ARM® Cortex®-M23 Processor

Revision: r1p0

Integration and Implementation Manual

Confidential

ARM®

ARM DIT 0062C (ID111716)

ARM Cortex-M23 Processor
Integration and Implementation Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
25 April 2016	A	Confidential	First release for r0p0
29 July 2016	B	Confidential	First release for r1p0
18 November 2016	C	Confidential	Second release for r1p0

Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

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-M23 Processor Integration and Implementation Manual

Preface

This preface introduces the *ARM® Cortex®-M23 Processor Integration and Implementation Manual (IIM)*. It contains the following sections:

About this book

This book is for the Cortex-M23 processor.

Product revision status

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

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

Intended audience

This book is written for experienced hardware and *System-on-Chip* (SoC) engineers who might or might not have experience with ARM products. Such engineers typically have experience of writing Verilog and of performing synthesis, but might have limited experience of integrating and implementing ARM products.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for a description of the processor design platforms and tools, including the supported design flow, directory structure, and design hierarchy.

Chapter 2 *Configuration Guidelines*

Read this for a description of the configuration options that you must be aware of before implementing and integrating the processor into your design.

Chapter 3 *Key Integration Points*

Read this for a description of the key points that you must consider when you integrate the processor with your SoC design.

Chapter 4 *Functional Integration Guidelines*

Read this for guidelines on how to perform functional integration of the processor.

Chapter 5 *Key Implementation Points*

Read this for a description of the key points that you must consider when you implement the processor.

Chapter 6 *Floorplan Guidelines*

Read this for a description of floorplanning guidelines.

Chapter 7 *Design For Test*

Read this for information about the design for test features.

Chapter 8 *Execution Testbench*

Read this for a description of the Cortex-M23 processor execution testbench.

Chapter 9 *Netlist Dynamic Verification*

Read this for information about how to perform netlist dynamic verification on the processor.

Chapter 10 *DSM Generation*

Read this for a description of the *Design Simulation Model* (DSM).

Chapter 11 *Sign-off*

Read this for a description of the ARM verification criteria, and how to sign off your design.

Appendix A *GPIO Programmers Model*

Read this for a description of the *General Purpose Input/Output* (GPIO) programmers model.

Appendix B *GRBIKMCU GPIO Integration*

Read this for a description of the example *Microcontroller Unit* (MCU) system supplied with the execution testbench.

Appendix C *CoreSight SoC*

Read this for the location of the CoreSight SoC-400 configuration files.

Appendix D *SysTick Examples*

Read this for examples of how to set up the SysTick timer during integration.

Appendix E *Debug Driver*

Read this for a description of the debug driver supplied with the integration kit.

Appendix F *Power Intent*

Read this for a description of the optional *State Retention and Power Gating* (SRPG) features of the processor.

Appendix G *CTI*

Read this for a description of the Cortex-M23 processor *Cross Trigger Interface* (CTI).

Appendix H *Signal Timing Constraints*

Read this for a description of the signal timing constraints that must be applied when you implement the processor.

Appendix I *IP-XACT*

Read this for a description of how you can configure an IP-XACT description of the processor.

Appendix J *Revisions*

Read this for a list of the technical changes between released issues of this book.

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 *ARM® Glossary*, http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html.

Conventions

This book uses the conventions that are described in:

- *Typographical conventions* on page x.

- *Timing diagrams*.
- *Signals* on page xi.

Typographical conventions

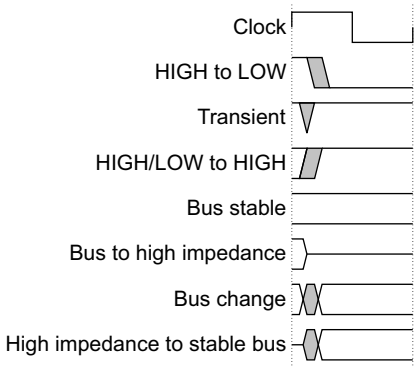
The following table describes the typographical conventions:

Style	Purpose
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.

Timing diagrams

The figure named *Key to timing diagram conventions* 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.



Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in *Key to timing diagram conventions*. If a timing diagram shows a single-bit signal in this way then its value does not affect the accompanying description.

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.

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

Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, http://infocenter.arm.com, for access to ARM documentation.

See the Cortex-M23 processor Release Note for details of EDA tool vendor documents.

ARM publications

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

- *ARM® AMBA® 5 AHB Protocol Specification, AHB5, AHB-Lite* (ARM IHI 0033).
- *ARM® CoreSight™ Components Technical Reference Manual* (ARM DDI 0314).
- *ARMv8-M Architecture Reference Manual* (ARM DDI 0553).
- *ARM® Cortex®-M23 Processor Technical Reference Manual* (ARM DDI 0550).
- *ARM® CoreSight™ MTB-M23 Technical Reference Manual* (ARM DDI 0564).
- *ARM® CoreSight™ ETM-M23 Technical Reference Manual* (ARM DDI 0563).
- *ARM® Cortex®-M23 Processor Release Note* (AT621-DC-06003)
- *ARM®-Cortex®-M23 Processor Cadence Reference Methodology Release Note* (AT590-RM-70000).
- *ARM® CoreSight™ Architecture Specification* (ARM IHI 0029).
- *ARM® CoreSight™ SoC-400 User Guide* (ARM DUI 0563).
- *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2* (ARM IHI 0031B).
- *ARM® AMBA® Designer ADR-400 User Guide* (ARM DUI 0333P).
- *Low Power Interface Specification*, *ARM® Q-Channel and P-Channel Interfaces* (ARM IHI 0068).
- *CoreSight™ Technology System Design Guide* (ARM DGI 0012).

Other publications

This section lists relevant documents published by third parties:

- *IEEE 1149.1-2001 IEEE Standard Test Access Port and Boundary Scan Architecture (JTAG)*.

Feedback

ARM welcomes feedback on this product and its documentation.

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.
- The number, ARM DIT 0062C.
- The page numbers to which your comments apply.
- 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 gives an overview of the process of integrating and implementing the Cortex-M23 processor. It contains the following sections:

1.1 About the processor

The Cortex-M23 processor is an energy-efficient processor with a very low gate count. It is intended to be used for microcontroller and deeply embedded applications that require an area-optimized processor. The Cortex-M23 processor supports *State Retention and Power Gating* (SRPG) with up to three power domains to enable very energy efficient silicon implementation and a trace interface supporting the optional CoreSight *Micro Trace Buffer* (MTB) or *Embedded Trace Macrocell* (ETM).

The Cortex-M23 processor product includes an optimized *Debug Access Port* (DAP) component that supports Serial Wire, Serial Wire Multi-Drop, and JTAG protocols for connection to off-chip debuggers. It also includes an optimized *Cross Trigger Interface* (CTI) component to allow debug cross-triggering with other devices in System-on-Chip implementations.

The Cortex-M23 processor supports the following levels of hierarchy for integration or implementation:

- `GREBE` - Processor component level.
- `GREBEINTEGRATION` - Processor, *Wakeup Interrupt Controller* (WIC), Q-Channel, either *Micro Trace Buffer* (MTB) or *Embedded Trace Macrocell* (ETM), and *Cross Trigger Interface* (CTI).
- `GREBEINTEGRATION_MCU` - `GREBEINTEGRATION`, DAP, *Trace Port Interface Unit (TPIU),* MCU_ROM table, and the APB interconnect.

  Note

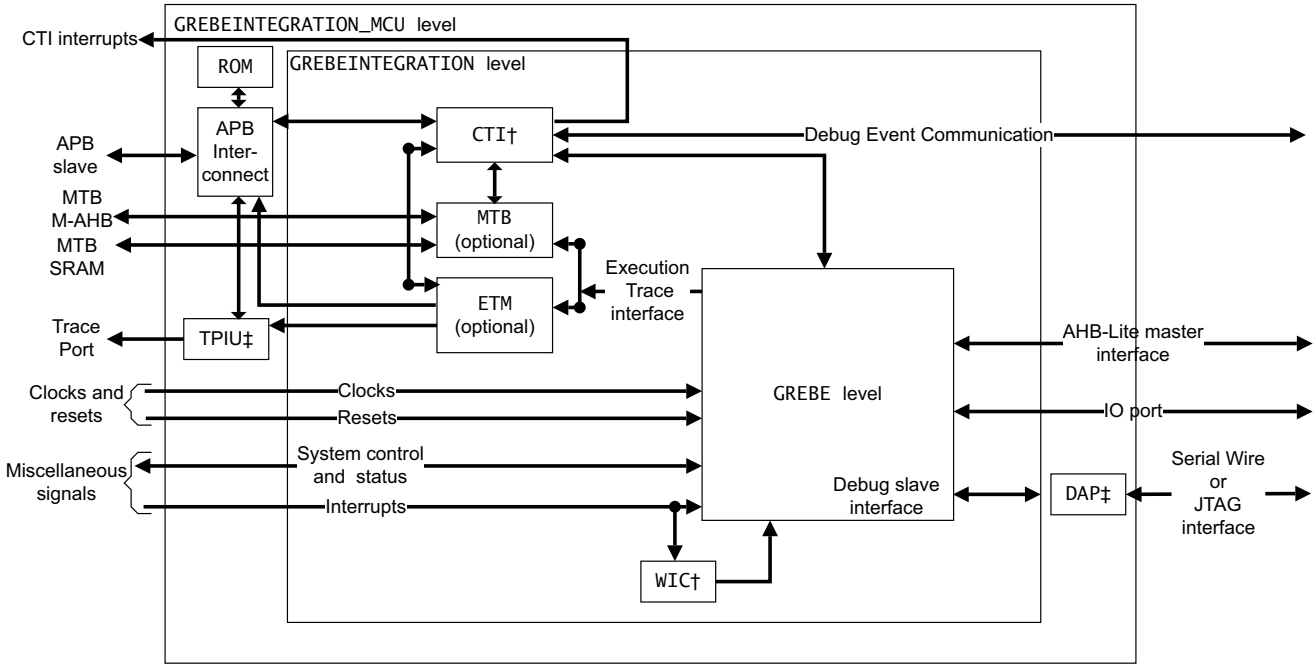
  — The TPIU can only be present if the ETM is present.

  — The MCU_ROM table can only be present if the TPIU is present.

Figure 1-1 on page 1-3 shows the integration levels and the main interfaces of the processor.

Note

You can modify the `GREBEINTEGRATION` and `GREBEINTEGRATION_MCU` levels for your own requirements.



†Only present if GREBEINTEGRATION is configured to include it
‡Only present if GREBEINTEGRATION_MCU is configured to include it

Figure 1-1 Module hierarchy and the main interfaces

GREBE component level

Contains the basic processor. This level is not modifiable.

GREBEINTEGRATION level

Instantiates the GREBE component level and provides example integration with either the CoreSight MTB-M23 or the CoreSight ETM-M23. The CoreSight MTB-M23 and the CoreSight ETM-M23 are licensed and delivered separately from the Cortex-M23 processor. It also provides an example interface with the optional Cross Trigger Interface and WIC. This level is provided as a working example of a single-processor subsystem. You can modify it to suit your requirements.

GREBEINTEGRATION_MCU level

Instantiates the GREBEINTEGRATION component level and includes the DAP, TPIU, MCU_ROM, and ABP interconnect.

The Cortex-M23 processor is highly configurable. Although the top-level signals at the GREBEINTEGRATION_MCU, GREBEINTEGRATION, and GREBE levels are independent of configuration, the functionality of these interfaces depends on the configuration. For example, the debug interfaces at each level are non-functional if your configuration does not include debug. For more details about configuring the GREBEINTEGRATION_MCU, GREBEINTEGRATION, or GREBE component levels, see Chapter 2 *Configuration Guidelines*.

Note

- The Cortex-M23 product includes some example user modifiable Verilog RTL files, including the GREBEINTEGRATION_MCU.v and GREBEINTEGRATION.v files that are described in this section. Files within the grebe, grb_dap, grb_cti, and grb_tpiu directories must not be modified.

- If you also license the CoreSight MTB-M23, files within the `grb_mtb` directory must not be modified.
- If you also license the CoreSight ETM-M23, files within the `grb_etm` directory must not be modified.

1.2 About integration and implementation

The flows that you can use to integrate a Cortex-M23 processor into your SoC can vary depending on the capacity of your tools.

Figure 1-2 shows the integration and implementation flow when you first implement the Cortex-M23 processor and then integrate the implemented processor into your system.

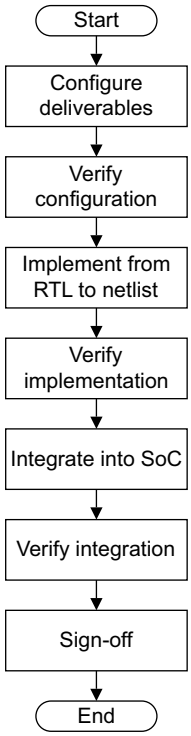


Figure 1-2 Implementation and integration flow

Figure 1-3 on page 1-6 shows the integration and implementation flow when you first integrate the Cortex-M23 processor into your system and then implement your system.

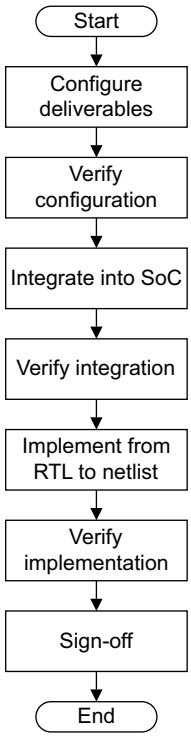


Figure 1-3 Integration and implementation flow

1.2.1 Integration

Integration is the first step in this 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.
- Verifying the processor within the SoC design.

Various implications arise for your design, depending on the elements that you include. You must consider:

- Interfaces, especially the treatment of unused signals.
- Verification.

Although you might have extra elements in your design, the main connection is the AHB-Lite bus that conforms to the AMBA® 5 AHB Protocol.

1.2.2 Implementation

Figure 1-4 on page 1-7 shows the implementation process, including the top-level inputs, resources, outputs, and controls and constraints for implementation.

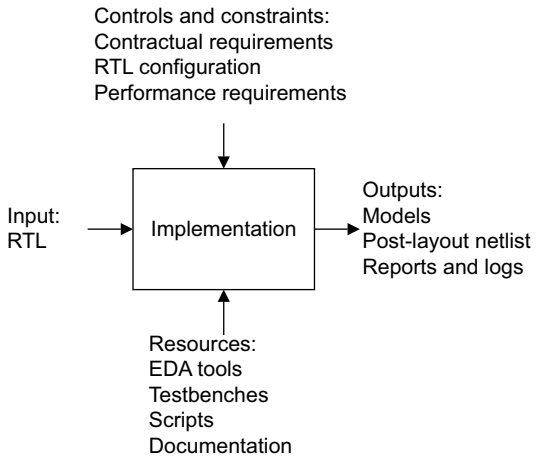


Figure 1-4 Implementation process

For an overview of the implementation process, see:

- *Implementation resources*.
- *Implementation controls, constraints, and obligations*.
- *Implementation inputs* on page 1-8.
- *Implementation flow* on page 1-8.
- *Implementation outputs* on page 1-8.

For information on the deliverables, see:

- *Reference data* on page 1-11.

Implementation resources

This guide assumes that you have suitable EDA tools and compute resources for implementation. See the *ARM® Cortex-®M23 Processor Release Note* for a list of deliverables and any specific tool revisions that are required for implementation.

Note

The *ARM® Cortex®-M23 Processor Release Note* describes any special requirements that might affect the flow, such as details of any special tool requirements that enable optional flows within the implementation.

Implementation controls, constraints, and obligations

Figure 1-4 shows the general controls and constraints that apply to the implementation.

You must implement the device in accordance with your contract. The details set out in this implementation guide are designed to help you implement the RTL into products. 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 licenses 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 regarding any third party's products or services is not an express or implied approval or endorsement of the use thereof.

Implementation inputs

The *ARM® Cortex®-M23 Processor Release Note* describes 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.
  - GDSII.
  - *Standard Delay Format* (SDF).
- Test:
  - *Automatic Test Pattern Generation* (ATPG) vectors.

Implementation flow

Figure 1-5 on page 1-9 shows the implementation flow.

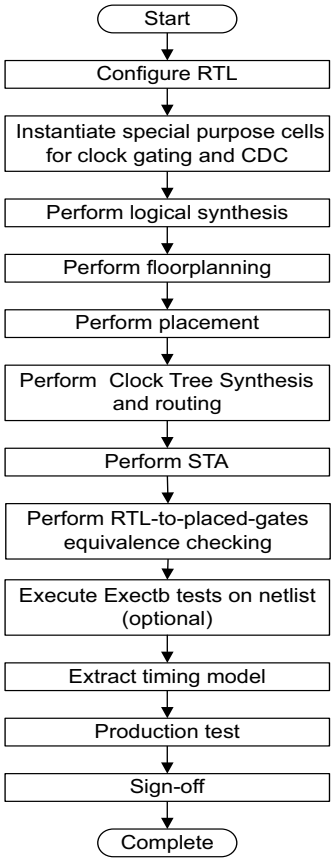


Figure 1-5 Implementation flow

Note

For information on contractual obligations to complete sign-off as part of the completed flow, see *Implementation controls, constraints, and obligations* on page 1-7.

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, see Chapter 11 *Sign-off* for more information.

1.4 Reference data

Before starting, you must ensure that the unpacked deliverables are located in the correct directory structure. Figure 1-6 on page 1-12 shows the principal directory structure when you unpack the deliverables.

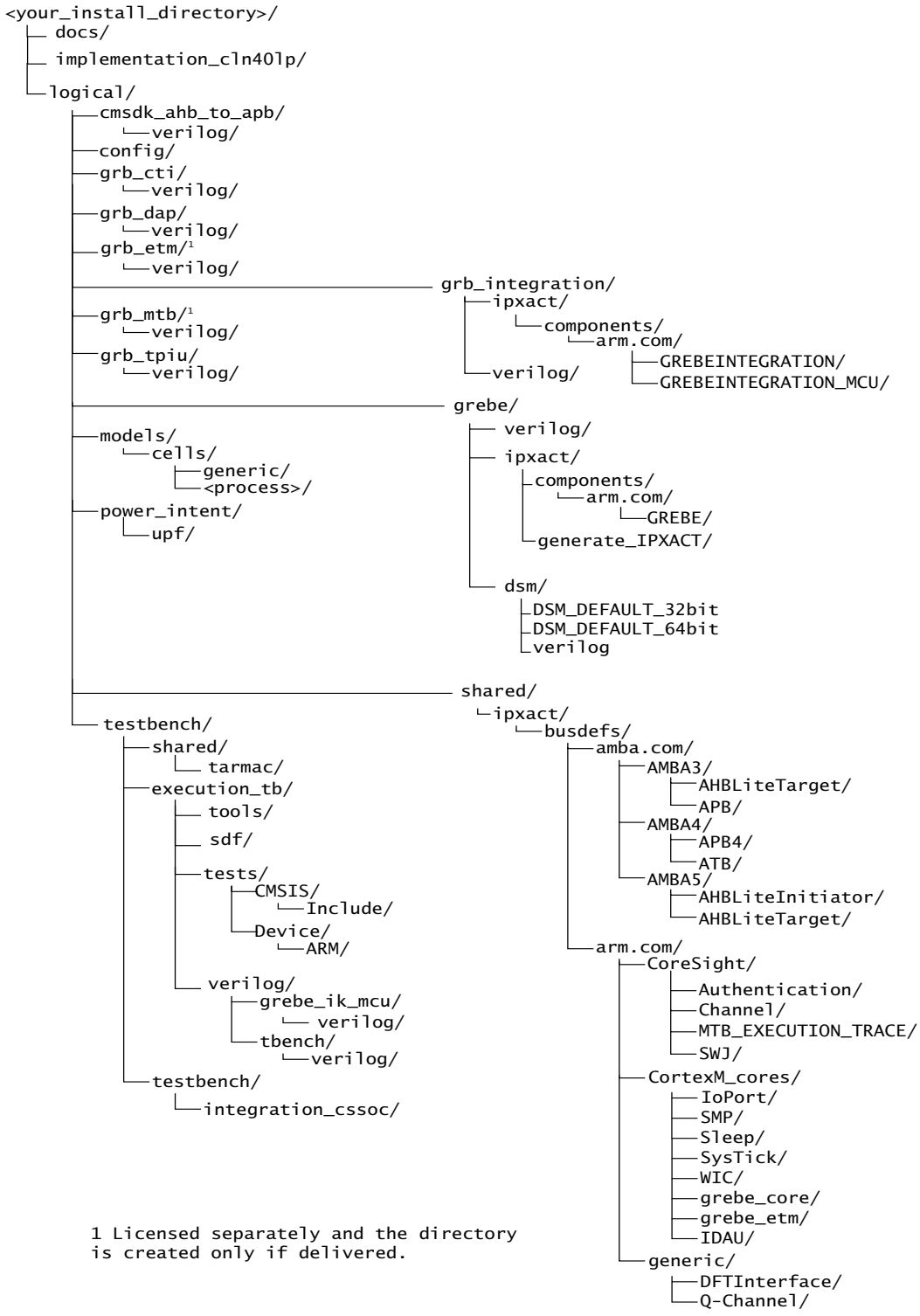


Figure 1-6 Directory structure

Chapter 2
Configuration Guidelines

This chapter describes the guidelines for RTL configuration. These enable you to tailor the RTL to the specific requirements of the target application. It contains the following sections:

2.1 About configuration guidelines

Caution

For successful configuration of the RTL, you must set up the configurable options. Failure to complete all the necessary configuration can result in malfunction.

The Cortex-M23 processor RTL includes the following configurable integration levels:

- The `GREBE` level implements the Cortex-M23 processor, the *Nested Vectored Interrupt Controller* (NVIC), breakpoint unit, watchpoint unit, and associated debug control.
- The example `GREBEINTEGRATION` level instantiates a `GREBE` level.
  - Either an optional CoreSight MTB-M23 or an optional CoreSight ETM-M23.
  - An optional Cortex-M23 processor CTI.
- The example `GREBEINTEGRATION_MCU` level instantiates the `GREBEINTEGRATION` level pre-integrated with the Cortex-M23 processor Serial Wire or JTAG DAP. There is also TPIU and an MCU_ROM table at this level, which is only relevant if the ETM is present.

  Support files are provided to enable integration of this *Processor Integration Layer* (PIL) with other ARM PILs using the CoreSight SoC product, available separately from ARM.

If you are building a Cortex-M23 subsystem as part of a larger System on Chip (SoC) design, ARM recommends that you base your design on the example `GREBEINTEGRATION` level. If you are building a single-processor design based on Cortex-M23 that requires the debug features of the processor, ARM recommends that you base your design on `GREBEINTEGRATION_MCU` level. If you are building a single processor system, or any system that does not require the debug features of the processor, you can base your design on the `GREBE` level.

The configurable options available depend on the level of hierarchy you choose to implement.

At all levels, parameters control the configurable options.

2.2 Configuring the RTL script

To configure the RTL script, you must set the configuration options.

To set your chosen configuration options:

1. Change your working directory to logical/config/.
2. Edit GREBE.conf to match your required configuration.
3. Run configure_rtl.pl to configure the design and generate the read-only file grebe_defs.v. The configuration script generates a message stating whether the configuration file has been parsed successfully or not, and outputs a summary of the user-defined values. The script that generates the configured design is a PERL script. You must ensure that PERL is installed in the execution path.

Since the configure_rtl.pl file generates grebe_defs.v, IP-XACT files, and configuration files for the synthesis ARM recommends running the configure_rtl.pl file in order to change the configuration, rather than forcing the parameters at GREBE, GREBEINTEGRATION or GREBEINTEGRATION_MCU instantiation.

