTICHOUT** of system level CTI or CTM, or tie all LOW.
CTICHOUT[3:0]	Output	CTI channel output.	Either connect to **CTICHIN** of system CTI or CTM, or leave unconnected.
CTIIRQ[1:0]	Output	CTI interrupt, active HIGH.	Either connect to two of **IRQ[479:0]** inputs or an external interrupt controller, or leave unconnected.

The following table shows the power control and sleep signals for the `TEALMCU`.

Table F-11 Power control and sleep interface signals

Signal name	Direction	Description	Connection information
SLEEPING	Output	When HIGH indicates that the processor is ready to enter a low-power state. When LOW, indicates that the processor is running or wants to leave sleep mode. If **SLEEPHOLDACKn** is LOW, then the processor does not perform any fetches until **SLEEPHOLDREQn** is driven HIGH.	Connect to your power management unit
SLEEPDEEP	Output	Indicates that the processor and ETM are ready to enter a low-power state and the wake-up time is not critical. Only active when **SLEEPING** is HIGH.	
SLEEPHOLDACKn	Output	Acknowledge signal for **SLEEPHOLDREQn**. If this signal is LOW, irrespective of the **SLEEPING** signal value, the processor does not advance in execution and does not perform any memory operations.	Connect to your power management unit
SLEEPHOLDREQn	Input	Request to extend the processor sleeping state regardless of wake-up events. If the processor acknowledges this request driving **SLEEPHOLDACKn** LOW, this guarantees the processor remains idle even on receipt of a wake-up event.	
WICSENSE[482:0]	Output	Active HIGH signal. Indicates which input events can cause the WIC to generate the **WAKEUP** signal. The `WICLINES` configuration parameter determines the usable width of this signal. Therefore only the **WICSENSE**[`WICLINES`-1:0] bits are implemented and the remaining bits are driven LOW. The mapping to input events is: **WICSENSE[482:3]** **IRQ[479:0]**. **WICSENSE[2]** **EDBGRQ**. **WICSENSE[1]** **NMI**. **WICSENSE[0]** **RXEV**.	Connect to low-power control logic, or leave unconnected if the WIC is not present
WICENREQ	Input	Active HIGH request for deep sleep to be WIC-based deep sleep. This is driven from the power management unit.	Connect to low-power control logic, or tie LOW if the WIC is not present
WICENACK	Output	Active HIGH acknowledge signal for **WICENREQ**.	Connect to low-power control logic, or leave unconnected if the WIC is not present
WAKEUP	Output	Active HIGH signal to the power management unit that indicates a wake-up event has occurred and the processor system domain requires its clocks and power restored.	

Table F-12 Q-Channel interface signals

Signal name	Direction	Description	Connection information
COREQREQn	Input	Core quiescence request signal	Connect to your power management unit. **Note** For more information on the Q-Channel interface signals, see the *Low Power Interface Specification ARM® Q-Channel and P-Channel Interfaces*
COREQACCEPTn	Output	Core quiescence request accepted	
COREQDENY	Output	Core quiescence request denied	
COREQACTIVE	Output	Core active or activation request	
FPUQREQn	Input	FPU domain quiescence request signal	
FPUQACCEPTn	Output	FPU domain quiescence request accepted	
FPUQDENY	Output	FPU domain quiescence request denied	
FPUQACTIVE	Output	FPU logic active or activation request	
DBGQREQn	Input	Debug domain quiescence request signal	
DBGQACCEPTn	Output	Debug domain quiescence request accepted	
DBGQDENY	Output	Debug domain quiescence request denied	
DBGQACTIVE	Output	Debug logic active or activation request	
MTBQREQn	Input	MTB domain quiescence request signal	
MTBQACCEPTn	Output	MTB domain quiescence request accepted	
MTBQDENY	Output	MTB domain quiescence request denied	
MTBQACTIVE	Output	MTB logic active or activation request	
CORERET	Input	Core power domain in retention	
FPURET	Input	FPU power domain in retention	
DBGRET	Input	Debug power domain in retention	

The following table shows the Teal DAP signals for the `TEALMCU`.