Caution

The configuration process generates grebe_defs.v. You must not modify it because:

- The configuration process is self-checking and prevents the production of incorrect configurations.
- The configuration process duplicates and modifies design files to match the configuration you specified. The design files are not valid if you modify grebe_defs.v.

2.3 Configuration options

The following sections describe the Cortex-M23 processor configuration options:

- *GREBE configuration options*.
- *GREBEINTEGRATION configuration options* on page 2-7.
- *GREBEINTEGRATION_MCU configuration options* on page 2-9.

2.3.1 GREBE configuration options

Table 2-1 shows the configuration options summary for GREBE.

Table 2-1 GREBE options summary

Parameter	Default value	Supported values	Description
ACG	YES	NO, YES	Specifies if internal architectural clock gates are included to minimize dynamic power dissipation: NO Exclude architectural clock gates. YES Include architectural clock gates. **Note** If you include architectural clock gating, you must replace the provided model, grb_acg.v, with a direct instantiation of a clock gating cell from your target cell library.
BE	NO	NO, YES	Specifies the endianness for data transfers: NO Little-endian. YES Byte-invariant big-endian. **Note** Instruction fetches and PPB accesses are always little-endian.
BKPT	4	0-4	Specifies the number of breakpoint unit comparators implemented. **Note** If you exclude debug, this parameter has no effect. See the DBG parameter description in this table.
DBG	YES	NO, YES	Specifies whether the debug extensions are implemented: NO Exclude debug functionality. YES Include debug functional. The configuration of debug is also controlled by the BKPT and WPT parameters. **Note** Setting the DBG parameter to NO can result in the **DCLK** input becoming unloaded.
HWF	NO	NO, YES	Halfword fetching only: **NO** Fetch instructions using 32-bit AHB-Lite accesses whenever possible. **YES** Fetch instructions using only 16-bit AHB-Lite accesses.
IOP	NO	NO, YES	I/O port: NO Exclude I/O port functionality. YES Include I/O port functionality.

Table 2-1 GREBE options summary (continued)

Parameter	Default value	Supported values	Description
IRQDIS_n	0	0, 1 for each bit.	IRQ line disable mask. Bit m of this 32-bit parameter disables IRQ[32n+m] and WICLINES[32n+m+2]. For each bit **0** IRQ enabled. **1** IRQ disabled. **Note** Bits n>=NUMIRQ are ignored.
MPU_NS	8	0, 4, 8, 12, 16	Specifies the number of implemented *Non-secure Memory Protection Unit* (MPU_NS) regions: 0 Exclude MPU_NS functionality. 4 Include MPU_NS functionality (4 MPU_NS regions). 8 Include MPU_NS functionality (8 MPU_NS regions). 12 Include MPU_NS functionality (12 MPU_NS regions). 16 Include MPU_NS functionality (16 MPU_NS regions).
MPU_S	8	0, 4, 8, 12, 16	Specifies the number of implemented *Secure Memory Protection Unit* (MPU_S) regions: 0 Exclude MPU_S functionality. 4 Include MPU_S functionality (4 MPU_S regions). 8 Include MPU_S functionality (8 MPU_S regions). 12 Include MPU_S functionality (12 MPU_S regions). 16 Include MPU_S functionality (16 MPU_S regions). **Note** If you exclude Security Extension, the Secure MPU is also excluded.
NUMIRQ	32	0-240	Specifies the highest interrupt number (NUMIRQ-1) of implemented user interrupts: 0 No functional IRQ lines 1 **IRQ[0]**. 2 **IRQ[1:0]**. **...** **240** **IRQ[239:0]**. NUMIRQ should be used with IRQDIS_n.
RAR	NO	NO, YES	Specifies whether all synchronous states or only architecturally required states are reset: NO Only architecturally required state is reset. YES All state is reset. **Note** • When RAR is YES, all registers in the design can be reset, incurring an area penalty. • When RAR is NO, the registers in the design that do not require a reset have no reset.

Table 2-1 GREBE options summary (continued)

Parameter	Default value	Supported values	Description
SAU	4	0, 4, 8	Specifies the number of implemented *Security Attribute Unit* (SAU) regions: 0 Exclude SAU functionality. Only SAU control is implemented. 4 Include SAU functionality (4 SAU regions). 8 Include SAU functionality (8 SAU regions). **Note** • If SAU is set to 0, an external *Implementation Defined Attribution Unit* (IDAU) interface can specify the memory regions. • If you exclude Security Extension, the SAU is also excluded.
SDIV	NO	NO, YES	Specifies the implemented divider: NO Include the fast, 17-cycle divider. YES Include the small, 34-cycle divider.
SECEXT	YES	NO, YES	Specifies whether the ARMv8-M security extensions are included: NO ARMv8-M security extensions are not included. YES ARMv8-M security extensions are included.
SMUL	NO	NO, YES	Specifies the implemented multiplier: NO Include the fast, single-cycle multiplier. YES Include the small, 32-cycle multiplier.
SYST	2	0, 1, 2	Specifies whether the SysTick timer functionality is included: 0 Exclude the SysTick timer. 1 Include the Secure SysTick timer if the Security Extension is included or include the Non-secure SysTick timer if the Security Extension is not included. 2 Include the Secure and Non-secure SysTick timers if the Security Extension is included. This value is not valid if the Security Extension is not included.
VTOR	YES	NO, YES	Vector Table Offset Register: NO Exclude VTOR. YES Include VTOR. **Note** If you include Security Extension, then two VTORs are implemented, one for the Secure and one for the Non-secure state.
WPT	2	0-4	Specifies the number of watchpoint unit comparators implemented. **Note** If you exclude debug, this parameter has no effect. See the DBG parameter description in this table.

2.3.2 GREBEINTEGRATION configuration options

GREBEINTEGRATION instantiates the GREBE level and some additional components. See Figure 1-1 on page 1-3. It includes configuration options available at the GREBE level and new configuration options that control behavior at this level. Table 2-2 shows the new configuration options available at this level, and configuration options that have additional behavior at this level.

Table 2-2 GREBEINTEGRATION options summary

Parameter	Default value	Supported values	Description
ACG	YES	NO, YES	At this level, additionally specifies if an internal architectural clock gate, which is connected to the CoreSight MTB-M23, is included to minimize dynamic power dissipation: NO Exclude architectural clock gate. YES Include architectural clock gate. **Note** If you include architectural clock gating, you must replace the provided model, grb_acg.v, with a direct instantiation of a clock gating cell from your target cell library.
CTI	YES	NO, YES	Specifies whether the CTI is included: NO Exclude the CTI. YES Include the CTI. **Note** If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.
ETM	NO	NO, YES	Specifies whether the ETM is included: NO Exclude the ETM. YES Include the ETM. **Note** Exclusive with MTB present. If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.
MTB	NO	NO, YES	Specifies whether the CoreSight MTB-M23 is included: NO Exclude the CoreSight MTB-M23. YES Include the CoreSight MTB-M23. **Note** Exclusive with ETM present. If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.
MTBAWIDTH	12	5-32	Address width of MTB SRAM.
WIC	YES	NO, YES	Specifies whether the WIC interface is implemented: NO Exclude the WIC interface. YES Include the WIC interface.

Table 2-2 GREBEINTEGRATION options summary (continued)

Parameter	Default value	Supported values	Description
WICLINES	34	2-242	Specifies the lines supported by the WIC interface: 2 Only **NMI** and **RXEV** are supported. 3 **NMI**, **RXEV**, and **IRQ[0]** are supported. 4 **NMI**, **RXEV**, and I**RQ[1:0]** are supported. ... 242 **NMI**, **RXEV**, and **IRQ[239:0]** are supported. **Note** If you exclude the WIC interface, this parameter has no effect. See the WIC parameter description in this table.

2.3.3 GREBEINTEGRATION_MCU configuration options

GREBEINTEGRATION_MCU instantiates the GREBEINTEGRATION level and some additional components. See Figure 1-1 on page 1-3. It includes configuration options available at the GREBE level and new configuration options that control behavior at this level. Table 2-3 shows the new configuration options available at this level, and configuration options that have additional behavior at this level.

Table 2-3 GREBEINTEGRATION_MCU options summary

Parameter	Default value	Supported values	Description
AHBSLV	YES	NO, YES	Specifies the bus protocol implemented on the SLV port. This is a debug port. See Chapter 4 *Functional Integration Guidelines* for additional information: NO The SLV port implements a Cortex-M23 processor DAP-specific protocol. YES The SLV port implements a subset of AHB. ARM recommends that you set this parameter to YES if you implement the GREBE level for integration with any debug infrastructure other than the Cortex-M23 processor DAP, for example, the generic CoreSight DAP.
BASEADDR	0xE00FF003	-	Specifies the value of the DAP MEM-AP base register that is read by debug tools to locate the first CoreSight component in the system. The default value enables debug tools to identify only the Cortex-M23 processor. 0xE00FE003 Set to this value if ETM is set to YES, to point to the MCU ROM table, listing the TPIU. 0xE00FF003 Set to this value if ETM is set to NO, to point to the processor ROM table. If you implement any additional CoreSight components or want debug tools to be able to uniquely identify your system, you must override this value. See *CoreSight ROM tables* on page 4-50. **Note** If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.
HALTEV	NO	NO, YES	Specifies if the DAP includes halt event signaling support: NO Exclude halt event signaling support. YES Include halt event signaling support. If you include this feature, the Debug Port (DP) Event Status Register is implemented in the Cortex-M23 processor DAP. The Event Status Register enables a debugger to determine whether the processor is in debug HALT state without needing to read the processor DHCSR.S_HALT register. This enables you to save power on some implementations when using a debugger that uses the Event Status register. **Note** If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.

Table 2-3 GREBEINTEGRATION_MCU options summary (continued)

Parameter	Default value	Supported values	Description
JTAGnSW	NO	NO, YES	Specifies the external debug protocol implemented by the Cortex-M23 processor DAP: NO Implement Serial Wire. YES Implement JTAG. **Note** • If you implement the JTAG protocol, the Cortex-M23 processor DAP returns the DPIDR value 0x0BF11477 unless modified by an *Engineering Change Order* (ECO). The JTAG IDCODE scan chain returns 0x0BA05477. • If you implement the Serial Wire protocol, the Cortex-M23 processor DAP returns a DPIDR value that depends the SWMD parameter value. See SWMD in this section. • If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.
SWMD	NO	NO, YES	DAP Serial Wire Multi-drop Support: NO Implement Serial Wire protocol version 1. This enables a debugger to connect to a system with a single Serial Wire device. YES Implement Serial Wire protocol version 2, with target selection that supports a system with: • Multiple Serial Wire devices on the same interface. • A dormant state that enables multiple devices, with different protocols such as JTAG, to share signals from the same interface. See the *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2*. **Note** • If you implement Serial Wire protocol version 1 (SWMD = 0), the Cortex-M23 processor DAP has a DPIDR value of 0x0BF11477 unless modified by an ECO. • If you implement Serial Wire protocol version 2 (SWMD = 1), the Cortex-M23 processor DAP has a DPIDR value of 0x0BF12477 unless modified by an ECO. • If you implement the JTAG protocol (JTAGnSW = 1), this parameter has no effect. • If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.

Table 2-3 GREBEINTEGRATION_MCU options summary (continued)

Parameter	Default value	Supported values	Description
TARGETID	0x00000001	-	This parameter defines the contents of the Target Identification register in the Cortex-M23 processor DAP that uniquely identifies the system that the Cortex-M23 processor DAP is connected to. The parameter contains the following fields: [31:28] TREVISION. [27:12] TPARTNO. [11:1] TDESIGNER. [0] 1. The debugger uses the TPARTNO and TDESIGNER fields within this register, and the Target Instance field in the Data Link Protocol Identification register, to connect to a specific instance of the Cortex-M23 processor DAP in a system containing more than one Serial Wire multi-drop device. See the *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2* for more information. TDESIGNER must be initialized to your JEDEC-assigned 11-bit code, formed from the 4-bit count of the number of JEP-106 continuation codes and the 7-bit JEP-106 identity code assigned to you by JEDEC: TDESIGNER = {count[3:0], identity[6:0]}. **Note** • This is only used when multi-drop Serial Wire Debug is implemented. • If you implement Serial Wire protocol version 1 (SWMD = 0), this parameter has no effect. • If you implement the JTAG protocol (JTAGnSW = 1), this parameter has no effect. • If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.

Chapter 3
Key Integration Points

This chapter describes the key integration points that you must consider when you integrate the processor into your SoC design. It contains the following sections:

3.1 About key integration points

This chapter contains a list of the main points to consider when you integrate the Cortex-M23 processor into your SoC design. You must read this chapter with the rest of the information in this book, and the information referred to in *Additional reading* on page xi.

Although you can use this chapter to check that you have covered the integration steps described in the other chapters, you must complete all the integration steps described in the later chapters.

3.2 Key integration tasks

Table 3-1 lists the GREBE component level key integration tasks.

Table 3-1 GREBE component level key integration tasks

Key task		Description
1.	Connect the **SCLK**, **HCLK**, and **DCLK**[a] clocks correctly.	See *Clocks* on page 4-3
2.	Connect the **nHRESET**, **nDBGRESET**, and **nSYSPORESET**[a] resets correctly.	See *Resets* on page 4-7
3.	Tie off or connect the following interface inputs appropriately: • External AHB-Lite interface. • AHB interface extensions. • I/O port. • Interrupt interface. • Debug slave interface. • Miscellaneous signals. • SysTick signals. • ETM interface.[b]	See *Interfaces* on page 4-10
4.	Tie off or connect the CTI interface inputs appropriately:	See Appendix G *CTI*
5.	Tie off or connect the CoreSight MTB-M23 interface inputs appropriately:	
6.	Verify your design using the execution testbench.	See Chapter 8 *Execution Testbench*

a. Tie LOW if your Cortex-M23 processor configuration does not include debug.
b. This is only applicable if you have included the ETM interface.

Table 3-2 lists the GREBEINTEGRATION level key integration tasks.

Table 3-2 GREBEINTEGRATION level key integration tasks

Key task		Description
1.	Connect the **FCLK**, **SCLK**, **HCLK**, and **DCLK**[a], clocks correctly.	See *Clocks* on page 4-3
2.	Connect the **nPORESET**, **nSYSPORESET**, **nHRESET**, and **nDBGRESET**[a] resets correctly.	See *Resets* on page 4-7
3.	Tie off or connect the following interface inputs appropriately: • External AHB-Lite interface. • AHB-Lite interface extensions. • I/O port. • Interrupt interface. • External debugger interface. • Miscellaneous signals. • SysTick signals. • *Power Management Unit* (PMU) interface. • WIC. • ETM interface.[b]	See *Interfaces* on page 4-10
4.	Tie off or connect the CTI interface inputs appropriately:	See Appendix G *CTI*
5.	Tie off or connect the CoreSight MTB-M23 interface inputs appropriately:	
6.	Verify your design using the execution testbench.	See Chapter 8 *Execution Testbench*

a. Tie LOW if your Cortex-M23 processor configuration does not include debug.
b. This is only applicable if you have included the ETM interface.

Table 3-3 lists the GREBEINTEGRATION_MCU level key integration tasks.

Table 3-3 GREBEINTEGRATION_MCU level key integration tasks

Key task		Description
1.	Connect the **FCLK**, **SCLK**, **HCLK**, **DCLK**[a], **TRACECLKIN**, and **SWCLKTCK** clocks correctly.	See *Clocks* on page 4-3
2.	Connect the **nPORESET**, **nHRESET**, **nDBGRESET**[a],**nTRST**, and **nTRESET** resets correctly.	See *Resets* on page 4-7
3.	Tie off or connect the following interface inputs appropriately: • External AHB-Lite interface. • AHB interface extensions. • I/O port. • Interrupt interface. • External debugger interface. • Miscellaneous signals. • SysTick signals. • *Power Management Unit* (PMU) interface. • WIC.	See *Interfaces* on page 4-10
4.	Tie off or connect the CTI interface inputs appropriately:	See Appendix G *CTI*
5.	Tie off or connect the CoreSight MTB-M23 interface inputs appropriately, including the MTBSRAM interface:	See the *ARM® CoreSight™ MTB-M23 Technical Reference Manual*
6.	Tie off or connect the TPIU and ETM interfaces appropriately:	
7.	Tie off or connect the DAP interface inputs appropriately:	
8.	Tie off or connect the APB interface components appropriately:	
9.	Tie off the CoreSight ROM table base address.[b]	See Chapter 2 *Configuration Guidelines*
10.	Verify your design. If you have licensed the ARM CoreSight SoC product, you can use it to produce integration tests.	-

a. Tie LOW if your Cortex-M23 processor configuration does not include debug.
b. This is only applicable if you have configured the Cortex-M23 processor processor to include debug.

Chapter 4
Functional Integration Guidelines

This chapter describes the guidelines for functional integration of the processor into your SoC design. It contains the following sections:

4.1 About functional integration

This chapter contains information that you must consider before or during the integration of the processor into your SoC design.

4.2 Clocks

This section provides information on how to use the Cortex-M23 processor clocks in your SoC design in:

- *GREBE level clocks* on page 4-4.
- *GREBEINTEGRATION level clocks* on page 4-4.
- *GREBEINTEGRATION_MCU level clocks* on page 4-5
- *Clock gating combinations* on page 4-6.

Figure 4-1 shows the Cortex-M23 processor clock domains.

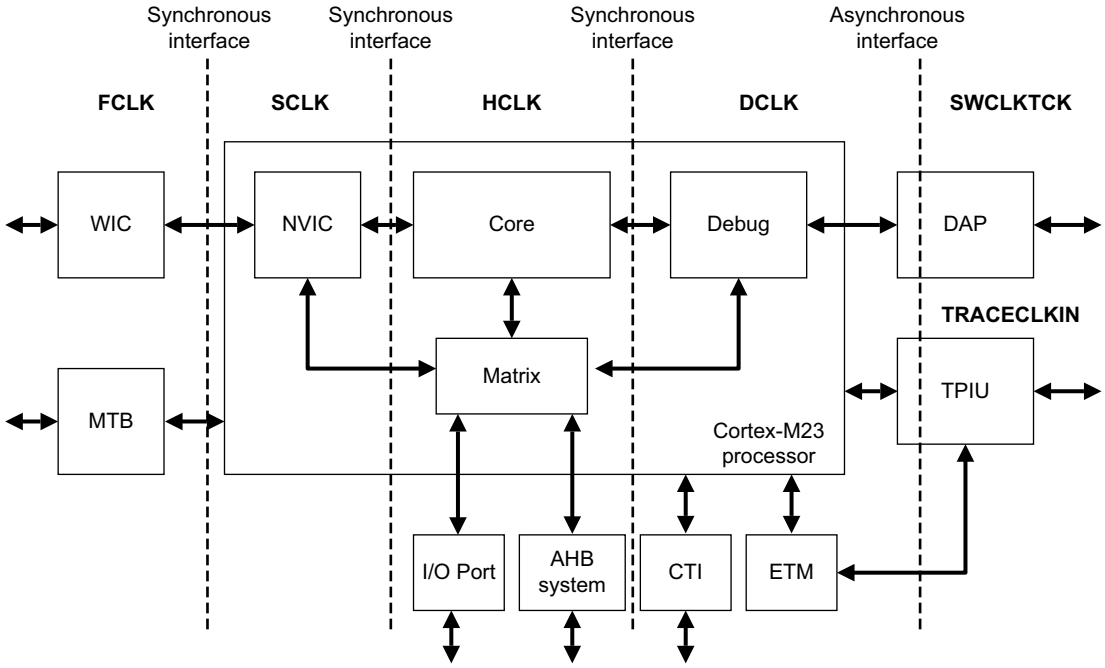


Figure 4-1 Cortex-M23 processor clock domains

4.2.1 GREBE level clocks

Table 4-1 shows the clocks at the GREBE level of hierarchy.

Table 4-1 GREBE level clocks

Name	Direction	Description	Connection information
SCLK	Input	Free running clock that clocks a small amount of logic in the processor system domain.	**SCLK** must always be running unless the processor is in WIC-mode deep sleep and no debugger is connected. When the processor is in deep sleep, you can gate the other clocks and run **SCLK** at reduced frequency.
HCLK	Input	Clock for the majority of the non-debug logic in the processor system domain.	**HCLK** must be derived directly from **SCLK**. Connect **HCLK** to the AHB layer that the processor is connected to. **HCLK** can be gated when the processor is sleeping and no debugger is connected. You can use **SLEEPING** to determine when the processor is sleeping.
DCLK	Input	Clock for the processor debug domain	**DCLK** must be derived directly from **SCLK**. **DCLK** must always be driven while a debugger is connected. It can be gated when no debugger is connected. You can use the **CDBGPWRUPACK** output from your system PMU to detect the presence of an external debugger requesting a connection.

Note

- Although under certain conditions, **HCLK** and **DCLK** can be gated versions of **SCLK** to minimize dynamic power dissipation, when active, **HCLK**, **DCLK**, and **SCLK** must all be synchronous and balanced to a ratio of 1:1.
- When a debugger is connected, **HCLK**, **DCLK**, and **SCLK** must all be running.
- **DCLK** must be tied LOW if the Cortex-M23 processor is configured without debug.

4.2.2 GREBEINTEGRATION level clocks

Table 4-2 shows the clocks of the GREBEINTEGRATION level of hierarchy.

Table 4-2 GREBEINTEGRATION level clocks

Name	Direction	Description	Connection information
FCLK	Input	Free running clock that clocks the WIC and Q-Channels.	**FCLK** must always be running.

Table 4-2 GREBEINTEGRATION level clocks (continued)

Name	Direction	Description	Connection information
SCLK	Input	System domain clock. This clocks a small amount of logic in the processor system domain.	**SCLK** must be derived directly from **FCLK**. It must always be running unless the processor is in WIC-mode deep sleep and no debugger is connected.
HCLK	Input	Clock for the majority of the non-debug logic in the processor system domain.	**HCLK** must be derived directly from **FCLK**. Connect **HCLK** to the AHB layer that the processor is connected to. **HCLK** can be gated when the processor is sleeping and no debugger is connected. You can use **GATEHCLK** to determine when you can gate **HCLK**.
DCLK	Input	Clock for the processor debug domain.	**DCLK** must be derived directly from **FCLK**. **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 `GREBEINTEGRATION` level to detect the presence of an external debugger requesting a connection.

The example PMU module provided, `grb_ik_pmu.v`, meets the clocking requirements at the `GREBEINTEGRATION` level.

Note

- Although under certain conditions, **HCLK**, **SCLK**, and **DCLK** can be gated versions of **FCLK** to minimize dynamic power dissipation, when active, **HCLK**, **SCLK**, **DCLK**, and **FCLK** must all be synchronous, and balanced to a ratio of 1:1.
- When a debugger is connected, **HCLK**, **DCLK**, and **SCLK** must all be running.
- **DCLK** must be tied LOW if the Cortex-M23 processor is configured without debug.

4.2.3 GREBEINTEGRATION_MCU level clocks

Table 4-3 shows the clocks of the `GREBEINTEGRATION_MCU` level.

Table 4-3 GREBEINTEGRATION_MCU level clocks

Name	Direction	Description	Connection information
FCLK	Input	Free running clock that clocks the `GREBEINTEGRATION_MCU` level and the TPIU.	**FCLK** must always be running.
SCLK	Input	System domain clock. This clocks a small amount of logic in the processor system domain.	**SCLK** must be derived directly from **FCLK**. It must always be running unless the processor is in WIC-mode deep sleep and no debugger is connected.
HCLK	Input	Clock for the majority of the non-debug logic in the processor system domain.	**HCLK** must be derived directly from **FCLK**. Connect **HCLK** to the AHB layer that the processor is connected to. **HCLK** can be gated when the processor is sleeping and no debugger is connected. You can use **GATEHCLK** to determine when you can gate **HCLK**.

Table 4-3 GREBEINTEGRATION_MCU level clocks (continued)

Name	Direction	Description	Connection information
DCLK	Input	Clock for the processor debug domain.	**DCLK** must be derived directly from **FCLK**. **DCLK** must always be driven while a debugger is connected. It can be gated when no debugger is connected.
SWCLKTCK	Input	Clock for the JTAG or Serial Wire interface.	**SWCLKTCK** is typically driven by an external debugger and is asynchronous to **FCLK**, **HCLK**, **SCLK**, and **DCLK**. When the DAP implements JTAG the flip flops use the negative edges.
TRACECLKIN	Input	TPIU trace clock.	**TRACECLKIN** can be asynchronous to other clocks.

The example PMU module provided, `grebe_ik_pmu.v`, meets the clocking requirements at the `GREBEINTEGRATION_MCU` level.

Note

- Although under certain conditions, **HCLK**, **SCLK**, and **DCLK** can be gated versions of **FCLK** to minimize dynamic power dissipation, when active, **HCLK**, **SCLK**, **DCLK**, and **FCLK** must all be synchronous, and balanced to a ratio of 1:1.
- When a debugger is connected, **HCLK**, **DCLK**, and **SCLK** must all be running.
- **DCLK**, **SWCLKTCK**, and **TRACECLKIN** must be tied LOW if the Cortex-M23 processor is configured without debug.

4.2.4 Clock gating combinations

Figure 4-2 shows the legal clock gating combinations.

Ungated / Gated	FCLK	SCLK	HCLK	DCLK
FCLK	-	invalid	invalid	invalid
SCLK	WIC sleep†	-	invalid	invalid
HCLK	WIC sleep†	Core sleeping‡	-	invalid
DCLK	WIC sleep†	No debugger connected¶	No debugger connected¶	-

† **SLEEPDEEP** is HIGH, **WICDSACKn** is LOW, **WAKEUP** is LOW, and no debugger is connected.
‡ **SLEEPING** is HIGH and no debugger is connected.
¶ **CDBGPWRUPACK** is LOW.

Figure 4-2 Clock gating combinations

4.3 Resets

This section describes considerations for connecting and using the Cortex-M23 processor resets.

Before connecting resets in your system, you must consider the following:

- Assertion of **SYSRESETREQ** must only cause an assertion of **nHRESET** and not **nDBGRESET**. This permits a debugger to maintain a connection during a processor reset.
- Register **SYSRESETREQ** to ensure that **nHRESET** is driven for the minimum reset time for your SoC design. **SYSRESETREQ** is cleared asynchronously by **nHRESET**. Do not connect **SYSRESETREQ** directly to **nHRESET**.

4.3.1 GREBE level resets

Table 4-4 shows the resets at the GREBE level of hierarchy.

Table 4-4 GREBE level resets

Name	Direction	Description	Connection information
nHRESET	Input	Reset for the processor system domain and the AHB system	Deassert **nHRESET** synchronously to **SCLK**. Assert **nHRESET** on powerup or System domain powerdown. Assert **nHRESET** for at least two **HCLK** cycles.
nDBGRESET	Input	Reset for the processor debug domain	Deassert **nDBGRESET** synchronously to **SCLK**. Assert **nDBGRESET** on powerup or Debug domain powerdown. Assert **nDBGRESET** for at least two **DCLK** cycles. Tie **nDBGRESET** LOW when no debugger is connected.
nSYSPORESET	Input	Power-on reset for the logic inside GREBE level that only resets on a Cold reset.	Deassert **nSYSPORESET** synchronously to **FCLK**. Assert **nSYSPORESET** on powerup or System domain powerdown. Assert **nSYSPORESET** for at least two **FCLK** cycles.

If your system includes debug (DBG = 1), the AHB-Lite master interface is used for processor transfers and external debugger transfers. When either **nHRESET** or **nDBGRESET** is asserted but not both, to ensure that the AHB-Lite master interface complies with the AHB-Lite protocol specification, you must:

- Assert **nHRESET** synchronously to **SCLK**, and only when **HREADY** is HIGH. ARM recommends that **nHRESET** is driven from a register that is clocked by **SCLK** and enabled by **HREADY**.
- Do not assert **nDBGRESET** when a debugger transfer is in progress. You can achieve this by ignoring debug reset requests while a debugger is connected.

There are no reset synchronizers within the GREBE level.

4.3.2 GREBEINTEGRATION level resets

Table 4-5 shows the resets of the GREBEINTEGRATION level of hierarchy.

Table 4-5 GREBEINTEGRATION level resets

Name	Direction	Description	Connection information
nHRESET	Input	Reset for the processor system domain and the AHB system.	Deassert **nHRESET** synchronously to **FCLK**. Assert **nHRESET** on powerup or System domain powerdown. Assert **nHRESET** for at least two **HCLK** cycles.
nDBGRESET	Input	Reset for the processor debug domain.	Deassert **nDBGRESET** synchronously to **FCLK**. Assert **nDBGRESET** on powerup or Debug domain powerdown. Assert **nDBGRESET** for at least two **DCLK** cycles.
nSYSPORESET	Input	Power-on reset for the logic inside GREBE level that only resets on a Cold reset.	Deassert **nSYSPORESET** synchronously to **FCLK**. Assert **nSYSPORESET** on powerup or System domain powerdown. Assert **nSYSPORESET** for at least two **FCLK** cycles.
nPORESET	Input	Power-on reset for the JTAG or Serial Wire DP logic, QChannel logic.	Deassert **nPORESET** synchronously to **FCLK**. External debuggers cannot connect when **nPORESET** is asserted.

Note

The differences between **nSYSPORESET** and **nPORESET** are:

- **nPORESET** is asserted when the full system is powered on.
- **nSYSPORESET** is asserted when **nPORESET** is asserted, but must also be asserted when System power domain is off, and deasserted only when power is available in the System power domain.

The example reset controller provided meets the reset requirements at the GRBIKMCU level. See *GRBIKMCU level* on page 8-22 for more information.

4.3.3 GREBEINTEGRATION_MCU level resets

Table 4-6 shows the resets of the GREBEINTEGRATION_MCU level.

Table 4-6 GREBEINTEGRATION_MCU level resets

Name	Direction	Description	Connection information
nHRESET	Input	Reset for the processor system domain and the AHB system.	Deassert **nHRESET** synchronously to **FCLK**. Assert **nHRESET** on powerup. Assert **nHRESET** for at least two **HCLK** cycles.
nDBGRESET	Input	Reset for the processor debug domain.	Deassert **nDBGRESET** synchronously to **FCLK**. Assert **nDBGRESET** on powerup. Assert **nDBGRESET** for at least two **DCLK** cycles.
nTRESET	Input	TPIU reset.	**nTRESET** must be asserted when **nDBGRESET** is asserted. Deassert **nTRESET** synchronously to **TRACECLKIN**.

Table 4-6 GREBEINTEGRATION_MCU level resets (continued)

Name	Direction	Description	Connection information
nTRST	Input	Reset for the JTAG TAP Controller in the JTAG *Debug Port* (DP).	**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 JTAG is not configured.
nSYSPORESET	Input	Power-on reset for the logic inside GREBE level that only resets on a Cold reset.	Deassert **nSYSPORESET** synchronously to **FCLK**. Assert **nSYSPORESET** on powerup or System domain powerdown. Assert **nSYSPORESET** for at least two **FCLK** cycles.
nPORESET	Input	Power-on reset for the JTAG or Serial Wire DP logic, QChannel logic, and logic inside GREBE level that only resets on a Cold reset.	Deassert **nPORESET** synchronously to **FCLK**. External debuggers cannot connect when **nPORESET** is asserted.

There is a reset synchronizer within the GREBEINTEGRATION_MCU level that synchronizes **nPORESET** with **SWCLKTCK** when entering DAP.

4.4 Interfaces

This section describes the interfaces of the Cortex-M23 processor in:

4.4.1 External AHB-Lite interface

This interface is present at the `GREBE`, `GREBEINTEGRATION`, and `GREBEINTEGRATION_MCU` levels.

Accesses on this interface can originate from processor-initiated transactions or transactions requested by an external debugger.

ARM recommends that you understand the AHB-Lite bus interface signals, described in the *ARM® AMBA® 5 AHB Protocol Specification*, before connecting any of the signals described in this section.

Table 4-7 shows the External AHB-Lite interface.

Table 4-7 External AHB-Lite signals

Name	Direction	Connection information
HADDR[31:0]	Output	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HBURST[2:0]	Output	Connect to the AHB arbiter and slaves through the bus infrastructure.

Table 4-7 External AHB-Lite signals (continued)

Name	Direction	Connection information
HPROT[6:0]	Output	Connect to the slaves through the bus infrastructure.
HSIZE[2:0]	Output	Connect to the slaves through the bus infrastructure.
HTRANS[1:0]	Output	Connect to the AHB arbiter and slaves through the bus infrastructure.
HWRITE	Output	Connect to the slaves through the bus infrastructure.
HWDATA[31:0]	Output	
HRDATA[31:0]	Input	
HREADY	Input	
HRESP	Input	
HEXCL	Output	Bus Exclusive Transfer.
HEXOKAY	Input	Bus Exclusive Answer.
HNONSEC	Output	Secure control signals. This signal is LOW for Secure transactions and HIGH for Non-secure transactions. When the Security Extension is not present, this signal is always HIGH. **Note** This signal is HIGH for transactions during Secure execution on memory regions defined as Non-secure in the SAU or IDAU.
HMASTER	Output	Enables the system to distinguish between processor initiated and debugger initiated accesses: **LOW** Processor access. **HIGH** Debugger access.

AHB-Lite interface transaction types

Table 4-8 shows the Cortex-M23 processor generated AHB-Lite interface transaction types.

Table 4-8 AHB-Lite interface transaction types

Transaction	HTRANS[1:0]	HPROT[0]	HMASTER[a]	HSIZE[2:0]
Bus idle	00	X	X	0XX
Core instruction fetch[b]	10	0	0	001
				010
Core word load/store	10	1	0	010
Core halfword load/store	10	1	0	001
Core byte load/store	10	1	0	000
Debug word read/write	10	1	1	010
Debug halfword read/write	10	1	1	001
Debug byte read/write	10	1	1	000

a. **HMASTER** can be used by peripherals to distinguish between accesses from the processor core and accesses from the debugger.
b. If you implement the processor with `HWF=1`, the value of **HSIZE** for core instruction fetches is always `b001`, indicating 16-bit. If `HWF=0`, then **HSIZE** uses both values of `b010` and `b001`.

HPROT[6:2] derivation is based on the configuration of the *Memory Protection Unit* (MPU) and debug interface, or when the MPU is not implemented, the attributes defined by the default memory map of the processor. When no MPU is present, the default attributes are applied to both processor and debugger transactions. See the *ARMv8-M Architecture Reference Manual* for more information.

Table 4-9 shows the **HPROT[6:2]** memory types when an MPU is either not present or is disabled.

Table 4-9 Memory types for HPROT[6:2]

Address	Memory type	HPROT[6:2]	Recommended usage
`0xE0100000` - `0xFFFFFFFF`	Device XN[a]	`00001`	Vendor SYS.
`0xE0000000` - `0xE00FFFFF`	Strongly ordered XN	-	System Control Space.
`0xA0000000` - `0xDFFFFFFF`	Device XN	`00001`	Peripherals.
`0x80000000` - `0x9FFFFFFF`	Normal WT[b]	`00110` for write `01110` for read	Off chip RAM.
`0x60000000` - `0x7FFFFFFF`	Normal WBWA[c]	`01111`	Off chip RAM.
`0x40000000` - `0x5FFFFFFF`	Device XN	`00001`	Peripherals.
`0x20000000` - `0x3FFFFFFF`	Normal WBWA	`01111`	On chip RAM.
`0x00000000` - `0x1FFFFFFF`	Normal WT[b]	`00110` for write `01110` for read	ROM or flash.

a. XN means eXecute Never.
b. WT means Write-Through.
c. WBWA means write-back-write-allocate.

AHB-Lite byte lane definition

Table 4-10 shows the byte lanes the interface uses during the data phase on **HRDATA** or **HWDATA**, and the mapping to bytes in the source or destination processor core register, `Rd`, for all transfer sizes and each endianness configuration.

Table 4-10 AHB-Lite byte lane assignments

Address phase			Corresponding data phase			
HSIZE [1:0]	HADDR [1:0]	Processor endianness	HxDATA [31:24]	HxDATA [23:16]	HxDATA [15:8]	HxDATA [7:0]
`00`	`00`	-	-	-	-	`Rd[7:0]`
`00`	`01`	-	-	-	`Rd[7:0]`	-
`00`	`10`	-	-	`Rd[7:0]`	-	-
`00`	`11`	-	`Rd[7:0]`	-	-	-

Table 4-10 AHB-Lite byte lane assignments (continued)

Address phase			Corresponding data phase			
HSIZE [1:0]	**HADDR [1:0]**	**Processor endianness**	**HxDATA [31:24]**	**HxDATA [23:16]**	**HxDATA [15:8]**	**HxDATA [7:0]**
01	00	Little-endian	-	-	Rd[15:8]	Rd[7:0]
01	00	Big-endian	-	-	Rd[7:0]	Rd[15:8]
01	10	Little-endian	Rd[15:8]	Rd[7:0]	-	-
01	10	Big-endian	Rd[7:0]	Rd[15:8]	-	-
10	00	Little-endian	Rd[31:24]	Rd[23:16]	Rd[15:8]	Rd[7:0]
10	00	Big-endian	Rd[7:0]	Rd[15:8]	Rd[23:16]	Rd[31:24]

Note

- **HBURST** is always 3'b000.
- Instruction fetches are performed as 32-bit word for requests aligned on 32-bit, or 16-bit halfword accesses, if you implement the processor with HWF=0. Instruction fetches are always performed as 16-bit halfword accesses, if you implement the processor with HWF=1.

4.4.2 AHB interface extensions

This interface is present at the GREBE, GREBEINTEGRATION, and GREBEINTEGRATION_MCU levels.

You can use AHB interface extensions with the AHB interface to provide efficient connection to memory controllers and other peripherals. If this interface is not in use, you can leave it unconnected.

Table 4-11 shows the AHB-Lite interface extensions.

Table 4-11 AHB-Lite interface extensions

Name	Direction	Description
CODENSEQ	Output	This signal is valid when **HTRANS** is HIGH and **HPROT[0]** is LOW. It provides an augmentation to **HTRANS** to enable more efficient detection of sequential instruction fetches. When driven LOW, it indicates that the current instruction fetch is guaranteed to be sequential to the previous fetch. When driven HIGH, it indicates that the current instruction fetch is not guaranteed to be sequential to the previous fetch. This signal does not take interleaved data accesses or debugger accesses into account. This signal can be used to connect to the instruction prefetchers in the system.
CODEHINT[3:0]	Output	Prefetch hint bus: **CODEHINT[3]** When LOW, indicates that the next AHB cycle will not be a data transaction on behalf of the processor. It will either be a fetch, IDLE, or debugger transaction. When HIGH, any access may appear in the next AHB cycle. **CODEHINT[2]** When LOW, indicates that if the next AHB address phase for fetch will be to the word following the previous instruction fetch. It also indicates that the next AHB address phase for fetch will be sequential. When HIGH, indicates that the next cycle AHB that is a fetch will not be sequential or sequential to the previous, that is, **CODENSEQ** maybe be set. It also indicates that the next AHB address phase for fetch may be sequential or non-sequential. **CODEHINT[1]** When LOW, indicates that the next AHB address phase for fetch will be sequential. When HIGH, indicates that the next cycle AHB will either be IDLE, or, depending on **CODENSEQ**, a sequential fetch, or a fetch to a location the same as, or behind the previous instruction fetch. It also indicates that next AHB address phase for fetch will be either sequential or non-sequential (specific to unresolved conditional branch). **CODEHINT[0]** When LOW, indicates that the next AHB address phase for fetch will be sequential. When HIGH, indicates that the next cycle AHB will either be IDLE, or, depending on **CODENSEQ**, a sequential fetch, or a fetch to a location the same as, or ahead of the previous instruction fetch. It also indicates that next AHB address phase for fetch will be either sequential or non-sequential (specific to unresolved conditional branch). Use these signals to connect to a prefetch unit and influence prefetch decisions.

Table 4-11 AHB-Lite interface extensions (continued)

Name	Direction	Description
DATAHINT[1:0]	Output	Data transaction hint bus: **DATAHINT[1]** Indicates, for an AHB address phase generated by the processor, that the current data load was generated by a PC relative load. **DATAHINT[0]** Indicates, in an AHB address phase, that the current data load is for an exception vector.
SPECHTRANS	Output	A faster, speculative version of **HTRANS[1]**, active in the same cycle as **HTRANS[1]**. This signal is used to provide a version of **HTRANS[1]** to read-only slaves if timing closure on **HTRANS[1**] is problematic. This signal indicates the processor is ready to do a transaction, but does not take into account suppression of **HTRANS[1]**. This suppression can be caused by: • Attempted unaligned transactions. • Attempted instruction fetches from `Execute-Never`, `XN`, regions. • Transactions that access the internal PPB or the I/O port. • MPU or SAU faults. • Change of privacy during nonsequential fetches (1 cycle). • `STREX` fail because of the internal monitor. • `SPLIM` faults. If you use **SPECHTRANS**, be sure to maintain AHB compliance. For example, it is not acceptable to introduce an **HREADY** wait-state in response to a **SPECHTRANS** for which **HTRANS[1]** does not also become asserted.
CURRNS	Output	This signal indicates the security state of the processor at the time the access is performed. When the processor is in Secure state and performs an access to a Non-secure memory region, this signal is LOW while **HNONSEC** or **IONONSEC** are HIGH. When the processor fetches information from the Secure stack while in Non-secure state, for example during exception returns or return from function, this signal is HIGH while **HNONSEC** or **IONONSEC** are LOW. When the processor is in Non-secure state and branches to a Secure region, this signal is HIGH until an `SG` instruction is committed.

AMBA® Considerations

You must abide by the rules of AMBA® 5 AHB irrespective of the AHB interface extensions.

4.4.3 AHB memory considerations

AHB memories implemented for the purpose of executing code must be byte, halfword and word data, readable for both instruction and data transactions. A slave can respond with word data.

Writeable AHB memories must be capable of accepting byte writes without corrupting neighboring bytes.

The Cortex-M23 processor does not guarantee that addresses are aligned on the size when **HTRANS** is IDLE. This could cause some AHB protocol checkers to fail.

4.4.4 I/O port

This interface is present at the `GREBE`, `GREBEINTEGRATION`, and `GREBEINTEGRATION_MCU` levels.

The Cortex-M23 processor provides an optional interface to provide low-latency input and output using load and store instructions to a dedicated address range. Set the `IOP` parameter to `1` to include the I/O port in your design. See *GREBE configuration options* on page 2-4.

The interface operates entirely in a single **HCLK** cycle, with transaction address and associated read or write data generated or returned in the same cycle. Because of the tight timing constraints of this interface, ARM recommends that you keep the number of attached peripherals to a minimum.

Logic external to the processor decodes addresses, and indicates to the processor if it must issue transactions for a particular address to the AHB-Lite or the I/O port.

The I/O port consists of a decode interface and a transaction interface. The decode interface consists of an address query output, **IOCHECK**, and a response from an external decoder, on **IOMATCH**. The external decoder combinatorially decodes the address on **IOCHECK** and asserts **IOMATCH** if transactions must use the I/O port.

For example, if all transactions in the address `range 0xABCD0000` to `0xABCDFFFF` are to use the I/O port, then the **IOMATCH** must be constructed as:

```
wire IOMATCH = IOCHECK[31:16] == 0xABCD;
```

Note

IOMATCH must never be HIGH if the value of **IOCHECK[31:28]** is `0xE`, as the address space `0xE0000000` to `0xEFFFFFFF` is reserved for the processor System Control Space. If you do not observe this restriction, the processor might malfunction.

The I/O port transaction interface performs a transaction by driving the **IOTRANS** signal HIGH, with **IOWRITE** driven LOW for a read and HIGH for a write. Based on the values of **IOADDR**, **IOMASTER**, **IOSIZE** and **IOPRIV**, any attached peripheral must, in the same **HCLK** cycle, sample **IOWDATA** for a write, or provide data on **IORDATA** for a read. The active byte lanes for the I/O port are identical to those for AHB-Lite, based on **IOADDR[1:0]** and **IOSIZE**. See *AHB-Lite byte lane definition* on page 4-12.

The Cortex-M23 processor execution testbench provides an example general-purpose I/O block suitable for connection to the I/O port interface. See *Testbench structure* on page 8-21.

Table 4-12 shows the I/O port signals.

Table 4-12 I/O port signals

Name	Direction	Description
IOCHECK[31:0]	Output	I/O address decoder query.
IOMATCH	Input	I/O address decoder response: **LOW** Address on **IOCHECK** uses AHB. **HIGH** Address on **IOCHECK** uses I/O port.
IOTRANS	Output	I/O port transaction valid.
IOADDR[31:0]	Output	I/O port transaction address.
IOWRITE	Output	I/O port transaction write control: **LOW** Transaction is a read. **HIGH** Transaction is a write.
IOWDATA[31:0]	Output	I/O port write data, for writes.

Table 4-12 I/O port signals (continued)

Name	Direction	Description
IOSIZE[1:0]	Output	I/O port transaction size: **0b00** Byte, 8-bit. **0b01** Halfword, 16-bit. **0b10** Word, 32-bit. **0b11** Never used.
IOPRIV	Output	I/O port transaction privilege level: **LOW** Non-Privileged. **HIGH** Privileged.
IOMASTER	Output	I/O port transaction source: **LOW** Transaction from software on processor. **HIGH** Transaction through debug interface.
IORDATA[31:0]	Input	I/O port read data, for reads.
IONONSEC	Output	I/O port transaction security: **LOW** Transaction is Secure. **HIGH** Transaction is Non-secure. When the Security Extension is not present, this signal is always HIGH. This signal is valid when **IOADDR** is valid. **Note** This signal is HIGH for transactions during Secure execution on memory regions defined as Non-secure in the SAU or IDAU.

Figure 4-3 on page 4-18 shows two single read accesses. In this example, the I/O device asserts IOMATCH and the processor responds, in the same cycle, by asserting IOTRANS.

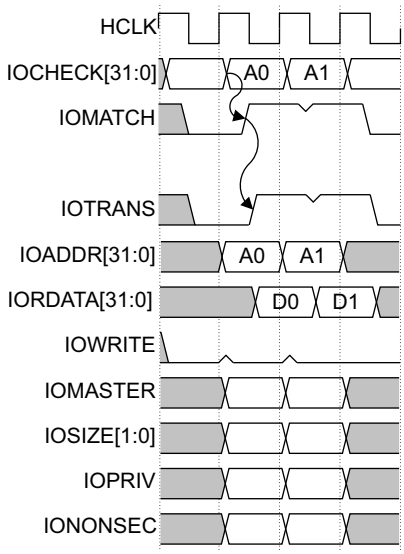


Figure 4-3 I/O port read timing single read

Figure 4-4 shows two single write accesses. In this example, the I/O device asserts IOMATCH and the processor responds, in the same cycle, by asserting IOTRANS.

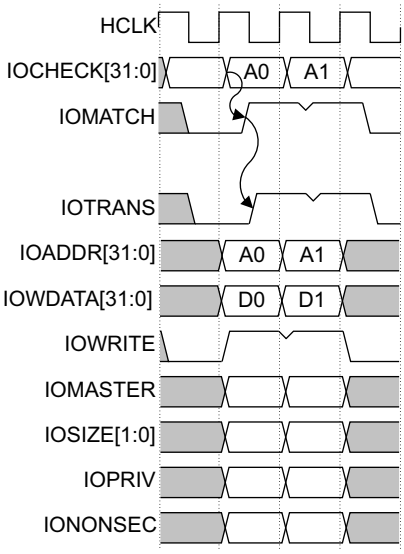


Figure 4-4 I/O port write timing single writes

Figure 4-5 on page 4-19 shows a multiple write access. In this example, the processor asserts IOTRANS in the cycle after the I/O device asserts IOMATCH.