Table F-13 Teal DAP signals

Signal name	Direction	Description	Connection information
nTRST	Input	JTAG test logic reset	Connect to your external debug controller
TDI	Input	JTAG data in	
TDO	Output	JTAG data out	

Table F-13 Teal DAP signals (continued)

Signal name	Direction	Description	Connection information
nTDOEN	Output	JTAG TDO output enable	Connect to your external debug controller
SWDITMS	Input	Serial wire data in or JTAG TMS	
SWDO	Output	Serial wire data out	
SWDOEN	Output	Serial wire data out enable	
DAPEN	Input	DAP enable	Tie this signal LOW to disable the DAP
CDBGPWRUPREQ	Output	DAP powerup request	Connect to your Power Management Unit
CDBGPWRUPACK	Input	DAP powerup acknowledge	
INSTANCEID[3:0]	Input	Serial Wire or Debug Port Instance ID	Tie this signal LOW to disable the DAP
JTAGSEL	Output	DAP JTAG-DP select	-
SWSEL	Output	DAP SW-DP select	-

The following table shows the TPIU signals for the `TEALMCU`.

Table F-14 TPIU signals

Signal name	Direction	Description	Connection information
MAXPORTSIZE[1:0]	Input	Max TPIU port size	Connect to your external trace capture device
TRACECLK	Output	TRACECLK Output	
TRACEDATA[3:0]	Output	Trace Data	
TRACESWO	Output	Single Wire Output	
SWOACTIVE	Output	SWO mode selected	
TRACEPORTSIZE[1:0]	Output	Trace port size	
SWCLKTCK	Input	SW/JTAG DP clock	
TRACECLKIN	Input	TPIU trace port clock	
TRESETn	Input	TPIU trace port reset	

The following table shows the miscellaneous interface signals for the `TEALMCU`.

Table F-15 Miscellaneous interface signals

<table>
<tr><th>Signal name</th><th>Direction</th><th>Description</th><th>Connection information</th></tr>
<tr><td>ECOREVNUM[35:0]</td><td>Input</td><td>ECO revision number. The ECO revision field mappings are:
[35:32] MTB.
[31:28] ETM.
[27:24] CTI.
[23:20] ROM table.
[19:16] ITM.
[15:12] SCS.
[11:8] DWT.
[7:4] BPU.
[3:0] CPUID revision.</td><td>Tie all bits LOW. This signal must be brought up to the top level on your SoC design to prevent synthesis tools optimizing out the logic this signal drives. ARM provides instructions on how to tie these signals in the event of an ECO change.</td></tr>
<tr><td>TRCENA</td><td>Output</td><td>Trace Enable. This signal reflects the setting of the DEMCR.TRCENA, indicating that the DWT and ITM units are enabled (when implemented).</td><td>Connect to clock gating and power gating logic for the TPIU.</td></tr>
<tr><td>TSVALUEB[63:0]</td><td>Input</td><td>Global timestamp value.</td><td>Connect to a natural binary count value if global timestamping is required. If not used, you must tie all bits LOW.</td></tr>
<tr><td>TSCLKCHANGE</td><td>Input</td><td>Timestamp clock ratio change.</td><td>Pulse this input if either CLKIN or the timestamp clock changes, even if the ratio does not change. Tie LOW if TSVALUEB is not used, or if TSVALUEB is generated from CLKIN.</td></tr>
<tr><td>LOCKSVTAIRCR</td><td>Input</td><td>Asserting this signal prevents changes to:
• The Secure vector table base address.
• Handling of Secure interrupt priority.
• BusFault, HardFault, and NMI security target settings in the processor.
When this signal is:
HIGH Disables writes to the VTOR_S, AIRCR.PRIS, and AIRCR.BFHFNMINS registers.
LOW Unlocks these registers.</td><td>This signal can be changed dynamically. If you want the registers unlocked, tie LOW, otherwise drive with external logic.
Caution
Tying these signal HIGH causes loss of interrupt control.</td></tr>
<tr><td>LOCKNSVTOR</td><td>Input</td><td>Asserting this signal prevents changes to the Non-secure vector table base address.
When this signal is:
HIGH Disables writes to the VTOR_NS register.
LOW Unlocks this register.</td><td>These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic.
Caution
Tying these signal HIGH causes loss of interrupt control.</td></tr>
</table>

Table F-15 Miscellaneous interface signals (continued)