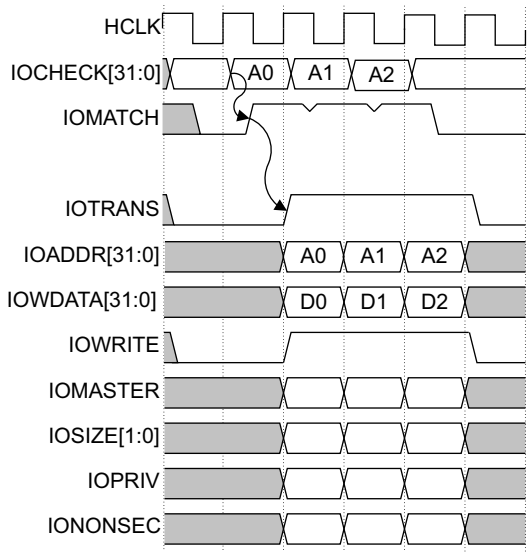


Figure 4-5 I/O port write timing store multiple

4.4.5 External Secure Attribution Unit Interface

The external SAU interface present on the processor and the *Implementation Defined Attribution Unit* (IDAU) interface is present at the `GREBE`, `GREBEINTEGRATION`, and `GREBEINTEGRATION_MCU` levels when the Security Extension is included.

An IDAU can control the security attributes for most of the memory the Cortex-M23 processor addresses to a granularity of 32-bytes.

The security level returned by the Cortex-M23 processor SAU is a combination of the region type defined in the internal SAU, if configured, and the type that is returned on the associated IDAU. If an address maps to regions defined by both internal and external attribution units, the region of the highest security level is selected. Table 4-13 provides examples of the final security level in this case.

Table 4-13 Examples of Highest Security Level Region

IDAU	SAU Region	Final Security
S	X	S
X	S	S
S-NSC or NS	S-NSC	S-NSC
S-NSC	S-NSC or NS	S-NSC
NS	NS	NS

Note

To allow Non-secure regions, you need to define a range of memory as Non-secure and/or Secure and Non-secure callable on the IDAU interface.

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 external IDAU overriding any security level.

To allow an IDAU to specify the security level for all memory regions not defined by enabled SAU regions, 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 CFGSECEXT input is set to 1.

Table 4-14 shows the signals for two external IDAU interfaces, A and B. All signal directions are relative to the processor.

Note

- When the Cortex-M23 processor's I/O Port is not present, only port A is used.
- When the Cortex-M23 processor's I/O Port is present:
  - Port A is used for data access, debug accesses, and some of the instruction fetches.
  - Port B is used for the remaining instruction fetches.
- IDAU responses must not change when **HREADY** is low.

Table 4-14 External IDAU Interfaces

Name	Direction	Description	Connection information
IDAUADDRA[26:0]	Output	Address of the transfer	**IDAUADDRA[26:0]** is the 32-byte address associated with the access address. For example, for a 32-bit memory address, **IDAUADDRA[26:0]** is **HADDR[31:5]**.
IDAUADDRB[26:0]	Output	Address of the transfer	**IDAUADDRB[26:0]** is the 32-byte address associated with the access address. For example, for a 32-bit memory address, **IDAUADDRB[26:0]** is **HDDR[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. This region ID is used by the TT instruction when **IDAUIDVA** is set. Tie this input LOW if an IDAU is not included.

Table 4-14 External IDAU Interfaces (continued)

Name	Direction	Description	Connection information
IDAUIDVA	Input	Region number valid	**IDAUIDVA** indicates that the IDAU region number is valid. This signal is used by the TT instruction. 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 access behaves as if the processor security state defined by the DHCSR.S_SDE register bit, or security level defined by **HNONSEC** on D-AHB, defines the security level. 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 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 access behaves as if the processor security state defined by the DHCSR.S_SDE register bit, or security level defined by **HNONSEC** on D-AHB, defines the security level. 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, the processor SAU or IDAU determines the security attribution unless the address is specified as exempt from security attribution checks. Tie this input LOW if an IDAU is not included.

4.4.6 Interrupt interface

This interface is present at the GREBE, GREBEINTEGRATION, and GREBEINTEGRATION_MCU levels.

Table 4-15 shows the signals of the external interrupt interface, and describes how you must connect the signals in your SoC design.

Table 4-15 Interrupt signals

Name	Direction	Description	Connection information
IRQ[239:0]	Input	External interrupt signals	When the processor is implemented with fewer than 240 external interrupts, the unused **IRQ** inputs must be tied LOW. When correctly configured by software, the Cortex-M23 processor takes an interrupt in response to an **IRQ** input either being held HIGH, or being HIGH for a single cycle of: • **FCLK** for the GREBEINTEGRATION or GREBEINTEGRATION_MCU level configured to include WIC support for the respective **IRQ** line. • **SCLK** for all other cases.
NMI	Input	Non-maskable interrupt	Irrespective of software configuration, the Cortex-M23 processor takes an NMI exception in response to **NMI** being held HIGH, or being HIGH for a single cycle of: • **FCLK** for the GREBEINTEGRATION or GREBEINTEGRATION_MCU level configured to include WIC support for the **NMI**. • **SCLK** for all other cases. **Note** NMI may not be able to immediately preempt the processor execution when AIRCR.BFHFNMINS=1 and the processor executes Secure Hardfault handler

The Cortex-M23 processor does not implement synchronizers for the **IRQ[239:0]** and **NMI** inputs. If you want to use asynchronous interrupts, you must implement external synchronizers to reduce the possibility of metastability issues.

4.4.7 External debugger interface

The external debugger interface is only present at the GREBEINTEGRATION_MCU level.

It enables a compliant, off-chip debugger to gain access to the processor control resources and external memory.

Although you can configure GREBEINTEGRATION_MCU to implement either a JTAG debugger interface or a Serial Wire debugger interface, you cannot implement both.

Note

- If you require both Serial Wire and JTAG interfaces, you can connect a suitable DAP, like the CoreSight DAP, to the debug slave interface instead of the DAP. If this is the case, you must ensure that AHBSLV is configured correctly for your DAP.
- If your processor is not configured for debug, you must correctly tie off the JTAG or Serial Wire debugger interface signals.

For more information on integrating:

- The JTAG interface, see *JTAG signals* on page 4-23.
- The Serial Wire interface, see *Serial Wire signals* on page 4-23.

JTAG signals

Table 4-16 lists the JTAG interface signals.

For more information, see *IEEE 1149.1-2001 Test Access Port and Boundary Scan Architecture*.

Table 4-16 JTAG signals

Name	Direction	Description	Connection information
nTRST	Input	JTAG nTRST	See *Resets* on page 4-7.
TDI	Input	JTAG TDI	Connect to input pad for **TDI**.
SWDITMS	Input	JTAG TMS	Connect to input pad for **TMS**.
SWCLKTCK	Input	JTAG TCK	Connect to input pad for **TCK**.
TDO	Output	JTAG TDO	Connect to output pad for **TDO**. Optionally you can use a tristate pad.
nTDOEN	Output	JTAG TDO output pad control signal	Connect to optional tristate pad for **TDO**.

Serial Wire signals

Table 4-17 lists the Serial Wire signals.

Table 4-17 Serial Wire signals

Name	Direction	Description	Connection information
SWDO	Output	Serial Wire Data Out	Connect to tristate pad for **SWDIO.**
SWDOEN	Output	Serial Wire Data Out output pad control signal	Connect to tristate pad for **SWDIO.**
SWCLKTCK	Input	Serial Wire Clock	Connect to input pad for **TCK**.
SWDITMS	Input	Serial Wire Data In	Connect to tristate pad for **SWDIO**.

Miscellaneous DAP signals

Table 4-18 lists the DAP signals.

Table 4-18 Miscellaneous DAP signals

Name	Direction	Description	Connection information
HALTED	Input	Processor Halted.	Connect to Cortex-M23 processor **HALTED** output.
DEVICEEN	Input	Enable debugger access to system through SLV interface.	Tie this signal HIGH to enable DAP accesses to the SLV bus, subject to access permission based on debug authentication signals. Tie this signal LOW to permanently disable debugger access. Connect this signal to your own logic, such as that connected to the Cortex-M23 processor **DBGEN**, to dynamically enable and disable debugger access.
INSTANCEID[3:0]	Input	Configuration for Serial Wire Multi-drop target selection.	If you implement a Serial Wire DAP with multi-drop, 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. See the description of the Target Selection Register in the *ARM® Debug Interface v5 Architecture Specification Supplement*. If you do not implement a Serial Wire DAP with multi-drop, tie this input to 0b0000.
SWDETECT	Output	Serial Wire protocol detected.	HIGH for one cycle of **SWCLKTCK** when a Serial Wire DAP is implemented, and a Serial Wire line reset sequence is attempted while the DAP is not in dormant state. Optionally, connect to your own logic. For example, to disable a JTAG test device when a Serial Wire DAP is implemented without multi-drop. See Figure 4-6 on page 4-25.

Figure 4-6 on page 4-25 shows an example of how you might use the **SWDETECT** signal to disable a JTAG TAP Controller when a Serial Wire line reset sequence is observed.

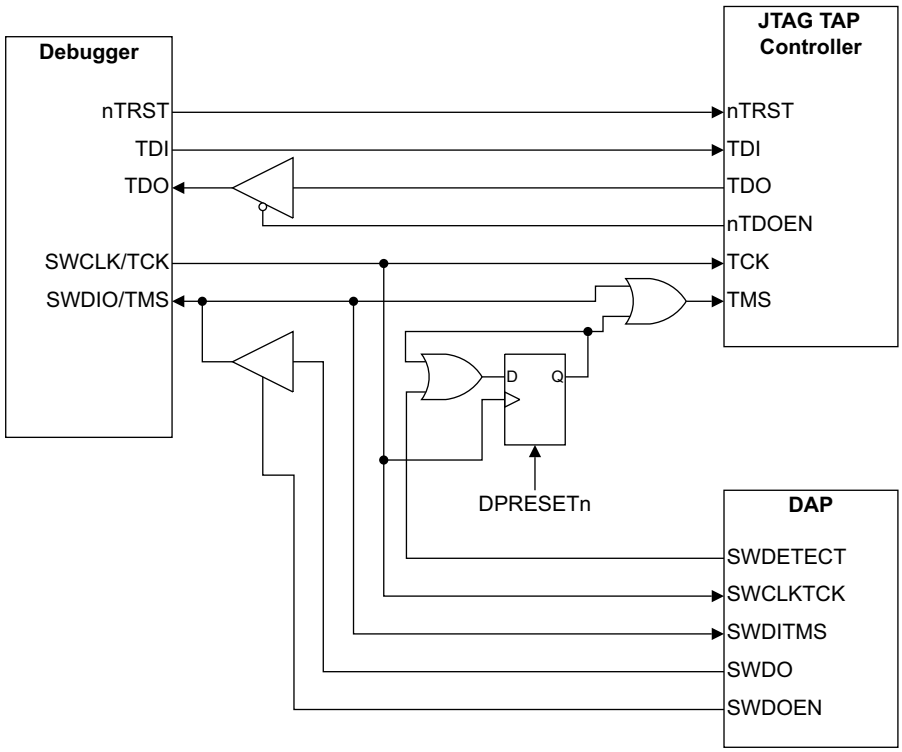


Figure 4-6 Example SWDETECT logic

4.4.8 Debug slave interface

The debug slave interface is present at the `GREBE` and `GREBEINTEGRATION` levels. It enables debugger access to processor control resources and system memory.

Table 4-19 shows the signals for the debug slave interface on the Cortex-M23 processor. These signals must be connected to either the DAP or the AHB-AP port of a CoreSight DAP.

If a system does not implement debug, tie all slave port inputs LOW.

The Cortex-M23 processor debug slave port behavior is configurable through the `AHBSLV` parameter. This parameter is set at implementation time. See *Configuration options* on page 2-4 for more information about the options for `AHBSLV`.

Table 4-19 Slave Port interface signals

Name	Direction	Connection information
HRDATAD[31:0]	Output	Connect to the **SLVRDATA** port of the DAP, or the **HRDATA** port of a CoreSight AHB-AP.
HREADYD	Output	Connect to the **SLVREADY** port of the DAP, or the **HREADY** port of a CoreSight AHB-AP.
HRESPD	Output	Connect to the **SLVRESP** port of the DAP, or the **HRESP[0]** port of a CoreSight AHB-AP. When connecting to a CoreSight AHB-AP, bit[1] of the **HRESP** port on the CoreSight™ AHB-AP must be tied LOW.
HADDRD[31:0]	Input	Connect to the **SLVADDR** port of the DAP, or the **HADDR** port of a CoreSight AHB-AP.
HTRANSD[1:0]	Input	Connect to **SLVTRANS** port of the DAP or the **HTRANS** port of a CoreSight AHB-AP.

Table 4-19 Slave Port interface signals (continued)

Name	Direction	Connection information
HWRITED	Input	Connect to the **SLVWRITED** port of the DAP, or the **HWRITE** port of a CoreSight AHB-AP.
HWDATAD[31:0]	Input	Connect to the **SLVWDATA** port of the DAP, or the **HWDATA** port of a CoreSight AHB-AP.
HSIZED[1:0]	Input	Connect to the **SLVSIZE** port of the DAP, or the **HSIZE[1:0]** port of a CoreSight AHB-AP.
HPROTD[6:0]	Input	Connect to the **SLVPROT** port of the DAP, or the **HPROT[6:0]** port of a CoreSight AHB-AP.
HNONSECD	Input	Connect to **SLVNONSEC** of the DAP, or the **HNONSEC** of a CoreSight DAP.

4.4.9 Debug authentication

You can use the **DBGEN**, **NIDEN**, **SPIDEN**, and **SPNIDEN** inputs to disable trace or prevent the **HALTED** output going active.

Table 4-20 lists the debug authentication signals and describes how to connect them.

Table 4-20 Debug authentication signals

Name	Direction	Connection information
DBGEN	Input	If you are not using debug authentication, then this signal must be tied HIGH. To permanently disable debug, tie this signal LOW. To dynamically enable and disable debug, connect this signal to your own logic.
NIDEN	Input	If you are not using debug authentication, then this signal must be tied HIGH. To permanently disable trace, tie this signal LOW. To dynamically enable and disable trace, connect this signal to your own logic.
SPIDEN	Input	If you are not using debug authentication, then this signal must be tied HIGH. To permanently disable Secure invasive debug, tie this signal LOW. To dynamically enable and disable Secure invasive debug, connect this signal to your own logic. **Note** Secure software can override this input by writing the DAUTHCTRL register.
SPNIDEN	Input	If you are not using debug authentication, then this signal must be tied HIGH. To permanently disable Secure non-invasive debug, tie this signal LOW. To dynamically enable and disable Secure non-invasive debug, connect this signal to your own logic. **Note** Secure software can override this input by writing the DAUTHCTRL register.

The **DBGEN, NIDEN**, **SPIDEN**, and **SPNIDEN** inputs must be synchronous to the **HCLK** input.

4.4.10 DWT interface

Table 4-21 shows the DWT signals and describes how to connect them.

Table 4-21 DWT signals

Name	Direction	Description
DWTMATCH[3:0]	Input	Indicates that the DWT trigger units have matched the conditions currently present on the address, data, and control buses.
DWTINOTD[3:0]	Input	Indicates that the DWT trigger units are performing comparisons on the PC value (set) or the data address (clear).

4.4.11 ASIC level signals

Table 4-22 shows the ASIC level signals and describes how to connect them.

Table 4-22 ASIC level signals

Name	Direction	Description
NIDEN[a]	Input	Non-invasive debug enable.
EXTIN[1:0]	Input	External input resource.
MAXEXTIN[1:0]	Input	Maximum supported external inputs.

a. This signal is driven by the **ETMNIDALLOWED** `GREBE` output

4.4.12 Miscellaneous signals

These signals are present at the `GREBE`, `GREBEINTEGRATION`, and `GREBEINTEGRATION_MCU` levels unless otherwise stated.

Table 4-23 shows the miscellaneous signals present on the processor, and describes how to connect the signals.

Table 4-23 Miscellaneous signals

Name	Direction	Description	Connection information
CPUWAIT	Input	Drive HIGH to cause the processor to wait for the signal to be LOW before coming out of reset.	If you do not require this functionality, tie this signal LOW.
DBGACTIVE	Output	Debug is enabled.	Connect to the QChannel module, which is instantiated at `GREBEINTEGRATION` level.
DBGRESTART[ab]	Input	External restart request.	If you are not using this signal to connect to a CTI, tie this signal LOW. If you are using this signal to connect to a CTI, contact ARM for connection information.

Table 4-23 Miscellaneous signals (continued)

Name	Direction	Description	Connection information
DBGRESTARTED[ab]	Output	Handshake for **DBGRESTART**.	If you are not using this signal to connect to a debug CTI, leave this signal unconnected. If you are using this signal to connect to a CTI, contact ARM for connection information.
DBGTRCENA	Output	Watchpoints are enabled in DEMCR.	Connect to the QChannel module, which is instantiated at `GREBEINTEGRATION` level.
ECOREVNUM[19:0][c] **ECOREVNUM[31:0]**[d] **ECOREVNUM[47:0]**[e]	Input	This bus provides a way to implement engineering change order modification for certain bits in the architected ID registers in the processor, DAP, and CoreSight MTB-M23.	These signals must be tied LOW unless you have an *Engineering Change Order* (ECO) from ARM. ARM recommends that you ensure each of the signal drivers is distinct and readily identifiable to facilitate possible metal layer modification.
EDBGRQ[a]	Input	External debug request.	This signal can be asserted by a debug agent in the system, for example the CoreSight MTB-M23, to request that the core enters Debug state. The Cortex-M23 processor only enters Debug state if debug is implemented and **C_DEBUGEN** in the DHCSR is set. Tie this signal LOW if you are not using it. See the *ARMv8-M Architecture Reference Manual* for more information. **Note** If Secure intrusive debug is not permitted and the processor runs in the Secure state, the processor enters Halt mode only when it starts executing Non-secure code.
GATEHCLK	Output	When HIGH, indicates that **HCLK** can be safely gated. You can use this signal to generate **HCLK** from **FCLK**.	This signal is only present at the `GREBEINTEGRATION` and `GREBEINTEGRATION_MCU` levels.
HALTED[a]	Output	When HIGH, indicates that the processor is in Debug state. **HALTED** remains asserted for as long as the processor remains in debug state.	Connect to the DAP if you are using the DAP **HALTED** input and if you are using the DAP Halt Event Signaling feature. If you are using this signal to connect to a CTI, contact ARM for connection information.

Table 4-23 Miscellaneous signals (continued)

Name	Direction	Description	Connection information
IRQLATENCY[7:0]	Input	The Cortex-M23 processor supports zero jitter interrupt latency for zero wait-state memory. **IRQLATENCY** specifies the minimum number of cycles between an interrupt that becomes pended in the NVIC, and the vector fetch for that interrupt being issued on the AHB-Lite interface.	If this bus is set to `0`, interrupts are taken as quickly as possible. For non-zero values, the processor ensures that a minimum of **IRQLATENCY**+1 **SCLK** cycles exist between an interrupt becoming pended in the NVIC and the vector fetch for the interrupt being performed. For zero jitter in a zero wait state memory system: • If you have excluded Security Extension, set this bus to at least a decimal value of 9. This causes an interrupt latency of 15 cycles from the assertion of the NVIC **IRQ** pin to execution of the exception handler, in a zero-wait-state system. • If you have included Security Extension, set this bus to at least a decimal value of 21. This causes an interrupt latency of 27 cycles from the assertion of the NVIC **IRQ** pin to execution of the exception handler, in a zero-wait-state system. For non-zero wait state memory, zero jitter can be achieved with a higher value on **IRQLATENCY**. **IRQLATENCY** is sampled synchronously to **SCLK**. **Note** No faults happen during interrupt return and stacking. Zero jitter interrupt latency cannot be guaranteed if there is fault.
LOCKUP	Output	When HIGH, indicates that the processor is in the architected Lockup state, as the result of an unrecoverable exception. See the *ARMv8-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 **nHRESET**. 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 **nHRESET** 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 **nHRESET** because of **LOCKUP**. If you require entry into Lockup state to reset the system, your code must enable the functionality in your watchdog unit.

Table 4-23 Miscellaneous signals (continued)

Name	Direction	Description	Connection information
RXEV	Input	A HIGH level on this input causes the ARM v8-M architecture defined Event Register to be set in the Cortex-M23 processor. This causes a WFE instruction to complete. It also awakens the processor if it is sleeping as the result of encountering a WFE instruction when the Event Register is clear.	The input to this signal must be constructed 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. Tie this input LOW if there are no non-interrupt event generating sources in your system.
TXEV	Output	A single **SCLK** cycle HIGH level is generated on this output every time an SEV instruction is executed on the Cortex-M23 processor.	Use this to signal an event to other ARM processors.
SLEEPDEEP	Output	Active only when **SLEEPING** is HIGH. Indicates that the **SLEEPDEEP** bit in the NVIC is set to 1.	
SLEEPING	Output	When HIGH, indicates the processor is idle, waiting for an interrupt on the **IRQ**, **NMI**, or internal SysTick, or HIGH level on **RXEV**. 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.	
SYSRESETREQ	Output	System Reset Request. When HIGH indicates that a reset has been requested by program code or a debugger writing to the **AIRCR.SYSRESETREQ** bit.	

a. Only functional if configuration includes debug.
b. DBGRESTART forms a 4-phase handshake with DBGRESTARTED to aid asynchronous implementations. Logic driving DBGRESTART must observe the 4-phase handshake protocol to ensure correct operation.
c. This bus is **[19:0]** for GREBE.
d. This bus is **[31:0]** for GREBEINTEGRATION.
e. This bus is **[47:0]** for GREBEINTEGRATION_MCU.

4.4.13 SysTick signals

These signals are present at the GREBE, GREBEINTEGRATION, and GREBEINTEGRATION_MCU levels.

The ARMv8-M system timer, SysTick, is a system-agnostic timer implementation for operating system use. When you configure the processor without the SysTick timer, both the **STCLKEN** and **CFGSTCALIB** signals are non-functional.

Note

- If the Security Extension is present and two SystTick timers are implemented, signals such as **STCLKEN** and **STCLKENNS** are used.

- **STCLKEN** is used if there is a single SysTick, or if there are two SysTicks in the case of Secure SysTick.

Software can configure the SysTick timer to select **SCLK** as its clock source, or an alternative clock source. See Appendix D *SysTick Examples* for more information.

In the Cortex-M23 processor, the alternative clock source is based on **SCLK** gated by **STCLKEN**. If you want to use an asynchronous clock to generate **STCLKEN**, you must use a synchronizer circuit. Figure 4-7 on page 4-32 shows an example asynchronous clock generating **STCLKEN**.

If integration is performed without an alternative clock source, tie **STCLKEN** LOW and tie **CFGSTCALIB[25]** HIGH.

Table 4-24 shows the **CFGSTCALIB[25:0]** values required for the software developer using the SysTick calibration register in the Cortex-M23 processor NVIC memory map.

Table 4-24 SysTick signals

Name	Direction	Description	Connection information
CFGSTCALIB[25]	Input	NOREF Indicates that no alternative reference clock source has been integrated.	Tie HIGH if **STCLKEN** has been tied off.
CFGSTCALIB[24][a]	Input	SKEW	Tie this LOW if the system timer clock, the external reference clock, or **SCLK** as indicated by **CFGSTCALIB[25]**, can guarantee an exact multiple of 10ms. Otherwise, tie this signal HIGH.
CFGSTCALIB[23:0][a]	Input	TENMS Provides an integer value to compute a 10ms (100Hz) delay from either the reference clock, or **SCLK** if the reference clock is not implemented.	For an example, apply the value `0x07A11F` if no reference is implemented, and **SCLK** is 50MHz.
STCLKEN	Input	System clock enable signal.	SysTick clock enable signal. If software configures the SysTick timer to use the external reference clock, then it decrements on each **SCLK** rising edge for which **STCLKEN** is HIGH.

a. Used for Non-secure SysTick if you have excluded Security Extension or a single SysTick is implemented or for Secure SysTick if you have included Security Extension.

Table 4-25 Non-secure SysTick signals

Name	Direction	Description	Connection information
CFGSTCALIBNS[25][a]	Input	NOREF Indicates that no alternative reference clock source has been integrated.	Tie HIGH if **STCLKENNS** has been tied off.
CFGSTCALIBNS[24] [a]	Input	SKEW	Tie this LOW if the system timer clock, the external reference clock, or **SCLK** as indicated by **CFGSTCALIBNS[25]**, can guarantee an exact multiple of 10ms. Otherwise, tie this signal HIGH.
CFGSTCALIBNS[23:0][a]	Input	TENMS Provides an integer value to compute a 10ms (100Hz) delay from either the reference clock, or **SCLK** if the reference clock is not implemented.	For an example, apply the value `0x07A11F` if no reference is implemented, and **SCLK** is 50MHz.
STCLKENNS	Input	System clock enable signal.	SysTick clock enable signal. If software configures the SysTick timer to use the external reference clock, then it decrements on each **SCLK** rising edge for which **STCLKENNS** is HIGH.

a. Used only if you have included Security Extension and two SysTick timers are implemented.

Figure 4-7 shows an example asynchronous clock generating **STCLKEN**.

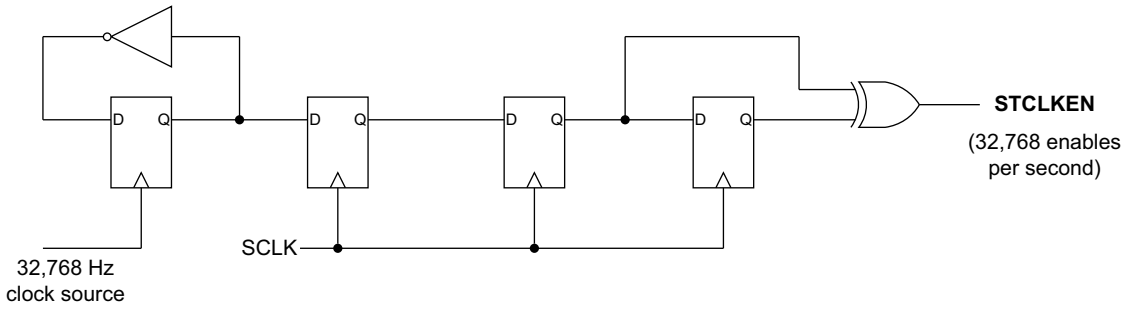


Figure 4-7 Asynchronous SysTick clock example

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

- **STCLKEN** LOW.
- **CFGSTCALIB[25]** HIGH.
- **CFGSTCALIB[24:0]** LOW.

4.4.14 WIC interface

Table 4-26 shows the WIC interface signals at the GREBE level.

Table 4-26 WIC interface signals at the GREBE level

Name	Direction	Description
WICDSREQn	Input	Active LOW request for deep sleep to be WIC-based deep sleep
WICDSACKn	Output	Active LOW acknowledge that deep sleep is WIC-based deep sleep
WICMASKISR[239:0]	Output	Interrupt sensitivity mask used for WIC wake-up detection
WICMASKNMI	Output	**NMI** sensitivity mask used for WIC wake-up detection
WICMASKRXEV	Output	**RXEV** sensitivity for WIC wake-up detection
WICLOAD	Output	Indicates that the WIC must reload its mask from **WICMASKISR**, **WICMASKNMI**, and **WICMASKRXEV**
WICCLEAR	Output	Indicates that the WIC must clear its mask

Table 4-27 shows the WIC interface signals at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels.

Table 4-27 WIC interface signals at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels

Name	Direction	Description
WAKEUP[ab]	Output	Active HIGH signal to the PMU that indicates a wake-up event has occurred and the processor system domain requires its clocks and power restored.
WICSENSE[a]	Output	Active HIGH set of signals. These indicate which input lines can cause the WIC to generate the **WAKEUP** signal.
WICENREQ[ac]	Input	Active HIGH request for deep sleep to be WIC-based deep sleep. This is driven from the PMU.
WICENACK[a]	Output	Active HIGH acknowledge signal for **WICENREQ**. **Note** If **WAKEUP** is asserted, **SYSQACTIVE** is also asserted.

a. Only functional if the WIC interface is included.
b. This signal can be ignored by the PMU when Q-channels are implemented.
c. Tie LOW if the WIC interface is not included.

The WIC interface is only functional if the WIC is included.

If no WIC interface is included, tie **WICDSREQn** HIGH.

Figure 4-8 on page 4-34 shows the WIC mode enable sequence that is required when using either the legacy interface or the Q-Channel interface. See *Power Management Unit - Legacy Interface* on page 4-34 and *Power Management Unit - Q-Channel Interface* on page 4-36.

It includes:

- The **WICENREQ** and **WICENACK** PMU interface signals that are present at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels.

- The **WICDSREQn** and **WICDSACKn** WIC signals that are present at the GREBE level only.

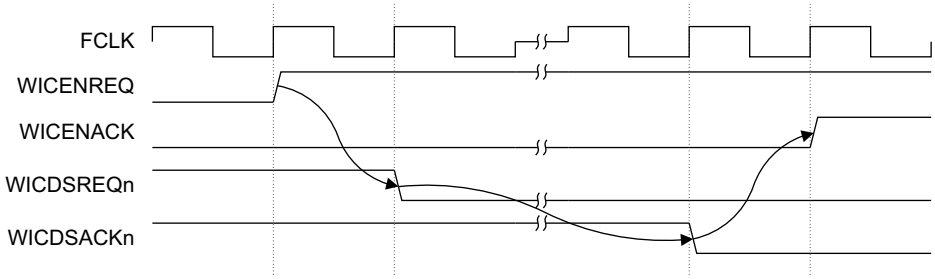


Figure 4-8 WIC mode enable sequence

4.4.15 Power Management Unit - Legacy Interface

The PMU interface is only present at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels.

Table 4-28 shows the PMU legacy interface signals.

Table 4-28 PMU legacy interface signals

Name	Direction	Description
CDBGPWRUPREQ	Output	Active HIGH signal that indicates an external debugger request to the PMU to powerup the debug domain in readiness for a debugging session. This signal must be synchronized before use. **Note** The PMU must also power up the system domain when this signal is HIGH.
CDBGPWRUPACK	Input	Active HIGH signal that indicates the debug domain is powered up in response to **CDBGPWRUPREQ** being HIGH. This signal must be driven synchronously to **FCLK**.
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.
SLEEPHOLDACKn[a]	Output	Response to **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.

a. SLEEPHOLDREQn forms a 4-phase handshake with SLEEPHOLDACKn to aid asynchronous implementations. Logic driving SLEEPHOLDREQn must observe the 4-phase handshake protocol to ensure correct operation.

The PMU legacy interface and the PMU Q-Channel interface are mutually exclusive and cannot be used together. When using the legacy interface you must ensure that Q-Channel interface is driven in this way:

- Permanently tie LOW the **SYSQREQn** and **DBGQREQn** inputs.
- Ignore the **SYSQACCEPTn**, **SYSQDENYn**, **SYSQACTIVE**, **DBGQACCEPTn**, **DBGQDENYn**, and **DBGQACTIVE** outputs.

CDBGPWRUPREQ and **CDBGPWRUPACK** form a four-phase handshake. It is a requirement that **CDBGPWRUPACK** is deasserted out of power-on reset and is only asserted or deasserted in response to the assertion and deassertion respectively of **CDBGWPRUPREQ**.

For more information about the possible low-power modes, see Table F-2 on page F-5.

If you are not implementing a PMU and a multi-power domain system, you must synchronize **CDBGPWRUPREQ** to **FCLK** and connect the synchronized signal to **CDBGPWRUPACK**.

You must ensure that when **CDBGPWRUPACK** is HIGH, all processor power domains are powered-up and all clocks are running.

Figure 4-8 on page 4-34 shows how the **WICENREQ** and **WICENACK** signals are sequenced. Figure 4-9 on page 4-36 shows how the **WAKEUP** and **WICSENSE** signals are sequenced.

Figure 4-9 on page 4-36 shows the power down timing sequence. It includes:

- The **SLEEPING** and **SLEEPDEEP** signals.
- The WIC interface signals that are present at the `GREBE` level only. See Table 4-26 on page 4-33
  - **WICLOAD**.
  - **WICCLEAR**.
  - **WICMASK*** (representing **WICMASKISR**, **WICMASKNMI**, and **WICMASKRXEV**).
  - **WICINT*** (representing **IRQ**, **NMI**, and **RXEV**).
  - **WICPEND*** (representing **WICPENDISR**, **WICPENDNMI**, and **WICPENDRXEV**).
- The WIC interface signals that are present at the `GREBE`, `GREBEINTEGRATION`, and `GREBEINTEGRATION_MCU` levels. See Table 4-29 on page 4-37
  - **WICENACK.**
  - **WAKEUP.**
  - **WICSENSE.**

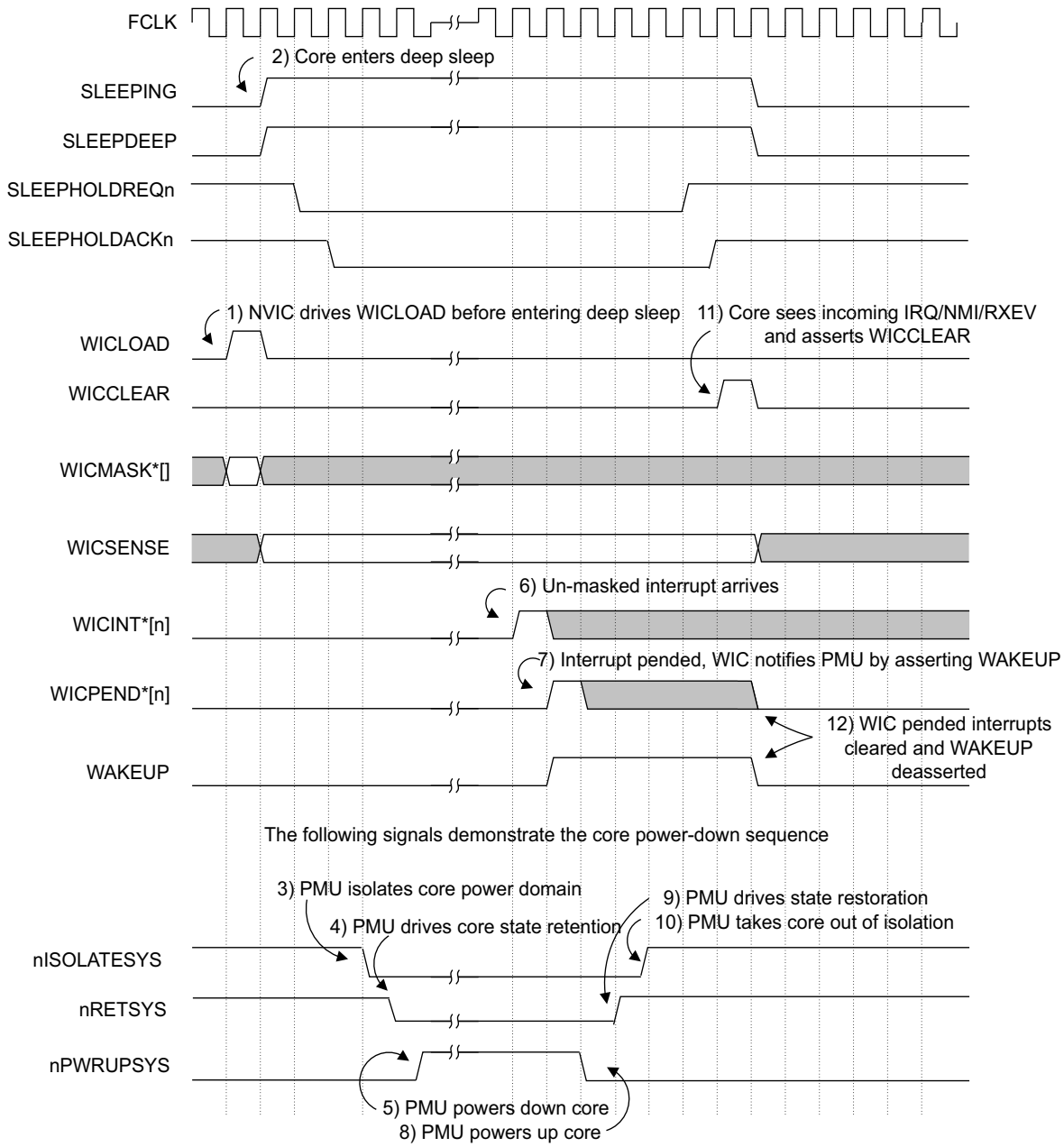


Figure 4-9 Power down timing sequence - legacy interface

4.4.16 Power Management Unit - Q-Channel Interface

The PMU interface is only present at the `GREBEINTEGRATION` and `GREBEINTEGRATION_MCU` levels.

Table 4-29 shows the PMU Q-Channel interface signals.

Table 4-29 PMU Q-Channel interface signals

Name	Direction	Description
SYSQREQn	Input	Q-Channel quiescent request signal.
SYSQACCEPTn	Output	Q-Channel quiescent request accept signal.
SYSQDENY	Output	Q-Channel quiescent request denial signal.
SYSQACTIVE	Output	Q-Channel active signal. **1** Active HIGH signal indicates the SYS Power domain needs to exit the quiescent state. The PMU controller should respond by driving **SYSQREQn** HIGH exiting the quiescent state. **0** In running state, it provides a criterion for initiating a quiescence request.
DBGQREQn	Input	Debug quiescent request signal.
DBGQACCEPTn	Output	Debug quiescent request accept signal.
DBGQDENY	Output	Debug quiescent request denial signal.
DBGQACTIVE	Output	Debug active signal. **1** Active HIGH signal indicates the DBGPower domain needs to exit the quiescent state. The PMU controller should respond by driving **DBGQREQn** HIGH exiting the quiescent state. **Note** When **DBGQACTIVE** is HIGH, **SYSQACTIVE** is also HIGH. **0** In running state, it provides a criterion for initiating a quiescence request.

The PMU Q-Channel interface and the PMU legacy interface are mutually exclusive and cannot be used together. When using the Q-Channel interface, you must ensure that legacy interface is driven in this way:

- Permanently tie LOW the **CDBGPRWUPACK** input.
- Permanently tie HIGH **SLEEPHOLDREQn** input.
- Ignore the **CDBGPRWUPREQ** output.

The processor indicates a power state change request to the PMU using the Q-Channel **SYSQACTIVE** or **DBGQACTIVE**signals.

Warning

If **SYSQACTIVE** or **DBGQACTIVE** are asserted, the PMU must respond by activating the corresponding power domain (if not already active). Failing to do this might result in deadlock.

Table 4-30 lists the cases where **SYSQACTIVE** and **DBGQACTIVE** are asserted, together with the response to powerup and power down requests.

Table 4-30 QACTIVE asserted for powerup and powerdown requests

Domain	QACTIVE asserted	Power up request	Power down request
System	• Core not sleeping. • Core sleeping with a wakeup request. • **DBGQACTIVE** is asserted.	Always accepted	Denied if either of the following is true: • **SYSQACTIVE** is asserted. • **DBGQACTIVE** is asserted. Otherwise accepted.
Debug	• DAP access to the debug domain • DHCSR.C_DEBUGEN bit is set • The DEMCR.TRCENA bit is set • ETM is enabled.	Accepted if the system is already in Q_RUN state. Otherwise the debug power-up request waits (in Q_EXIT state) the system to be in its Q_RUN state.	Denied if the **DBGQACTIVE** is asserted. Otherwise accepted.

If **SYSQACTIVE** or **DBGQACTIVE** are deasserted, the event can be treated as a hint to the PMU that the relative domain in no longer in use and can be deactivated.

If **SYSQACTIVE** and **DBGQACTIVE** are deasserted at the same time, it cannot be guaranteed that both System and Debug domains enter Q_STOPPED state at the same time. It is the responsibility of the PMU to ensure that clock gating and powerdown sequences respect the allowed states. This means that the PMU must ensure that the system domain is not gated or powered down before the debug domain.

In addition, when **SYSQACTIVE** is low, the **DEEPSLEEP** output can be observed by the PMU to decide what level of powerdown is required, such as Retention or powerdown. This does not apply for the DBG domain.

Note

As stated in Table 4-30, any DAP access requires the DBG and the SYS power domains to be switched on.

However, any non-DAP initiated access into the Debug elements in DBG power domain, such as CTI and ETM, are aborted with a **PSLVERR**.

Figure 4-10 on page 4-39 shows the power down timing sequence. It includes:

- The **SLEEPDEEP** signal, which is only used by the PMU for low-power mode.
- The WIC interface signals that are present at the `GREBE` level only. See Table 4-26 on page 4-33
  - **WICLOAD**.
  - **WICCLEAR**.
  - **WICMASK*** (representing **WICMASKISR**, **WICMASKNMI**, and **WICMASKRXEV**).
  - **WICINT*** (representing **IRQ**, **NMI**, and **RXEV**).
  - **WICPEND*** (representing **WICPENDISR**, **WICPENDNMI**, and **WICPENDRXEV**).

- The WIC interface signals that are present at the GREBE, GREBEINTEGRATION, and GREBEINTEGRATION_MCU levels. See Table 4-29 on page 4-37
  - **WICENACK.**
  - **WICSENSE.**
- The Q-Channel PMU interface signals that are present at GREBEINTEGRATION, and GREBEINTEGRATION_MCU, which are **SYSQACTIVE**, **SYSQREQN**, and **SYSQACCEPTN**. See Table 4-29 on page 4-37.

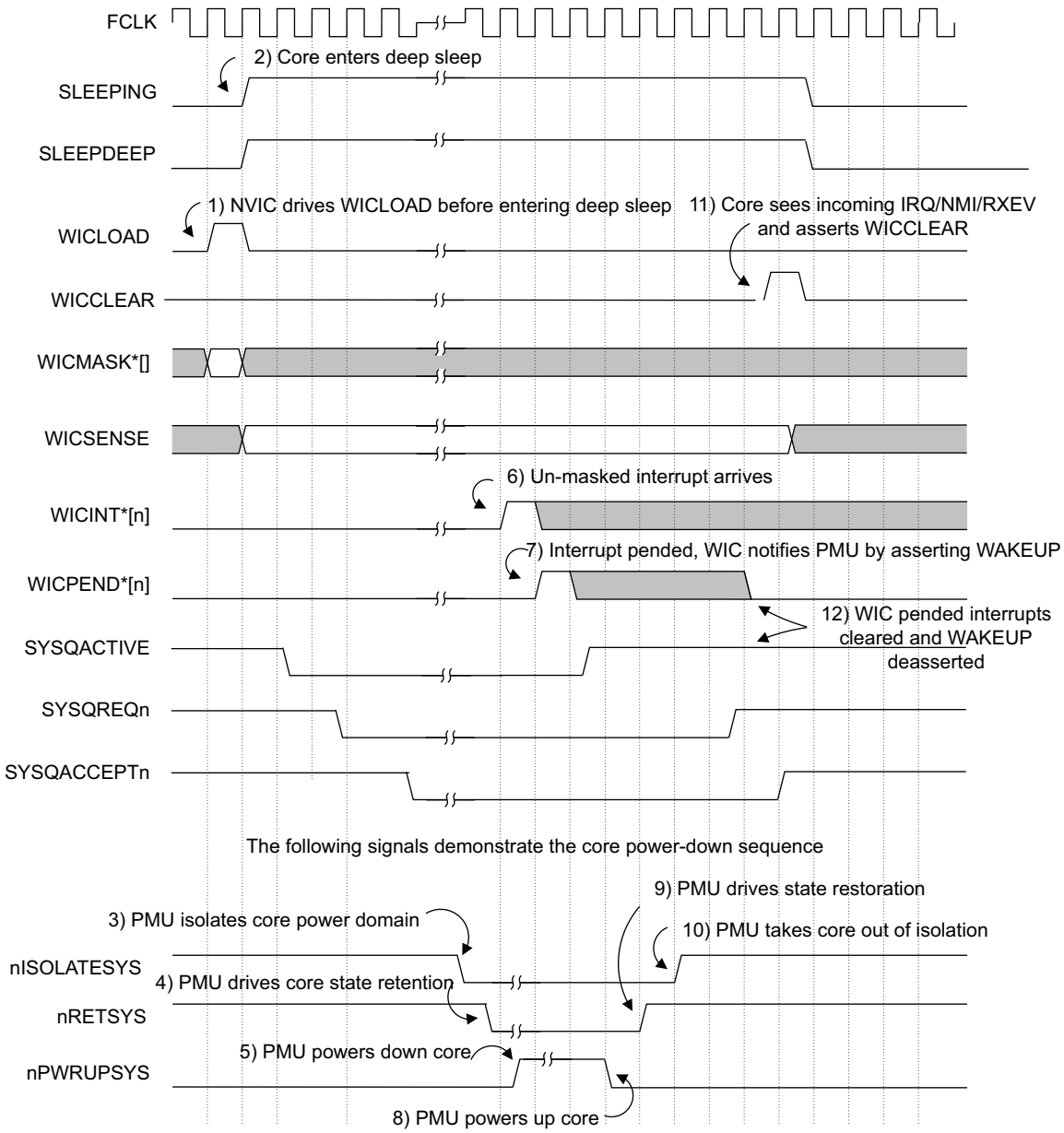


Figure 4-10 Power down timing sequence - Q-Channel interface

4.4.17 Execution trace

The execution trace signals are only present at the GREBE level. These output signals enable you to connect to the optional CoreSight MTB-M23 component to provide execution trace. The CoreSight MTB-M23 is licensed and delivered separately from the Cortex-M23 processor. See the *ARM® CoreSight™ MTB-M23 Technical Reference Manual* for details of the CoreSight™ MTB-M23.

Table 4-31 shows the execution trace interface signals.

Table 4-31 Execution trace signals

Name	Direction	Description
ATOMIC	Output	Instruction address is not instruction-related.
IAEX[30:0]	Output	Instruction address[31:1] in execute.
IAEXEN	Output	Instruction address enable.
IAEXSEQ	Output	Instruction address is sequential.

Note

These outputs are tied LOW if:

- The CoreSight MTB-M23 interface is not implemented.
- Non-invasive debug is not enabled depending on the current security level.

4.4.18 Trace port

Table 4-32 shows the trace port signals.

Table 4-32 Trace port signals

Name	Direction	Description
ATDATAM[7:0]	Output	8-bit trace data.
ATVALIDM	Output	**ATDATAM** is valid.
ATIDM[6:0]	Output	Trace source ID.
ATREADYM	Input	Indicates that the trace port is able to accept the data on **ATDATAM**
AFREADYM	Output	Indicates that the ETM FIFO is empty.

Note

The **AFVALID** functionality is not supported, so the trace port has no way of flushing data from the ETM's FIFO. **AFREADY** is used to indicate that ETM FIFO is empty. If the ETM is used with a TPIU which uses **AFVALID**/**AFREADY**, some external handshaking is needed. In the Cortex-M23 processor, this is done by the grb_dbg_misc module.

4.4.19 Test interface

Table 4-33 shows the functions of the test interface pins.