Signal name	Direction	Description	Connection information
LOCKSMPU	Input	Asserting this signal prevents changes to programmed Secure MPU memory regions and all writes to the registers are ignored. When this signal is: **HIGH** Disables writes to the MPU_CTRL, MPU_RNR, MPU_RBAR, MPU_RLAR, MPU_RBAR_An and MPU_RLAR_An from software or from a debug agent connected to the processor in Secure state. **LOW** Unlocks these registers. This signal has no affect if the Cortex-M33 processor has not been configured with support for the ARMv8-M Security Extension, or if no Secure MPU regions have been configured.	These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic. **Caution** Tying these signal HIGH causes loss of memory protection control.
LOCKNSMPU	Input	Asserting this signal prevents changes to Non-secure MPU memory regions already programmed. All writes to the registers are ignored. **HIGH** disables writes to the MPU_CTRL_NS, MPU_RNR_NS, MPU_RBAR_NS, MPU_RLAR_NS, MPU_RBAR_A_NSn and MPU_RLAR_A_NSn from software or from a debug agent connected to the processor. **LOW** Unlocks these registers. This signal has no affect if the Cortex-M33 processor has been configured without any Non-secure MPU regions.	These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic. **Caution** Tying these signal HIGH causes loss of memory protection control.
LOCKSAU	Input	Asserting this signal prevents changes to Secure SAU memory regions already programmed. All writes to the registers are ignored. **HIGH** Disables writes to the SAU_CTRL, SAU_RNR, SAU_RBAR and SAU_RLAR registers from software or from a debug agent connected to the processor. **LOW** Unlocks these registers. This signal has no affect if the Cortex-M33 processor has not been configured with support for the ARMv8-M Security Extension, or if no SAU regions have been configured.	These signals can be changed dynamically. If you want the registers unlocked, tie all bits LOW, otherwise drive with external logic. **Caution** Tying these signal HIGH causes loss of security attribution control.

The following table shows the external maskable and non-maskable interrupts signals for the `TEALMCU`.

Table F-16 Interrupt interface

Signal name	Direction	Description	Connection information
IRQ[479:0]	Input	External interrupt signals. The `NUMIRQ` parameter configures the implemented bits of this signal. **Note** • **IRQ** and **NMI** signals are active HIGH and the hardware is agnostic between pulse- and level-signaled interrupts. • You must ensure that the **IRQ** and **NMI** signals to the processor are synchronized to **CLKIN** using the appropriate circuit.	Connect to interrupt logic. The number of functional interrupt signals depends on your implementation. Tie any bits that are not implemented LOW.
NMI	Input	Non-Maskable Interrupt.	
CURRPRI[7:0]	Output	Current interrupt priority level	Might be connected to your interrupt logic or remain unconnected.
INTNUM[8:0]	Output	Interrupt number of the current execution context, from the *Interrupt Program Status Register* (IPSR). **Note** When the processor is in Thread mode, **INTNUM** is 0.	

The following table shows the FPU exception signals for the `TEALMCU`.

Table F-17 FPU signals

Signal name	Direction	Description	Connection information
FPIXC	Output	Masked floating-point inexact exception	Cumulative exception flags from the *Floating Point Status and Control Register* (FPSCR). These signals indicate when a floating-point exception has occurred.
FPIDC	Output	Masked floating-point input denormal exception	
FPOFC	Output	Masked floating-point overflow exception	
FPUFC	Output	Masked floating-point underflow exception	
FPDZC	Output	Masked floating-point divide-by-zero exception	
FPIOC	Output	Invalid operation	

The following table shows the event and Lockup signals for the `TEALMCU`.

Table F-18 Events and errors

Signal name	Direction	Description	Connection information
TXEV	Output	Event transmitted as a result of `SEV` instruction. This is a single-cycle pulse. You can use it to implement a more power efficient spin-lock in a multiprocessor system.	Connect to other processors in a multiprocessor system. In a multiprocessor system, **TXEV** from each processor can be broadcast to the **RXEV** input of the other processors. Leave unconnected if not required.
RXEV	Input	When HIGH this signal sets the Event register in the processor that is defined in the ARMv8-M architecture. This causes a WFE instruction to complete. It also wakes up the processor, if it is sleeping because it executed a WFE instruction.	You must construct the input to this signal as the logical-OR of all non-interrupt event generating sources of interest in your system. For example, the **TXEV** output of other ARM processors, or a single cycle completion signal from peripherals not already connected to any interrupt lines. You must add synchronization logic if this signal is driven from a different clock domain. Tie this input LOW if there are no non-interrupt event generating sources in your system.
LOCKUP	Output	When HIGH, indicates that the processor is in the architected Lockup state, because of an unrecoverable exception. See the *ARM®v8-M Architecture Reference Manual* for more information.	You can connect this signal to your own logic, for example a watchdog device, that can reset the processor using **nSYSRESET**. If your system executes instructions from a programmable memory, for example flash, after powerup, you must consider how that memory is programmed. The processor might enter Lockup state very quickly if the memory is uninitialized. If **nSYSRESET** is asserted immediately, there might not be enough time to connect a debugger to halt the processor and leave Lockup state. ARM recommends that your watchdog logic includes a software-programmable enable bit that gates the assertion of **nSYSRESET** because of **LOCKUP**. If you require entry into Lockup state to reset the system, your code must enable the functionality in your watchdog unit.