Table 4-33 Test interface pin functions

Name	Direction	Description
DFTRSTDISABLE	Input	GREBEINTEGRATION and GREBEINTEGRATION_MCU levels only. Drive this signal HIGH to disable the internal reset synchronizers. This enables test pattern generation tools to have controllability of reset flops in the design.
DFTSE	Input	**DFTSE** is the scan-enable input. Drive this signal HIGH during vector scan-in/scan-out, and LOW during normal operation.
DFTCGEN	Input	**DFTCGEN** enables the clock gating cells during scanning in or out of test patterns. Drive this signal HIGH during vector scan-in/scan-out, and LOW during normal operation.

Note

Scan insertion during synthesis introduces one or more *Scan-In* (SI) and *Scan-Out* (SO) pins.

See Chapter 7 *Design For Test* for more information about testing and the testing interface.

4.4.20 MTB SRAM interface

The MTB SRAM interface is only present at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels. See *ARM® CoreSight™ MTB-M23 Integration and Implementation Manual*.

Table 4-34 MTB SRAM interface

Name	Direction	Description
RAMHCLK	Output	MTB RAM clock, optionally gated.
RAMRD [31:0]	Input	MTB RAM read-data.
RAMAD[MTBAWIDTH**-3:0]**	Output	MTB RAM address.
RAMWD[31:0]	Output	MTB RAM write-data.
RAMCS	Output	MTB RAM chip select.
RAMWE[3:0]	Output	MTB RAM byte lane write enables.

4.4.21 MTB trace control

The MTB trace control interface is present at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels. See *ARM® CoreSight™ MTB-M23 Technical Reference Manual*.

Table 4-35 MTB trace control signals

Name	Direction	Description
TSTART	Input	MTB trace start.
TSTOP	Input	MTB trace stop.

The **TSTART** and **TSTOP** inputs can be used to drive the MTB inputs, but CTI can drive them as well, see Appendix G *CTI*.

Table 4-36 MTB miscellaneous signals

Name	Direction	Description
MTBSRAMBASE[31:0]	Input	Location of MTB RAM in memory map

4.4.22 MTB AHB interface

The MTB AHB interface is present at the `GREBEINTEGRATION` and `GREBEINTEGRATION_MCU` levels. See *ARM® CoreSight™ MTB-M23 Technical Reference Manual*.

Table 4-37 MTB AHB Interface

Name	Direction	Description
HADDRM[31:0]	Input	AHB address into MTB.
HPROTM[3:0]	Input	AHB properties into MTB.
HSIZEM[2:0]	Input	AHB transaction size into MTB.
HTRANSM[1:0]	Input	AHB transaction to MTB.
HWDATAM[31:0]	Input	AHB write-data for MTB.
HBURSTM[2:0]	Input	AHB burst type to DCS.
HSELRAMM	Input	AHB selects MTB SRAM.
HSELSFRM	Input	AHB selects special function registers.
HWRITEM	Input	AHB write to MTB.
HREADYM	Input	AHB ready for MTB.
HRDATAM[31:0]	Output	AHB read-data for MTB SRAM.
HREADYOUTM	Output	AHB ready out of MTB.
HRESPM	Output	AHB error response from MTB.

4.4.23 APB MCU interface

The APB MCU interface is present at the `GREBEINTEGRATION_MCU` level. This port can access the ETM, TPIU, and CTI APB slave ports.

Table 4-38 APB MCU interface

Name	Direction	Description
DCLK	Input	Clock. The rising edge of **DCLK** times all transfers on the APB.
nDBGRESET	Input	Reset. The APB reset signal is active LOW.
PSEL	Input	Select. The APB bridge unit generates this signal to target ETM, TPIU, and CTI.
PENABLE	Input	Enable. Indicates the second and subsequent cycles of an APB transfer.
PADDR[19:2]	Input	APB address bus.

Table 4-38 APB MCU interface (continued)

Name	Direction	Description
PPROT[2:0]	Input	Protection control signals.
PREADY	Output	This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
PWRITE	Input	Direction. Indicates an APB write access when HIGH and an APB read access when LOW.
PSTRB[3:0]	Input	Data strobe. This input is not used and can be driven LOW.
PWDATA[31:0]	Input	Write data. The peripheral bus bridge unit drives this bus during write cycles when **PWRITE** is HIGH.
PRDATA[31:0]	Output	Read Data. The selected slave drives this bus during read cycles when **PWRITE** is LOW.
PSLVERR	Output	Slave access error. Indicates that the access failed when HIGH.

4.4.24 APB ETM interface

The APB ETM interface is present at the `GREBEINTEGRATION` level. This port can access the ETM APB slave port.

Table 4-39 APB ETM interface

Name	Direction	Description
DCLK	Input	Clock. The rising edge of **DCLK** times all transfers on the APB.
nDBGRESET	Input	Reset. The APB reset signal is active LOW.
PSELT	Input	Select. The APB bridge unit generates this signal to target ETM.
PENABLET	Input	Enable. Indicates the second and subsequent cycles of an APB transfer.
PADDRT[11:2]	Input	APB address bus.
PPROTT[2:0]	Input	Protection control signals.
PREADYT	Output	This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
PWRITET	Input	Direction. Indicates an APB write access when HIGH and an APB read access when LOW.
PSTRBT[3:0]	Input	Data strobe. This input is not used and can be driven LOW.
PWDATAT[31:0]	Input	Write data. The peripheral bus bridge unit drives this bus during write cycles when **PWRITE** is HIGH.
PRDATAT[31:0]	Output	Read Data. The selected slave drives this bus during read cycles when **PWRITE** is LOW.
PSLVERRT	Output	Slave access error. Indicates that the access failed when HIGH.

4.4.25 APB CTI interface

The APB CTI interface is present at the GREBEINTEGRATION level. This port can access the CTI APB slave port.

Table 4-40 APB CTI interface

Name	Direction	Description
DCLK	Input	Clock. The rising edge of **DCLK** times all transfers on the APB.
nDBGRESET	Input	Reset. The APB reset signal is active LOW.
PSELC	Input	Select. The APB bridge unit generates this signal to target CTI.
PENABLEC	Input	Enable. Indicates the second and subsequent cycles of an APB transfer.
PADDRC	Input	APB address bus.
PPROTC[2:0]	Input	Protection control signals.
PREADYC	Output	This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
PWRITEC	Input	Direction. Indicates an APB write access when HIGH and an APB read access when LOW.
PSTRBC[3:0]	Input	Data strobe. This input is not used and can be driven LOW.
PWDATAC[31:0]	Input	Write data. The peripheral bus bridge unit drives this bus during write cycles when **PWRITE** is HIGH.
PRDATAC[31:0]	Output	Read Data. The selected slave drives this bus during read cycles when **PWRITE** is LOW.
PSLVERRC	Output	Slave access error. Indicates that the access failed when HIGH.

4.4.26 Core ETM interface

Table 4-41 shows the core ETM interface.

Table 4-41 Core ETM interface

Name	Direction	Qualified by	Connection information
ETMIVALID	Input	No qualifier	Instruction in execute is valid. Marks that an opcode has entered the first cycle of execute.
ETMIBRANCH	Input	**ETMIVALID**	Opcode is a branch target. Marks that current code is the destination of a *Program Counter* (PC) modifying event (branch, interrupt processing).
ETMIINDBR	Input	**ETMIBRANCH**	Opcode branch target is indirect. Marks that the current opcode is a branch target whose destination the PC content cannot deduce. For example, LSU, register move, or interrupt processing.
ETMDVALID	Input	No qualifier	Signals that the current data address as seen by the *Data Watchpoint and Trace* (DWT) is valid on this cycle.
ETMICCFAIL	Input	**ETMIVALID**	Opcode condition code fail or pass. Marks if the current opcode has failed or passed its conditional execution check. An opcode is conditionally executed if it is a conditional branch, or for all other opcode found in an IT block.

Table 4-41 Core ETM interface (continued)

Name	Direction	Qualified by	Connection information
ETMINTSTAT[2:0]	Input	No qualifier	Interrupt status. Marks the interrupt status of the current cycle: `000` No status. `001` Interrupt entry. `010` Interrupt exit. `011` Interrupt return. `100` Vector fetch and stack push. **ETMINTSTAT** Entry/Return is asserted in the first cycle of the new interrupt context. Exit occurs without **ETMIVALID**.
ETMINTNUM[8:0]	Input	**ETMINTSTAT**	Interrupt number. Marks the interrupt number of the current execution context.
ETMIA[31:1]	Input	No qualifier	Instruction address. Indicates the current fetch address of the opcode in execution, or the last opcode executed. You can determine the context by examining: • **ETMIVALID**. • **HALTED**. • **SLEEPING**. The ETM examines this net when **ETMIVALID** is asserted. The DWT examines this net for PC samples and bus watching.
ETMFOLD	Input	**ETMIVALID**	Opcode fold. Indicates that an IT opcode has been folded in this cycle. PC advances past the current (16-bit) opcode and the IT instruction (16 bits). This affects the **ETMIA**.
ETMCANCEL	Input	No qualifier	Current opcode in execute has been canceled. Opcodes that are interrupted restart or continue on return to this execution context. These include: • `LDR/STR`. • `LDRD/STRD`. • `LDM/STM`. • `U/SMULL`. • `MLA`. • `U/SDIV`. • `MSR`. • `CPSID`.
ETMISTALL	Input	No qualifier	Indicates that the last instruction signaled by the core has not yet entered execute. If **ETMICANCEL** is asserted with **ETMISTALL**, it indicates that the stalled instruction did not execute and the previous instruction was canceled.
ETMISB	Input	**ETMIVALID**	Instruction is `ISB`.
COREHALT	Input	No qualifier	Core is in debug state.

4.4.27 ETM interface

The ETM interface is present at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels.

Table 4-42 ETM interface

Name	Direction	Description
ETMFIFOFULLEN	Input	Indicates support for FIFOFULL functionality by reading an ETM register.
ETMPWRUP	Output	ETM is enabled.
ETMTRIGOUT	Output	ETM trigger output. **Note** This output is only present at the GREBEINTEGRATION level. It is used by the TPIU element at the GREBEINTEGRATION_MCU level and by the CTI element at the GREBEINTEGRATION level.
TSVALUEB	Input	Global timestamp value.
TSCLKCHANGE	Input	Timestamp clock ratio change.
ETMEN	Output	Set by the trace software tools to ensure that the trace output is enabled from the ETM. This signal can be used to enable the trace port pins to be shared with the GPIO pins under the control logic external to the ETM.
SE	Input	Scan enable. Connect to **DFTSE** that is used for other blocks.
CDSBYPASS	Input	Architectural clock gating bypass. Connect to **DFTCGEN** that is used for other blocks.

4.4.28 ATB interface

The ATB interface is present at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels.

Table 4-43 ATB interface

Name	Direction	Description
FCLK	Input	Global ATB clock.
nDBGRESET	Input	ATB reset.
ATREADYE	Input	ETM is ready to accept data.
ATDATAE[7:0]	Output	Trace data.
ATVALIDE	Output	A transfer is valid during this cycle. If LOW, all other AT signals must be ignored during this cycle.
AFREADYE	Output	ETM FIFO has been flushed.
ATIDE[6:0]	Output	Identifies the source of the trace.
AFVALIDE	Input	ETM FIFO flush request. This input is only present at the GREBEINTEGRATION level.

4.4.29 Cross Trigger Interface

The Cross Trigger Interface is present at the GREBEINTEGRATION and GREBEINTEGRATION_MCU levels. **CTICHIN** and **CTICHOUT** form the cross trigger event interface. If you are implementing a CoreSight debug system with cross triggering support, you should connect **CTICHIN** and **CTICHOUT** to a Cross Trigger Matrix at a higher level in your SoC. You should connect **CTIIRQ[1:0]** back into your system as interrupt sources for debug purposes.

Table 4-44 Cross Trigger Interface

Name	Direction	Description
CTICHIN[3:0]	Input	CTI Channel In.
CTICHOUT[3:0]	Output	CTI Channel Out.
CTIIRQ[1:0]	Output	CTI IRQ Request.

4.5 Security Considerations

If you need to protect your system from unauthorized access there are many considerations which are beyond the scope of this document. However, the following features of the Cortex-M23 processor may be useful when considering how to prevent unauthorized debug access to all or part of your system.

See *Interfaces* on page 4-10 for additional information on the signals described here.

4.5.1 HPROTD

The **HPROTD** input to the Cortex-M23 processor debug slave port is used to indicate the privilege level of the access being made, either into the processor *System Control Space* (SCS) space or through the processor to the AHB-Lite master port or IOP.

The ARMv8-M architecture requires that the processor SCS can only be accessed as privileged. You can use external logic to force all SLV accesses to be unprivileged when you want to block debugger access to the processor and debug registers.

Unprivileged accesses to the SCS generate a fault.

Unprivileged accesses to any other space are dependent on the logic implemented in the system. The system designer may choose to silently ignore writes and return zero data for reads, or may choose to fault all accesses.

4.5.2 HNONSECD

The **HNONSECD** input signal is set LOW for Secure transactions and HIGH for Non-secure transactions. Accessing a peripheral which is Secure only when **HNONSEC** is HIGH may generate a fault, or silently ignores writes and returns zero data.

If the Security Extension is not present, tie this signal HIGH.

Note

Depending on Secure debug authentication, accesses may not be permitted even if **HNONSEC** is LOW.

4.5.3 HPROT and IOPRIV

The **HPROT** signal on the AHB-Lite master and **IOPRIV** signal on the I/O port indicate the privilege of the access being performed. You can factor privilege information into your system and peripherals if you want to prevent unprivileged accesses to some or all memories and peripherals.

4.5.4 IONONSEC and HNONSEC

The **IONONSEC** and **HNONSEC** output signals are set LOW for Secure transactions and HIGH for Non-secure transactions. If Security Extension is not present, this signal is always HIGH.

Note

These signals are HIGH when the processor runs in Secure state and accesses a Non-secure memory region.

4.5.5 HMASTER and IOMASTER

The **HMASTER** signal on the AHB-Lite master and **IOMASTER** signal on the I/O port indicate whether an access originated from the processor (due to the fetch or execution of program code) or from an access on the processor SLV port (for example, from a debugger). You can factor master information into your system and peripherals if you want to prevent SLV or processor accesses to some or all memories and peripherals.

4.6 CoreSight

The Cortex-M23 processor, CTI, TPIU, CoreSight ETM-M23, and CoreSight MTB-M23 are CoreSight compliant devices. Read the following sections if you intend to build a CoreSight compliant debug system that is based on these components.

4.6.1 CoreSight ROM tables

The Cortex-M23 processor includes an internal CoreSight compliant ROM table and debug resources, to enable you to build CoreSight compliant debug systems. If you configure the Cortex-M23 processor to include debug, 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* and the *CoreSight™ Technology System Design Guide* for more information. The Cortex-M23 processor ROM table provides pointers to the memory-mapped SCS and debug resources. See the *ARM® Cortex®-M23 Processor Technical Reference Manual* for more information about the Cortex-M23 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 implement ETM, an additional MPU ROM table is instantiated at `GREBEINTEGRATION_MCU` level to describe the TPIU and link the processor ROM table.

See Chapter 8 *Execution Testbench* for more information:

GREBE

No ROM table.

GREBEINTEGRATION

Processor ROM table.

GREBEINTEGRATION_MCU

Either processor ROM table (no ETM) or MCU ROM table (ETM and TPIU).

4.6.2 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 under which a debugger might need 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 should consider how your AHB-Lite 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 still allowing debug slave port. See Table 4-23 on page 4-27.

Chapter 5
Key Implementation Points

This chapter describes the key implementation points that you must consider when you implement the Cortex-M23 processor. It contains the following sections:

Note

This chapter references steps that are not described in this guide. See the relevant reference methodology documents from ARM for descriptions of these steps.

5.1 About key implementation points

This chapter contains a list of the main points to consider when you implement the Cortex-M23 processor. Read this chapter with the rest of the information in this guide, and your Reference Methodology documentation.

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

5.2 Key implementation tasks

Table 5-1 lists the key tasks for implementation.

Table 5-1 Key implementation tasks

Key task		Description
1.	Select level of hierarchy to implement. This can be either: a. The MCU integration level, `GREBEINTEGRATION_MCU`. b. The integration level, `GREBEINTEGRATION`. c. The processor component level, `GREBE`. d. A higher level in your SoC that includes the Cortex-M23 processor.	See *About configuration guidelines* on page 2-2 and *Configuration options* on page 2-4.
2.	Configure the processor parameters.	See *Configuration options* on page 2-4.
3.	Select appropriate library cells for clock gating and *Clock-Domain Crossing* (CDC) purposes.	See *Other considerations for implementation* on page 5-4.
4.	If you require SRPG, ensure that the implementation level includes pins for power, retention, and isolation control.	See Appendix F *Power Intent*.
5.	If you require SRPG, select appropriate UPF file and library cells for power gating.	See Appendix F *Power Intent*.
6.	Perform synthesis.	See the Reference methodology documents from your EDA tool vendor for information on equivalence checking tools.
7.	Determine optimum floorplan.	See *Considerations for floorplans* on page 6-6.
8.	Perform place and route	
9.	Perform LVS and DRC checks.	
10.	Perform timing verification.	
11.	Perform characterization.	
12.	Run DFT.	See Chapter 7 *Design For Test* and the Reference methodology documents from your EDA tool vendor.
13.	Perform formal verification using logical equivalence checking tools.	See the Reference methodology documents from your EDA tool vendor.
14.	Optionally run the execution testbench tests on the netlist with SDF annotation.	See *Running the testbench with the -netlist option* on page 8-17.
15.	Perform sign-off in accordance with the agreed criteria and your sign-off obligations.	See Chapter 11 *Sign-off*.
16.	Sign off your implementation.	

Note

The outputs from the tasks in Table 5-1 must produce complete and verified deliverables as described in *Requirements for sign-off* on page 11-4.

5.3 Other considerations for implementation

There are points that you must consider when you implement design options that are not covered by the configuration options that are described in Chapter 2 *Configuration Guidelines*.

5.3.1 Clock considerations

Almost all clocks in the Cortex-M23 processor use positive edges. The only exception is **SWCLKTCK** when DAP implements JTAG. In this case, the flops use the negative edges.

5.3.2 Architectural clock gating

The Cortex-M23 processor design instantiates 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 the synthesis tools use during the implementation process. The architectural clock gating cell must be provided as a module named `grb_acg.v`.

Architectural clock gating is optional because correct operation of the processor is not dependent on it. If architectural clock gating is not required:

1. Configure your processor to have the `ACG` parameter set to 0.
2. Ensure that your `grb_acg` module drives the **CLKOUT** output directly from the **CLKIN** input.

If architectural clock gating is required:

1. Configure your processor to have the `ACG` parameter set to 1.
2. Ensure that your `grb_acg` module directly instantiates a positive-edge clock gating cell from your library.
3. Connect the clock inputs and outputs, clock enable and scan enable signals correctly.

The `logical/models/cells/generic/grb_acg.v` file contains a configurable module that you can use for simulation and logical equivalence checking. Do not use this version for synthesis.

5.3.3 Special purpose cells

The Cortex-M23 processor and its associated example system components have several interfaces that require special consideration other than the logical behavior described in the RTL. These include, for example, CDC boundaries and reset synchronizers.

To ensure that all these cases are implemented correctly, any single logic function with special requirements has its own individual cell description. Each cell can be instantiated several times within the design. For each different cell, you must provide a module definition that uses chosen elements from your library that meet the specific cell requirements. The Reference Methodology provides modules for any given library. See the *ARM® Reference Methodology Release Note* for the location of these files.

For each individual cell, there is a reference version that is provided in `logical/models/cells/generic`. This is provided to enable RTL simulation and logical equivalence checking for Sign-off. You must not use these versions for synthesis. The cells that you must provide depend on the components you include in your design. Table 5-2 shows all the cells in the Cortex-M23 processor and its associated example system components.

Table 5-2 Cortex-M23 processor special purpose cells

Component	Cells
GRBDAP[a]	grb_dap_cdc_capt_sync.v
	grb_dap_cdc_comb_and.v
	grb_dap_cdc_comb_and_addr.v
	grb_dap_cdc_comb_and_data.v
	grb_dap_cdc_send.v
	grb_dap_cdc_send_addr.v
	grb_dap_cdc_send_data.v
	grb_dap_cdc_send_reset.v
	grb_dap_jt_cdc_comb_and.v
	grb_dap_sw_cdc_capt_reset.v
	grb_dap_sw_cdc_capt_sync.v
GREBEINTEGRATION	grb_dbg_reset_sync.v[b]
System components	grb_rst_send_set.v
	grb_rst_sync.v
	grb_pmu_acg.v
	grb_pmu_cdc_send_reset.v
	grb_pmu_cdc_send_set.v
	grb_pmu_sync_reset.v
	grb_sync.v
	grb_pmu_sync_set.v
GREBE GREBEINTEGRATION	grb_acg.v

a. This is included if you implement `GREBEINTEGRATION_MCU` configured to include debug.

b. This is only included if configured with debug.

Table 5-3 shows the module logical behavior and required DAP properties for correct CDC and glitch safe operation.

Table 5-3 Required properties for the processor and the DAP

Module	Description
grb_acg	Logically, these modules are architectural clock gates consisting of a latch and an AND gate. You must replace the clock gate simulation model in these modules with an architectural clock gate from your library to avoid glitches on the gated clock output. You must also connect the input module input **DFTCGEN** to the bypass input of the architectural clock gate to ensure that implementation inserted scan chains can be clocked when in shift mode.
grb_dap_cdc_capt_sync	Logically, this module is a two-flop synchronizer, used to sample signals asynchronous to the reference clock. You must choose flops designed to minimize the time required to resolve metastable outputs when the input setup or hold constraints are violated. This module is used only with a Serial Wire implementation of the Cortex-M23 processor DAP with multi-drop.
grb_dap_cdc_send grb_dap_cdc_send_reset grb_dap_cdc_send_addr grb_dap_cdc_send_data	Logically, these modules are sets of launch flops with a synchronous enable input. Modules with a _reset suffix have an asynchronous reset or asynchronous set input that is always used. The other modules have a reset input that only requires connection if the reset all registers option is used at the GREBEINTEGRATION_MCU level. The output of these registers must remain stable around clock edges when the synchronous enable is deasserted, or when the synchronous enable is asserted and the input to be loaded does not logically change the output.
grb_dap_cdc_comb_and grb_dap_cdc_comb_and_addr grb_dap_cdc_comb_and_data grb_dap_jt_cdc_comb_and	Logically, this module is a set of AND gates that mask a vector of input signals, generated in a source clock domain, because they are passed to logic in a destination clock domain, or are connected to logic in the same clock domain at times when the source might be metastable. The default instance is a single AND gate. The _addr and _data provide 4-bit and 32-bit AND masks, respectively. The mask input is driven synchronously to the destination clock domain, and the vector of output signals is sampled synchronously to the destination clock domain. When the mask input is LOW, the module holds the outputs LOW, regardless of the state of the other input signals, even if they are changing or metastable. The design ensures that the mask input is asserted only when the input signals are known to be stable. This module is used only when a JTAG implementation of the Cortex-M23 processor DAP is implemented.

Table 5-3 Required properties for the processor and the DAP (continued)

Module	Description
grb_dap_sw_cdc_capt_reset	Logically, this module is a capture register that has an asynchronous reset. The reset state of this register must be 1 for correct operation. This module samples **SWDITMS** that, during Serial Wire turnaround periods, might have undefined timing relative to **SWCLKTCK**. The output from the flop is used by the Serial Wire DP when **SWDITMS** is known to be stable according to the protocol, and at other times is masked externally through a grb_dap_cdc_and module. You must choose a flop designed to minimize the time required to resolve metastable outputs when the input setup or hold constraints are violated. This module is used only with a Serial Wire implementation of the Cortex-M23 processor DAP without multi-drop.
grb_dap_sw_cdc_capt_sync	Logically, this module implements a two-flop synchronizer, used to sample the **SWDITMS** input that may have arbitrary timing relative to **SWCLKTCK** while the Serial Wire protocol is in dormant state. The Serial Wire DP also uses output from the first flop when **SWDITMS** is known to be stable according to the protocol, and at other times is masked externally through a grb_dap_cdc_and module. The flops in this module must be reset asynchronously to 1 rather than 0. You must choose flops designed to minimize the time required to resolve metastable outputs when the input setup or hold constraints are violated.
grb_dbg_reset_sync	Logically, these modules are three-flop shift registers with an asynchronous reset, where the input of the shift register is connected to logic 1. The purpose of this module is to produce a reset that is asynchronously asserted and synchronously deasserted from a reset that is both asserted and deasserted asynchronously. In the case of grb_rst_sync, the reset must also be asserted synchronously relative to a synchronous, non-glutting reset request input. The final D-type in the synchronizer must be guaranteed to change cleanly, that is, never glitch while reset is held inactive.

Table 5-4 shows the module logical behavior and required PMU and reset controller properties for correct CDC and glitch safe operation.

Table 5-4 Required properties for the example PMU and the reset controller

Module	Parameter	Description
grb_pmu_acg	ACG	Logically, these modules are architectural clock gates consisting of a latch and an AND gate. You must replace the clock gate simulation model in these modules with an architectural clock gate from your library to avoid glitches on the gated clock output. You must also connect the input module input **DFTSE** to the bypass input of the architectural clock gate to ensure that implementation inserted scan chains can be clocked when in shift mode.
grb_pmu_sync_reset	-	Logically, these modules are two-flop synchronizers used to sample signals asynchronous to the reference clock. Modules with a _reset suffix have an asynchronous reset and modules with a _set suffix have an asynchronous set. You must choose flops designed to minimize the time required to resolve metastable outputs when the input setup or hold constraints are violated.
grb_pmu_sync_set	-	

Table 5-4 Required properties for the example PMU and the reset controller (continued)

Module	Parameter	Description
grb_rst_send_set	-	Logically these modules are sets of launch flops with a synchronous enable input. Modules with a _reset suffix have an asynchronous reset and modules with a _set suffix have an asynchronous set. The other modules have a reset input that only requires connection if the reset all registers option is used at the GREBEINTEGRATION level. The output of these registers must remain stable around clock edges when the synchronous enable is deasserted, or when the synchronous enable is asserted and the input to be loaded does not logically change the output.
grb_pmu_cdc_send_reset	-	
grb_pmu_cdc_send_set	-	
grb_rst_sync	-	Logically, these modules are three-flop shift registers with an asynchronous reset where the input of the shift register is connected to logic 1. The purpose of this module is to produce a reset that is asynchronously asserted and synchronously deasserted from a reset that is both asserted and deasserted asynchronously. In the case of grebe_rst_sync, the reset must also be asserted synchronously relative to a synchronous, non-glitching reset request input. The final D-type in the synchronizer must be guaranteed to change cleanly, that is, never glitch while reset is held inactive.
grb_sync	-	The same as grb_rst_sync except that is it used for asynchronous data input.

For each module, the parameter indicates whether the particular instance is required, according to the RTL configuration. When the appropriate parameter is non-zero, the corresponding cells must be present.

Note

You must ensure that the special purpose cells are preserved during synthesis. You must not ungroup or restructure them.

Chapter 6
Floorplan Guidelines

This chapter describes the floorplan that is used as a starting point for your design. It contains the following sections:

6.1 About floorplanning

ARM recommends that you perform the synthesis and layout at the GREBEINTEGRATION level, or a higher level of hierarchy in your SoC.

Figure 6-1 shows the top-level inputs, resources, outputs, and controls and constraints for floorplanning.

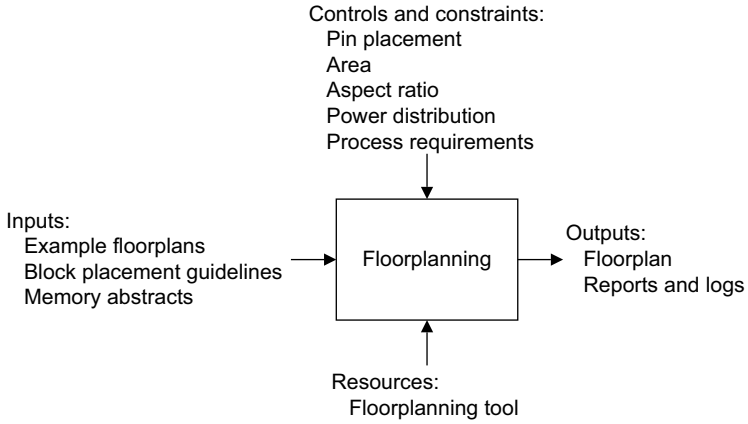


Figure 6-1 Floorplan process

6.2 Resource requirements for floorplanning

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

6.3 Controls and constraints for floorplanning

The aspect ratio of the floorplan depends on the size and number of memories in your system. You must minimize the paths from the processor to your memory blocks and the paths from your memory blocks to the processor. You must also ensure that the standard cells for the processor are grouped to ensure good timing performance. A poor floorplan or incorrectly grouped standard cells reduce the timing performance.

ARM does not expect you to perform hierarchical floorplanning of the processor.

6.4 Inputs to floorplanning

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

- Example floorplans.
- Block placement guidelines.
- Memory abstracts.

6.5 Considerations for floorplans

ARM recommends that you incorporate the power grid in the floorplan passed to your place and route tool so that a more accurate representation of the available routing resource is presented.

You must design the grid to meet the requirements of your library. However, ARM recommends a grid that satisfies an IR drop of 2% for V_{DD} and V_{SS} combined.

6.6 Outputs from floorplans

Output files are specific to your floorplanning tool and might include:

- Logs.
- Reports.
- Floorplan with power grid and pin locations.

Chapter 7
Design For Test

This chapter describes the *Design for Test* (DFT) features of the processor. It contains the following sections:

7.1 About design for test

The processor uses *Automatic Test Pattern Generation* (ATPG) test vectors for production test. To support ATPG, the synthesis tools insert scan chains into the macrocell. See the Reference Methodology documents supplied by your EDA tool vendor for more information.

During synthesis, your synthesis tools might support optional insertion of a test wrapper. This enables the creation of a black-box view of the macrocell, without reducing the testability of the macrocell or your SoC.

The processor is a fully synchronous design with all registers clocked off the rising clock edge, that means you can achieve very high test coverage.

Figure 7-1 shows the top-level inputs, resources, outputs, and controls and constraints for DFT.

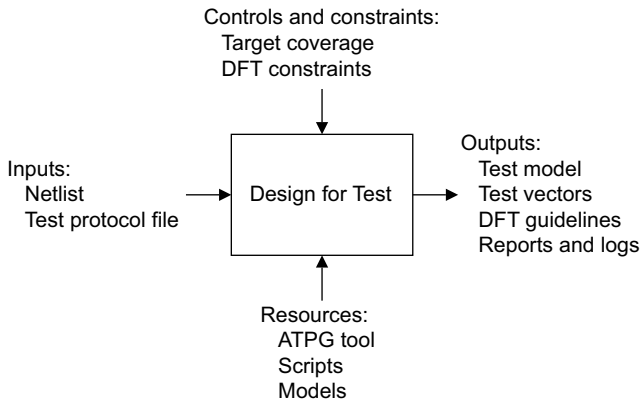


Figure 7-1 Design for Test process

7.2 Requirements for DFT

This guide assumes that you have suitable EDA tools and compute resources for DFT.

7.3 Controls and constraints for DFT

The following sections describe the controls and constraints for DFT:

- *Clock gating*.
- *Reset*.

7.3.1 Clock gating

To ensure that the scan chains operate correctly, without affecting the fault coverage, you must ensure that the clock gating cells are forced to enable when scan enable is asserted. This applies to:

- Clock gates that are inferred in the RTL and inserted automatically by the tools during the implementation flow.
- Architectural clock gates instantiated in the RTL.

Use **DFTCGEN** to enable the clock gating cells during scanning in or out of test patterns.

Drive **DFTCGEN** HIGH during vector scan-in/scan-out, and LOW during normal operation.

7.3.2 Reset

You can use **DFTRSTDISABLE** to disable the reset synchronizers during scan testing.

You must drive **DFTRSTDISABLE** HIGH during the scan part of scan testing, and LOW during normal operation.

Note

This reset disable pin is available only in the `GREBEINTEGRATION` and `GREBEINTEGRATION_MCU` levels.

The `GREBE` level does not implement reset synchronizers.

7.4 DFT features

For more information about the DFT features, see the Reference Methodology documents from ARM.

7.5 Reference data for DFT

The following sections give reference data for DFT:

- *Scan test insertion*.
- *Test wrapper insertion*.

7.5.1 Scan test insertion

Table 7-1 lists the scan test ports that access the internal scan chains in the macrocell.

Table 7-1 Scan test ports

Name	Direction	Description
DFTRSTDISABLE	Input	Disable reset synchronizers during scan testing so test tools have direct control over the resets.
DFTSE	Input	Enable Scan chains. High fan out of this signal means this can be a multicycle path, and cannot be switched at-speed. This signal must be tied LOW during functional mode.
DFTCGEN	Input	Forces clock on architectural clock gating cells. Forces inferred clock gating cells depending on the implementation choice. This signal is viewed as a secondary shift enable in order to improve coverage of clock-gated path. **DFTCGEN** bypasses any architectural clock-gate cell instantiations within the Cortex-M23 processor.

Note

See the Reference Methodology documentation from your EDA tools vendor for details of the scan ports.

7.5.2 Test wrapper insertion

If you implement a test wrapper, it provides production test access to the inputs and outputs of the processor. This gives increased test coverage when access to the primary inputs and outputs is impossible because the macrocell is deeply embedded in your SoC design.

Note

If you use a top-down flow, you do not require a test wrapper because the test tools have complete visibility of all logic paths. Because the processor does not have registered interfaces, ARM recommends that you do not have a test wrapper.

Chapter 8
Execution Testbench

This chapter describes the Cortex-M23 processor *Execution Testbench*. It contains the following sections:

8.1 About the execution testbench

The Cortex-M23 processor execution testbench is provided as a reference to enable easy integration of the Cortex-M23 processor into your system, and contains tests to check that you have performed this integration correctly.

The execution testbench instantiates the supplied GREBEINTEGRATION_MCU level of hierarchy. The execution testbench tests work with all valid configuration options of the Cortex-M23 processor, DAP, and optional CoreSight MTB-M23 and ETM-M23. The execution testbench RTL supports most of these configuration options, and you can modify it to suit your requirements, see *Modifying the execution testbench RTL for your SoC* on page 8-21, *Modifying execution testbench tests* on page 8-28 and *Execution testbench limitations* for more information.

The execution testbench supports both RTL and netlist simulation, and optionally supports *Unified Power Format* (UPF) and SPRG power aware simulation.

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. The test code is designed to be easily modifiable to work in your own system.

The execution testbench is based on a simple example microcontroller, GRBIKMCU, that instantiates the GREBEINTEGRATION_MCU level, ROM, RAM, a PMU, a System Level ROM table, a reset controller, and some *General Purpose Input Output* (GPIO). A second instantiation of the Cortex-M23 processor subsystem, the Debug Driver, is used to generate debug stimulus in the testbench if the configuration includes debug.

Figure 8-1 shows a high-level view of the execution testbench, in which the example microcontroller GRBIKMCU is the device under test. Figure 8-3 on page 8-21 shows a more detailed block diagram of the GRBIKMCU and the execution testbench. Figure E-1 on page E-4 shows a more detailed block diagram of the debug driver, grb_ik_debug_driver, that provides Serial Wire or JTAG stimulus in the execution testbench.

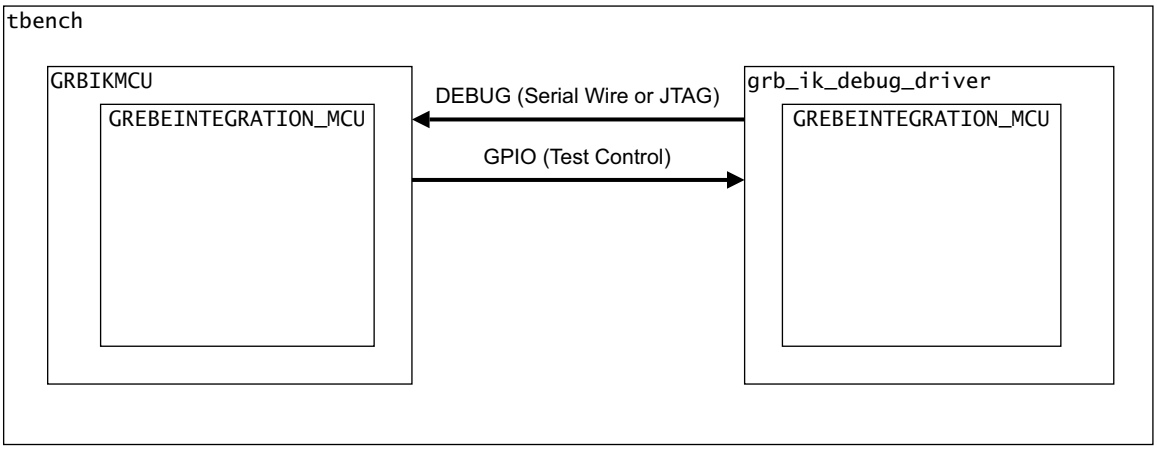


Figure 8-1 Cortex-M23 processor execution testbench overview

8.1.1 Execution testbench limitations

The Cortex-M23 processor execution testbench is designed to be a simple, representative example of a basic Microcontroller. As such, the execution testbench has some known limitations and is not a replacement for other testing. See Chapter 11 *Sign-off* for details of your Sign-off obligations.

The execution testbench tests do not exhaustively test the Cortex-M23 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 GRBIKMCU 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.

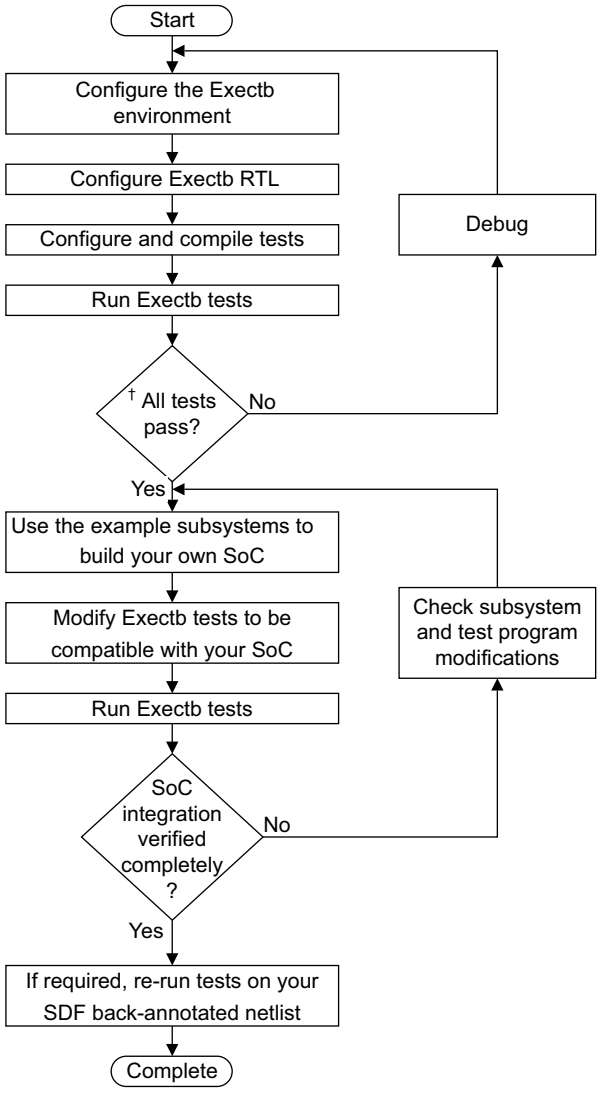
Execution testbench limitations include the following:

- Some implementation parameters do not have a program-visible effect, so cannot be tested in the execution testbench. For example ACG, RAR, AHBSLV, HWF, WIC[1], and WICLINES[1].
- Some tests are timing sensitive to test the effect of implementation parameters such as SMUL, SDIV, WIC, WICLINES.
- Some tests require the presence of at least one external interrupt pin.
- Some pins are tied-off to static values because the GRBIKMCU is designed to be a simple example, so are not tested. For example, all pins that are tied off, such as **IRQLATENCY**, **INSTANCEID**, **CPUWAIT**, QChannel protocol, **INITVTOR**, **INITVTORNS**, Debug authentication interface, and **CFGSECEXT** are tied off to a constant value.
- The execution testbench is designed to test a single processor subsystem such as a microcontroller.
- Some outputs are not verified, such as TPIU outputs, **SWDETECT**, **CURRNS**, and hint signals.

1. The test only checks for illegal values of EXPECTED_WICLINES and warns if the value of EXPECTED_WICLINES is not set for an interrupt that is expected to be implemented.

8.2 Cortex-M23 processor execution testbench flow

The Cortex-M23 processor execution testbench is intended as a platform from which you can develop a SoC incorporating the Cortex-M23 processor. Figure 8-2 shows the Cortex-M23 processor execution testbench flow.



† Depending on the configuration you choose, some tests might skip

Figure 8-2 Cortex-M23 processor execution testbench flow

8.3 Test overview

Table 8-1 shows the execution testbench test programs in the `logical/testbench/execution_tb/tests` directory.

Table 8-1 Execution testbench test programs

Test program	Description
`config_check`	This test verifies that the processor configuration matches the expected configuration values set in the `IKConfig.h` file.
`debug`	This test checks DAP accesses and the pins **LOCKUP**, **EDBGRQ**, **HALTED**, **DBGRESTART**, and **DBGRESTARTED**. **Note** • This test is only useful for implementations that are configured to include debug. • If you run this test on an implementation that is not configured to include debug, the test passes without performing useful work.
`dhry`	This test runs the Dhrystone benchmarking program. The default number of iterations is 5. You can change the number of iterations in one of two ways: • If you build tests using the `tests/Makefile`, edit the value of ITERATIONS there. • If you build tests using MDK-ARM™, right-click **GRBIKMCU - CMxxx** within the Dhrystone project to open the target options, then edit the value of ITERATIONS under the **C/C++** tab.
`etm_trace`[a]	This test generates trace using the ETM and the trace port. Checking is limited to validating the protocol of the generated trace.
`exclusive`	This test demonstrates the use of `LDREX` and `STREX` pairs.
`ext_debug_check`	This test reads and checks the processor ID value using only the DAP Serial Wire or JTAG interface. It does not require the GPIO connection between the MCU and the Debug Driver, and is provided as an example that can be used as the basis of new tests for your SoC. **Note** • This test is only useful for implementations that are configured to include debug. • If you run this test on an implementation that is not configured to include debug, the test passes without performing useful work.
`hello_world`	The processor reads and checks its processor ID and writes to the GPIO registers to print a simple message.
`interrupt`	This test exercises **NMI**, **IRQ**, **TXEV**, and **RXEV**.
`max_power`	This test executes instructions that exercise the Cortex-M23 processor and maximize power consumption. Run this test on your netlist to get power values.
`mpu`	This test sets up several MPU regions and demonstrates the generation of faults from the MPU. **Note** • This test is only useful for implementations that are configured to include the MPU. • If you run this test on an implementation that is not configured to include the MPU, the test passes without performing useful work.
`mtb_trace`[b]	Tests the MTB register access security.

Table 8-1 Execution testbench test programs (continued)

Test program	Description
ppb_iop_check	This tests checks that **IOMATCH** is not asserted for an access to a PPB register. **Note** • This test is only useful for implementations that are configured to include the I/O port. • If you run this test on an implementation that is not configured to include the I/O port, the test passes without performing useful work.
reset	This test checks the **SYSRESETREQ** output and that accesses to the AHB default slave cause a fault.
romtable	This test checks that it is possible for a debugger to autodetect the Cortex-M23 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, their content is displayed, but not checked. **Note** • This test is only useful for implementations that are configured to include debug. • If you run this test on an implementation that is not configured to include debug, the test passes without performing useful work.
security	This test demonstrates security switches.
sleep	This test exercises the sleep modes of the processor, and the **SLEEPING** and **SLEEPDEEP** pins. The test uses an interrupt to wake the processor. If the processor is configured to include debug, the test also wakes the processor from sleep through the DAP.
vtor	This test demonstrates relocating the vector table in the system region. **Note** • This test is only useful for implementations that are configured to include the VTOR. • If you run this test on an implementation that is not configured to include the VTOR, the test passes without performing useful work.

a. Only present if ETM is available.
b. Only present if MTB is available.

8.4 Configuring the testbench

This section describes how to configure the testbench.

The testbench instantiates the GRBIKMCU level of the testbench. See *Testbench structure* on page 8-21 for more information. The testbench supports simulation of one of:

- The Verilog RTL source.
- A netlist.

You can control each type of simulation by editing the corresponding Verilog command file.

Table 8-2 shows the Verilog command files in the logical/testbench/execution_tb/verilog/tbench/verilog/ directory.

Table 8-2 Verilog command files

File	Description
tbench.vc	Used for all simulations. Defines paths to testbench components and controls VCD generation.
rtl.vc	Used for RTL simulations. Defines the paths to the processor Verilog files.
netlist.vc	Used for netlist simulations. Defines the path to your netlist and any required Verilog source files. Modify this file to simulate a netlist.
dsm.vc[a]	Used for DSM models.

a. Only present when DSM is available.

Note

The debug slave protocol checker is supplied in logical/testbench/execution_tb/verilog/tbench/verilog/grb_slvpc.v.

The I/O port protocol checker is supplied in logical/testbench/execution_tb/verilog/tbench/verilog/grb_ioppc.v.

The simple 32-bit to 16-bit downsizing AHB-Lite bridge is supplied in logical/testbench/execution_tb/verilog/tbench/verilog/grb_32to16_dnsize.v.

The block supplied in logical/testbench/execution_tb/verilog/tbench/verilog/tbench.v instantiates:

- u_mcu - Example MCU system
- u_clk_src - Clock source
- u_poreset - Power-on reset generator
- u_dbg_drv - Debug driver

and contains the code for SDF timing annotations and VCD generation.

You can obtain the AMBA® 5 AHB-Lite protocol checker from ARM.

8.4.1 Netlist considerations

To simulate a netlist, you must modify the `netlist.vc` file as described in *Configuring the testbench* on page 8-7. Also, the following defines affect netlist simulations:

- The Verilog define `ARM_GRBIK_SDF`, in the file `logical/testbench/execution_tb/verilog/tbench/verilog/netlist.vc`, determines if delays specified by SDF are included in netlist simulations. Ensure this define is included if you want SDF annotation from netlist simulation. Ensure this define is not included if you do not want SDF annotation from netlist simulation.
- The Verilog define `ARM_GRBIK_NETLIST_IODELAY`, in the file `logical/testbench/execution_tb/verilog/tbench/verilog/tbench.v` points to the appropriate port delay SDF file. You can find example port delay SDF files in `logical/testbench/execution_tb/sdf`.
- The Verilog define `ARM_GRBIK_NETLIST_SDF` points to the SDF file from synthesis flow.
- The Verilog define `ARM_GRBIK_NETLIST_SCOPE` points to the netlist instance.

The input ports to the netlist are delayed using a port delay SDF file. The default delay on an input port is 3ns for the default 50MHz clock. The execution testbench provides two port delay SDF files for use with the `GREBEINTEGRATION` and `GREBEINTEGRATION_MCU` levels. They can be found in `logical/testbench/execution_tb/sdf`. Modify the appropriate file if the input ports for your netlist have changed or if you want to change the default input port delay.

The alternative way to delay input ports to the netlist is to use a Verilog timing wrapper around the netlist. This is not provided in the execution testbench.