The following table shows the signals on external IMPLEMENTATION DEFINED ATTRIBUTION UNIT (IDAU) interface for the `TEALMCU`.

Table F-19 External SAU Interface

Signal Name	Direction	Description	Connection Information
IDAUADDRA[26:0]	Output	Address of the region. **IDAUADDRA[26:0]** is the 32-byte IDAU region associated with the access address. For example, for a 32-bit memory address A, **IDAUADDRA[26:0]** is **A[31:5]**.	-
IDAUADDRB[26:0]	Output	Address of the region. **IDAUADDRB[26:0]** is the 32-byte IDAU region associated with the access address. For example, for a 32-bit memory address B, **IDAUADDRB[26:0]** is **B[31:5]**.	-
IDAUNSA	Input	Non-secure region response. The **IDAUNSA** signal defines the attributes of the IDAU region.	Tie this input HIGH if an IDAU is not included.

Table F-19 External SAU Interface (continued)

Signal Name	Direction	Description	Connection Information
IDAUNSB	Input	Non-secure region response. The **IDAUNSB** signal defines the attributes of the IDAU region.	Tie this input HIGH if an IDAU is not included.
IDAUNSCA	Input	Non-secure-callable region response. The **IDAUNSCA** signal defines the attributes of the IDAU region.	Tie this input LOW if an IDAU is not included.
IDAUNSCB	Input	Non-secure-callable region response. The **IDAUNSCB** signal defines the attributes of the IDAU region.	Tie this input LOW if an IDAU is not included.
IDAUIDA[7:0]	Input	Region number. **IDAUIDA[7:0]** is the 8-bit region identifier associated with the IDAU region. The value is written to the IREGION field of the result register value, Rd[31:24], the destination register of a TT instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.
IDAUIDB[7:0]	Input	Region number. **IDAUIDB[7:0]** is the 8-bit region identifier associated with the IDAU region. The value is written to the IREGION field of the result register value, Rd[31:24], the destination register of a TT instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.
IDAUIDVA	Input	Region number valid. **IDAUIDVA** indicates that the IDAU region number is valid. The value is written to the IRVALID field of the result register value, Rd[23], the destination register of a TT instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.
IDAUIDVB	Input	Region number valid. **IDAUIDVB** indicates that the IDAU region number is valid. The value is written to the IRVALID field of the result register value, Rd[23], the destination register of a TT instruction when the instruction is executed in Secure state.	Tie this input LOW if an IDAU is not included.
IDAUNCHKA	Input	Region exempt from attribution check. When **IDAUNCHKA** is HIGH, the address associated with the IDAU region is not subject to attribution or security checks. The security attribution is determined only by the processor security state for software reads and writes to the address or by **HNONSECD** on D-AHB and DHCSR.S_SDE for debug accesses. This behavior is independent of any security attribution associated with the address in the processor SAU or presented on the IDAU interface. When **IDAUNCHKA** is LOW, then the security attribution is determined by the processor SAU or IDAU unless the address is specified as always exempt from security attribution checks.	Tie this input LOW if an IDAU is not included
IDAUNCHKB	Input	Region exempt from attribution check. When **IDAUNCHKB** is HIGH, the address associated with the IDAU region is not subject to attribution or security checks. The security attribution is determined only by the processor security state for software reads and writes to the address or by **HNONSECD** on D-AHB and DHCSR.S_SDE for debug accesses. This behavior is independent of any security attribution associated with the address in the processor SAU or presented on the IDAU interface. When **IDAUNCHKB** is LOW, then the security attribution is determined by the processor SAU or IDAU unless the address is specified as always exempt from security attribution checks.	Tie this input LOW if an IDAU is not included.

The MTB interface connects to SRAM and can be used for both trace and general-purpose storage by the processor.

Table F-20 MTB interface signals

Signal name	Direction	Description	Connection information
MTBSRAMBASE[31:5]	Input	Location of MTB SRAM in processor memory map.	Connect to your trace or general-purpose storage as required.
HSELM	Input	Select access to MTB SRAM.	
HTRANSM[1:0]	Input	Transfer type.	
HBURSTM[2:0]	Input	Transfer burst length.	
HADDRM[31:0]	Input	Transfer address.	
HWRITEM	Input	Write transfer.	
HSIZEM[2:0]	Input	Transfer size.	
HNONSECM	Input	MTB AHB security level request. When asserted, **HNONSECM** indicates a Non-secure transfer.	
HWDATAM[31:0]	Input	Write data.	
HPROTM[6:0]	Input	Protection and outer memory attributes.	
HREADYM	Input	Ready for MTB.	
HREADYOUTM	Output	Ready out of MTB.	
HRDATAM[31:0]	Output	Read data.	
HRESPM	Output	Slave response.	
RAMCS	Output	RAM chip select.	
RAMAD[29:0]	Output	RAM address.	
RAMRD[31:0]	Input	RAM read data.	
RAMWD[31:0]	Output	RAM write data.	
RAMWE[3:0]	Output	RAM Write byte strobes.	

Appendix G
IP-XACT

This appendix describes the location and configuration of the IP-XACT files.

It contains the following sections:

G.1 About IP-XACT for Cortex®-M33 processor

IP-XACT for Cortex-M33 processor is an XML description file of the `TEAL` module. This file describes the component interfaces and I/O ports within IP-XACT IEEE 1685-2009.

Because it is supplied in an unconfigured state, you must process the XML description file to remove information, as appropriate, to match your required configuration, see *G.3 Generating the IP-XACT description* on page Appx-G-252.

Related information

Accellera: IP-XACT website.

G.2 Location of the IP-XACT description file

The following table shows the IP-XACT description file and hierarchy level, the Verilog module, it is associated with.

Table G-1 IP-XACT files

Verilog module	IP-XACT description	Description
`verilog/TEAL.v`	`ipxact/TEAL.xml`	The Cortex-M33 processor level containing the processor, debug, FPU, MTB, WIC, ETM, and CTI. When the IP-XACT configuration has been completed, the `.xml` file is in the `logical/teal/ipxact` directory.

The `Teal_unconfigured.xml` file supplied represents the configurable, but unconfigured, descriptions of the named levels of hierarchy.

Note

There is no IP-XACT description for the `TEALMCU` level of hierarchy.

Related concepts

G.3 Generating the IP-XACT description on page Appx-G-252.

G.3 Generating the IP-XACT description

This section describes how you can generate an IP-XACT description that is based on your chosen configuration of the IP.

To generate an IP-XACT description, you must run the `build_ipxact_component_teal.pl` script on an unconfigured, source IP-XACT file, and supply the Verilog configuration parameters that are described in *2.2 Configuration options* on page 2-27.

Note

- The `build_ipxact_component_teal.pl` script uses the `ipxact_lib_teal.pm` module. The script requires Perl utilities and Perl XML libraries. The PERL5LIB environment variable must be updated to include the path to the `ipxact_lib_teal.pm` module.
- For more information about the script, see the README file in `<trunk>/logical/teal/ipxact`.

You can run the `build_ipxact_component_teal.pl` script from the `logical/teal/ipxact` directory:

```
cd <path>/logical/teal/ipxact
```

Run `build_ipxact_component_teal.pl`, supplying the name of the IP-XACT file and the configuration file.

For example:

```
../../shared/tools/bin/build_ipxact_component_teal.pl -unconfigured_xml
Teal_unconfigured.xml -moduletop <name> -config <config_file> -keepdepends -override
```

The configuration file must be in the format:

```
parameter_name=parameter_value
```

The `build_ipxact_component_teal.pl` script generates the configured IP-XACT file in the same directory as the unconfigured source IP-XACT file. The default name for the configured IP-XACT file is the name of the top-level Verilog module. In this case, the default name is `TEAL.xml`. The name can be overridden using the `-moduletop <name>` option.

All this information can also be found in the file `logical/teal/ipxact/README`.

Related concepts

2.2 Configuration options on page 2-27.

G.4 Using the IP-XACT description

You can use the generated IP-XACT descriptions with IP-XACT aware EDA tools for RTL stitching. The IP-XACT descriptions reference bus definition files to describe the module interfaces. Copies of the ARM bus definition files are in the `logical/shared/ipxact/busdefs` directory.

The following figure shows the structure of the IP-XACT busdefs directory.

```
shared/
 └──ipxact/
     └──busdefs/
         ├──amba.com/
         │   ├──AMBA3/
         │   │   └──APB/
         │   ├──AMBA4/
         │   │   ├──ATB/
         │   │   └──APB4
         │   └──AMBA5/
         │       ├──AHB5Target/
         │       └──AHB5Initiator/
         └──arm.com/
             ├──CortexM_Cores
             │   └──IDAU/
             │       ├──SysTick/
             │       ├──coprocessor/
             │       ├──Sleep/
             │       ├──SMP/
             │       └──WICExternal/
             ├──CoreSight/
             │   └──EVENT/
             │       ├──Channel/
             │       ├──WTimestamp/
             │       └──Authentication/
             └──generic/
                 └──Staticcfg/
                     ├──interrupt/
                     ├──Q-Channel/
                     ├──RESET/
                     ├──DFTInterface/
                     ├──Wakeup/
                     ├──SRAM_sp_basic/
                     ├──clock/
                     ├──Status/
                     ├──InterruptInterface/
                     └──DynamicConfig/
```

Figure G-1 IP-XACT busdefs directory structure

Appendix H
Revisions

This appendix describes the technical changes between released issues of this document.

It contains the following section:

H.1 Revisions

This appendix describes the technical changes between released issues of this book.

Table H-1 Issue 0000-00

Change	Location	Affected
First release	-	-

Table H-2 Differences between issue 0000-00 and issue 0001-00

Change	Location	Affected
Updated Code and System AHB interfaces description	*4.6 Code and System AHB interfaces* on page 4-44	All
Clarified that for debug accesses, Debug data transfers appear as SINGLE, tied to `0b000`	*Table 1*	All
Clarified the use of the `EXPECTED_SAU` configuration option	*Table 10-4 Execution testbench processor configuration options* on page 10-114	All

Table H-3 Differences between issue 0001-00 and issue 0002-00

Change	Location	Affected
Clarified that the ETM and MTB parts are licensed and delivered separately from the processor	*1.4 Reference data* on page 1-24 *10.3 Test overview* on page 10-108	All
Clarified that the **CORECLKEN** must be used to control the system clock that is associated with the AHB bus	*4.2.1 Clock and clock enable signals* on page 4-36 *F.3 TEALMCU port list* on page Appx-F-232	All
Updated the connection information for the EPPB signals	*4.8 External Private Peripheral Bus* on page 4-52	All
Clarified coprocessor number in example operation transactions	*4.9 External coprocessor interface* on page 4-53	All
Added `teal_cdc_connect.v` DAP component cell to table.	*6.3.1 Special purpose cells* on page 6-87	All
Clarified values for `EXPECTED_MPU_NS` and `EXPECTED_MPU_S`	*Execution testbench processor configuration options* on page 10-114	All
Added Running with a DSM topic	*10.8.2 Running with a DSM* on page 10-123	All
Clarified the interconnect that is used in figures	*10.11.1 Testbench structure* on page 10-128	All
Updated the Sleep-hold interface description	*11.5 Sleep-hold interface* on page 11-151	All
Added missing example of WIC sleep clock control figure	*11.6.3 WIC sleep* on page 11-152	All
Updated **CTICHIN** description in Core power domain Q-Channel behavior table	*12.4.3 Domain behavior* on page 12-162	All
Clarified that the **CORECLKEN** signal synchronizes the system level clock gating	*12.5.1 System level clock gating* on page 12-165	All
Clarified the TPIU signals used	*C.3 Trace Port Interface Unit signals* on page Appx-C-195	All
Clarified the DAP signals used	*C.1 Debug Access Port signals* on page Appx-C-191	All

Table H-3 Differences between issue 0001-00 and issue 0002-00 (continued)

Change	Location	Affected
Clarified the signals used	*D.3 Configuration and initialization signal timing constraints* on page Appx-D-201	All
Updated the Generating the IP-XACT description	*G.3 Generating the IP-XACT description* on page Appx-G-252	All