8.5 Configuring the execution testbench RTL

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

8.5.1 Configuring the GREBEINTEGRATION_MCU level

The execution testbench instantiates the GREBEINTEGRATION_MCU level of hierarchy in both GRBIKMCU, the device under test, and the debug driver. The debug driver operates correctly with any legal configuration of the RTL parameter values, and does not require any of the optional features of the Cortex-M23 processor.

To configure the GREBEINTEGRATION_MCU level, see *GREBEINTEGRATION_MCU configuration options* on page 2-9 for more information.

8.5.2 Setting the implementation pin options

The execution testbench uses Verilog defines to capture system-specific values for implementation pin options. Table 8-3 lists the defines in logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_defs.v. You must modify the values to match your configuration of the GREBEINTEGRATION_MCU level.

Table 8-3 Implementation pin defines

Define	Description
ARM_GRBIK_STCALIB	SysTick calibration for the Secure SysTick timer if security extensions are included or the SysTick timer if security extensions are not included. See Figure 4-7 on page 4-32 for an asynchronous SysTick clock example.
ARM_GRBIK_STCALIBNS	SysTick calibration for the Non-secure SysTick when Secure and Non-secure SysTick timers are included.
ARM_GRBIK_IRQLATENCY	Interrupt Latency. See *Miscellaneous signals* on page 4-27.
ARM_GRBIK_INSTANCEID	Instance ID for the DAP, see *Miscellaneous DAP signals* on page 4-24.

8.5.3 Setting the execution testbench options

The execution testbench uses Verilog defines to capture system-specific values for the execution testbench options, which are listed in Table 8-4. You must modify the values to match your configuration.

Table 8-4 Execution testbench defines

Define	Description
ARM_GRBIK_CLK_PERIOD	Primary clock cycle in ns.
ARM_GRBIK_POR_CYCLES	Power-On-Reset assertion factor - 3 cycles.
ARM_GRBIK_TIMEOUT_CYCLES	Simulation Runaway Time Out factor - 2 million cycles.
ARM_GRBIK_POR_LENGTH	Power-On-Reset assertion time.
ARM_GRBIK_SIM_TIMEOUT	Timeout.

8.6 Configuring and compiling tests

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

The supplied tests are written in C and must be compiled before you execute them on the Cortex-M23 processor.

The tests have been developed and tested using ARM Compiler 6 running under Linux, and using Keil™ MDK-ARM under Windows. You can download an evaluation version of the:

- ARM Compiler 6, from `https://silver.arm.com`.
- Keil™ MDK-ARM from `http://www.keil.com/arm/mdk.asp`.
- GCC from version 5-2016q1.

Note

See the *ARM® Cortex®-M23 Processor Release Note* for the ARM Compiler and Keil™ MDK-ARM 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.

8.6.1 Execution testbench test configuration

The execution testbench tests check the configuration of the Verilog RTL on which they are running, against the expected values defined in the `IKConfig.h` header file. `IKConfig.h` contains several `EXPECTED_*` defines, for which the variable part is the same as the RTL configuration option. You must ensure that the configuration options match the configuration of your RTL, see *Configuration options* on page 2-4 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 `logical/testbench/execution_tb/tests/IKConfig.h`.

Execution testbench processor configuration options

Table 8-5 shows the execution testbench processor configuration options.

Table 8-5 Execution testbench processor configuration options

Name	Description
EXPECTED_BE	Expected value of BE parameter. See Table 2-1 on page 2-4. **Note** • If the Cortex-M23 processor is in big-endian configuration, you must also compile the execution testbench tests as big-endian. • If you are compiling tests using: — ARM Compiler and the makefile, edit the COMPILE_BE variable defined in logical/testbench/execution_tb/make.cfg. — Keil™ MDK-ARM, consult the MDK-ARM documentation for details of the changes you must make to your project files.
EXPECTED_BKPT	Expected value of BKPT parameter. See Table 2-1 on page 2-4.
EXPECTED_CTI	Expected value of CTI parameter. See Table 2-2 on page 2-7.
EXPECTED_ETM	Expected value of ETM parameter. See Table 2-2 on page 2-7.
EXPECTED_DBG	Expected value of DBG parameter. See Table 2-1 on page 2-4.
EXPECTED_IOP	Expected value of IOP parameter. See Table 2-1 on page 2-4.
EXPECTED_IRQDIS	Expected value of IRQDIS parameter. See Table 2-1 on page 2-4.
EXPECTED_MPU_S	Expected value of the Secure MPU parameter. See Table 2-1 on page 2-4. This applies to the Cortex-M23 processor and the DAP.
EXPECTED_MPU_NS	Expected value of the Non-secure MPU parameter. See Table 2-1 on page 2-4. This applies to the Cortex-M23 processor and the DAP.
EXPECTED_SAU	Expected value of SAU parameter. See Table 2-1 on page 2-4.
EXPECTED_NUMIRQ	Expected value of NUMIRQ parameter. See Table 2-1 on page 2-4.
EXPECTED_SMUL	Expected value of SMUL parameter. See Table 2-1 on page 2-4.
EXPECTED_SYST	Expected value of SYST parameter. See Table 2-1 on page 2-4.
EXPECTED_VTOR	Expected value of VTOR parameter. See Table 2-1 on page 2-4.
EXPECTED_WIC	Expected value of WIC parameter. See Table 2-1 on page 2-4.
EXPECTED_WICLINES	Expected value of WICLINES parameter. See Table 2-2 on page 2-7.
EXPECTED_WPT	Expected value of WPT parameter. See Table 2-2 on page 2-7.
EXPECTED_CSI	Expected CoreSight Integration Level. 0 Absent. 1 Present.
EXPECTED_SECEXT	Expected Security Extension. 0 Absent. 1 Present.

Execution testbench processor tie-off options

Table 8-6 shows the execution testbench processor tie-off options.

Table 8-6 Execution testbench processor tie-off options

Name	Description
EXPECTED_STCALIB	The expected value of **CFGSTCALIB[25:0]**. Ensure that this configuration option matches the tied-off value of the corresponding signal. See *SysTick signals* on page 4-30 for information about how to determine this value for your design.
EXPECTED_STCALIBNS	The expected value of **CFGSTCALIBNS[25:0]**. Ensure that this configuration option matches the tied-off value of the corresponding signal. See *SysTick signals* on page 4-30 for information about how to determine this value for your design.

Execution testbench DAP configuration options

Table 8-7 shows the execution testbench DAP configuration options.

Table 8-7 Execution testbench DAP configuration options

Name	Description
EXPECTED_BASEADDR	The expected value of the DAP CoreSight Component pointer. Ensure that this configuration option matches the tied-off value of the corresponding signal. See *CoreSight ROM tables* on page 4-50 for information about how to determine this value for your design.
EXPECTED_HALTEV	Expected value of HALTEV parameter. See Table 2-2 on page 2-7.
EXPECTED_JTAGnSW	Expected value of JTAGnSW parameter. See Table 2-2 on page 2-7.
EXPECTED_SWMD	Expected value of SWMD parameter. See Table 2-2 on page 2-7.

Execution testbench DAP tie-off options

Table 8-8 shows the execution testbench DAP tie-off options.

Table 8-8 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 *Miscellaneous DAP signals* on page 4-24.
EXPECTED_TARGETID	Expected value of TARGETID parameter. See Table 2-2 on page 2-7.

Execution testbench CoreSight MTB-M23 configuration options

Table 8-9 shows the execution testbench CoreSight MTB-M23 configuration options. If you have not licensed the CoreSight MTB-M23, or do not include it in your configuration, you must set the MTB parameter to 0 in your RTL and in IKConfig.h.

Table 8-9 Execution testbench CoreSight MTB-M23 configuration options

Name	Description
EXPECTED_MTB	Expected value of the MTB parameter. See Table 2-2 on page 2-7.
EXPECTED_MTB_BASEADDR	Expected value of the MTB_BASEADDR parameter. See Table 2-2 on page 2-7.
EXPECTED_MTBSRAM_BASEADDR	Expected value of MTBSRAM_BASEADDR parameter.
EXPECTED_MTB_AWIDTH	Expected value of MTB_AWIDTH parameter.

Execution testbench ECO and revision tie-off options

Table 8-10 shows the execution testbench processor tie-off options.

Table 8-10 Execution testbench ECO and revision tie-off options

Name	Description
EXPECTED_ECOREVNUM	Expected value of **ECOREVNUM**[47:0]. If ECO_REVISION_TEST is ON, then EXPECTED_ECOREVNUM is set to 0x5555555555555 to test the implementation of ECO revision number. See Table 4-23 on page 4-27.

8.6.2 Test program compilation using the ARM or GCC Compiler

This section describes how to compile the test programs with the ARM compiler under Linux using the make command. You might have to modify the make.cfg file in the logical/testbench/execution_tb/ directory to suit your environment, for example:

- The name of the ARM compiler.
- The endianness.
- Message printing.
- Changing the ECO revision number.
- Choosing between the ARM and GCC compiler.

Note

When you use the GCC compiler, the endianness parameter must match the endianness of the compiler.

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.
- Binary .disass files for debug with the tarmac.
- ASCII .inc files that other C program files might include.
- ASCII .rcf (ARM ROM Code) files for use with ARM ROM models.

To compile a test, use the Makefile in execution_tb/tests/ to compile the test programs. The command can be either:

run TETSNAME<test_program> for individual tests.

`make all` for all tests.

where:

`<test_program>` Selects an individual test program to compile. See Table 8-1 on page 8-5 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 run TESTNAME=hello_world
```

Note

- 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 listed in Table 8-1 on page 8-5.
- To remove the compiled tests and prepare for a fresh compilation, type:
  ```
  make clean
  ```

8.7 Running execution testbench tests

The testbench is run from the execution_tb directory. This section contains:

8.7.1 Running the testbench

The run script is provided as an example script to compile the testbench and simulate a specified test.

Type the following command from within the execution_tb/ directory to run the simulation.

```
make run TESTNAME=<test_name> OPT="<options>"
```

where <options> are:

-all	Use this switch to run all execution testbench tests.
-build	This switch is optional. Use it to compile the execution testbench Verilog, including the testbench, before running the test or tests.
-clean	Clean all.
-dsm	Use the DSM for the Cortex-M23 processor as specified in <installation_dir>/integration_kit/logical/tbench/verilog/dsm.vc.
-fsdb	Create the Fast Signal Database for simulation.
-gui	Launches the simulator GUI.
-h	Displays help.
-list	Displays a list of all the tests.
-make	This switch is optional. Use this switch to call make to compile the tests using the ARM Compiler tools or GCC compiler, if these tools are present. If you do not have the ARM Compiler tools or GCC compiler, compile the tests before you run this script.
-mti	Use the MTI simulator. The default is MTI.
-ius	Use the IUS simulator.
-netlist	Use this switch to run the execution testbench with a netlist.
-noexec	Use this switch to only print the commands to the screen.
-sdf <delay type>	Use SDF with the netlist. Choose <delay type> from min or max. If this switch is not specified, the default is zero delay.
-tarmac	Create tarmac. Using this switch for the first time requires the testbench to be recompiled with -build.
-upf	Use UPF for simulation.
-vcd	Enable VCD dumping.

-vcs	Use the VCS simulator.
-64	Use 64-bit simulation mode.

<testname> gives the name of the test image file to run. This file is in the execution_tb/tests directory. For example, if you want to compile the testbench and run the test hello_world.c using the 64-bit VCS simulator, type:

```
make run TESTNAME=hello_world OPT="-vcs -build -64 -make"
```

For more information about supported options, type:

```
make help
```

Running a test creates a directory within execution_tb/, corresponding to the specified simulator:

- MTI.
- VCS.
- IUS.

Ensure that the simulator of your choice is on your path before you execute the test.

Example output

The execution of make run prints a summary at the end unless it is used with the -noexec switch. This is also available in file execution_tb/logs/summary.txt.

An example summary follows. The summary was generated, using the execution testbench unmodified from the configuration in which it is delivered:

```
------------------------------------------------------------

                  Summary of run_etb

------------------------------------------------------------
hello_world      |        PASS |           - |
config_check     |        PASS |           - |
dhry             |        PASS |           - |
max_power        |        PASS |           - |
interrupt        |        PASS |           - |
reset            |        PASS |           - |
debug            |        PASS |           - |
romtable         |        PASS |           - |
mpu              |        PASS |           - |
vtor             |        PASS |           - |
ppb_iop_check    |        PASS |        SKIP |
exclusive        |        PASS |           - |
security         |        PASS |           - |
mtb_trace        |        PASS |        SKIP |
etm_trace        |        PASS |        SKIP |
------------------------------------------------------------
TOTAL PASSES :: 17
TOTAL FAILURES :: 0
TOTAL OVL FAILURES :: 0
TOTAL KILLED :: 0
TOTAL TESTS :: 17
------------------------------------------------------------
```

8.7.2 Running the testbench with the -dsm option

To use a DSM, you first need to generate a DSM corresponding to your configuration, see Chapter 10 *DSM Generation*, or use the provided one with the default configuration.

To use the provided one, go to directory `logical/grebe/dsm`, and use one of:

- `DSM_DEFAULT_32bit/GREBE_DSM.sh` for a 32-bit machine.
- `DSM_DEFAULT_64bit/GREBE_DSM.sh` for a 64-bit machine.

Note

You need to accept the license agreement before starting using a DSM.

The corresponding DSM will be extracted to directory:

- `grebe/simulation_models/linux_32bit/GREBE_DSM_r1p0` for a 32-bit machine.
- `grebe/simulation_models/linux_64bit/GREBE_DSM_r1p0` for a 64-bit machine.

If you use the Verilog PLI interface, adjust the `verilog/tbench/verilog/dsm.vc` file to point:

- Either to `$DSM_MODEL_PATH/GREBE_DSM.v` if you use the Verilog PLI interface.
- Or to `$DSM_MODEL_PATH/GREBE_DSM.sv` if you use the System Verilog DPI interface.

Note

Using the DPI interface is recommended as it simulates faster.

Make the following adjustment to the `make.cfg` file:

Set the `DSM_MODEL_DIR` to point to the location on the DSM model.

For example if you extracted the default 64-bit DSM, set:

```
DSM_MODEL_DIR = <GREBE_install_dir>/grebe/simulation_models/linux_64bit/GREBE_DSM_r1p0
```

Run a test in the normal way, for example:

```
make run TESTNAME=dhry OPT="-mti -dsm -tarmac -64"
```

Note

You need to set the `-64` and `-tarmac` options to correspond to the DSM that you use. The default DSM has tarmac support.

8.7.3 Running the testbench with the -netlist option

To run the testbench with the `-netlist` option, ensure that:

- The file `logical/testbench/execution_tb/verilog/tbench/verilog/netlist.vc` points to the netlist at the required level.
- The instantiation of the netlist in the execution testbench does not take parameters. You have to remove any parameters passed to the netlist instance by editing the instantiation within the file `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_sys.v`.

  Note

  The debug driver block also instantiates the netlist. You must edit the instantiation within the file `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_debug_driver.v`
- The file `logical/testbench/execution_tb/verilog/tbench/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 *Configuring the testbench* on page 8-7.

If the netlist is run with SDF annotation, ensure that the appropriate Verilog defines are correctly set. See *Netlist considerations* on page 8-8.

8.7.4 VCD generation

The execution testbench can be used to generate VCD files. To enable this, ensure that:

- Verilog define `ARM_GRBIK_VCD` in `logical/testbench/execution_tb/verilog/tbench/verilog/tbench.vc` is defined.
- Verilog defines `ARM_GRBIK_VCD_START` and `ARM_GRBIK_VCD_STOP` in `logical/testbench/execution_tb/verilog/tbench/verilog/tbench.vc` are set appropriately. There is no default value.

  Note

- When running Dhrystone with tarmac, the testbench automatically sets the `ARM_GRBIK_VCD_START` and `ARM_GRBIK_VCD_STOP` to the first loop of Dhrystone.

- Verilog define `ARM_GRBIK_VCD_FILE` in `logical/testbench/execution_tb/verilog/tbench/verilog/tbench.v` is defined.

8.7.5 Simulation logs

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 specified by `ARM_GRBIK_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-M23 processor, it passes but prints the following message in the simulation log:

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

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

8.7.6 Modifying the run_etb script

You can modify the `run_etb` script to include any extra simulator options you want to use. The `run_etb` script is located in the `logical/testbench/execution_tb/tools directory.`

You must modify the `run_etb` script if you want to include new tests when you use the `-all` switch for batch testing.

The testbench makefile calls this script to execute the tests.

8.7.7 Tarmac

For more information about the tarmac, see `logical/testbench/execution_tb/README`

8.8 Debugging failing tests

This section describes what you can do to help diagnose problems when tests fail or do not complete on hardware or testbench simulations.

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 provided by the debugger to identify where the test fails.

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.

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 `IKConfig.h`. If this test is failing, you must resolve the issues, because other tests assume that the `EXPECTED_*` values in `IKConfig.h` are correct. Check `logs/summary.txt`.

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 `IKConfig.h` match the configuration of the processor.

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 setting `GRBIKMCU_PRINTF` to `0` in `make.cfg` before compiling the tests. You might want to do this to reduce the runtime of the tests.

Warning

You must recompile the test files after performing a make clean to remove the files before regenerating them.

If a test is failing, enable message printing by defining `GRBIKMCU_PRINTF`, for example by setting it to `1` in `make.cfg`, and then recompile and rerun the test to help determine the reason for failure.

You can also enable message printing from the debug driver module by setting `DEBUGDRIVER_PRINTF` to `1` in `make.cfg` 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`, `debug`, and `sleep`.

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 additional calls to `printf()` into the code to help determine where the test is failing.

Note

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

8.9 Modifying the execution testbench RTL for your SoC

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

8.9.1 Testbench structure

Figure 8-3 shows the testbench structure and its instantiated execution testbench components.

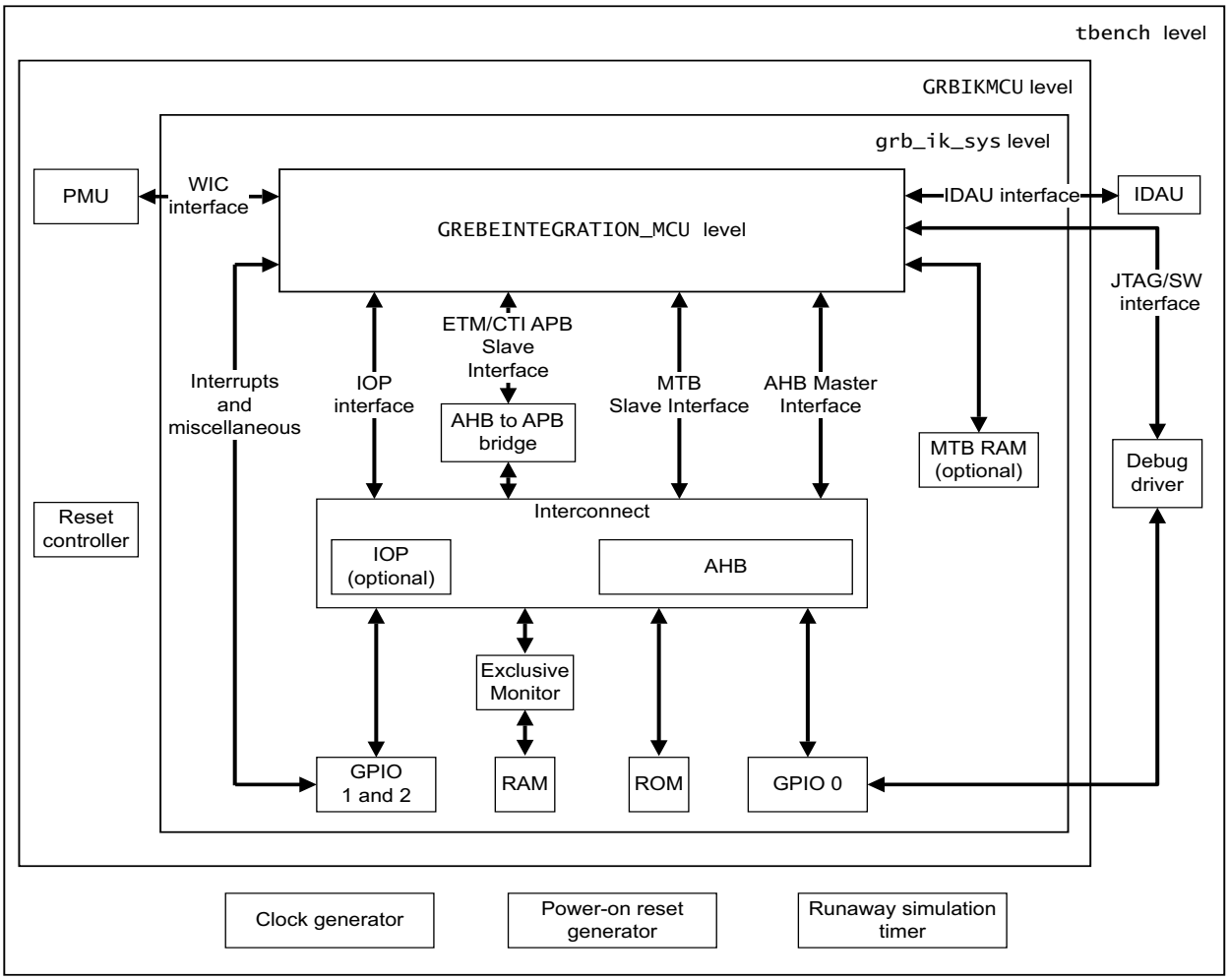


Figure 8-3 Execution testbench

The execution testbench contains these levels:

- *GREBEINTEGRATION_MCU level* on page 8-22.
- *grb_ik_sys level* on page 8-22.
- *GRBIKMCU level* on page 8-22.
- *tbench level* on page 8-22.

The `execution_tb/` path in Figure 1-6 on page 1-12 shows the supplied execution testbench directory structure.

GREBEINTEGRATION_MCU level

This level instantiates GREBEINTEGRATION level DAP, TPIU, and the APB interconnect.

grb_ik_sys level

This section describes the grb_ik_sys level and its components. The grb_ik_sys level contains:

- *GPIO* on page 8-25.
- *AHB default slave* on page 8-26.
- *ROM controller* on page 8-25.
- *ROM* on page 8-25.
- *SRAM controller* on page 8-25.
- *SRAM* on page 8-25.
- *AHB bus interconnect* on page 8-26.
- *IOP bus interconnect* on page 8-26.
- *Exclusive monitor* on page 8-26
- *IDAU* on page 8-26
- MTB SRAM.
- Miscellaneous logic used for integration test purposes.

grb_ik_sys allocates specific functions to the following GPIO signals:

General Purpose Input Output 0

This GPIO has all 32 I/O pins 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 Serial Wire or JTAG interface to the DAP using the debug driver block in the testbench.

See Appendix B *GRBIKMCU GPIO Integration*.

General Purpose Input Output 1

This GPIO drives signals in the example GRBIKMCU, for test purposes only. It drives PMU signals and other processor signals.

See Appendix B *GRBIKMCU GPIO Integration*.

General Purpose Input Output 2

This GPIO drives signals in the example GRBIKMCU, 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.

See Appendix B *GRBIKMCU GPIO Integration*.

GRBIKMCU level

The GRBIKMCU level is a simple example MCU and contains:

- *grb_ik_sys level*.
- *PMU* on page 8-27.
- *Reset controller* on page 8-27.

tbench level

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

- *GRBIKMCU level*.

- *Debug driver* on page 8-27.
- Modules specific to tbench level:

  Clock generator

  This generates clocks for the GRBIKMCU level.

  Power-on reset generator

  This generates power-on reset.

  Runaway simulation timer

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

8.9.2 GRBIKMCU memory map

Figure 8-4 shows the GRBIKMCU memory map.

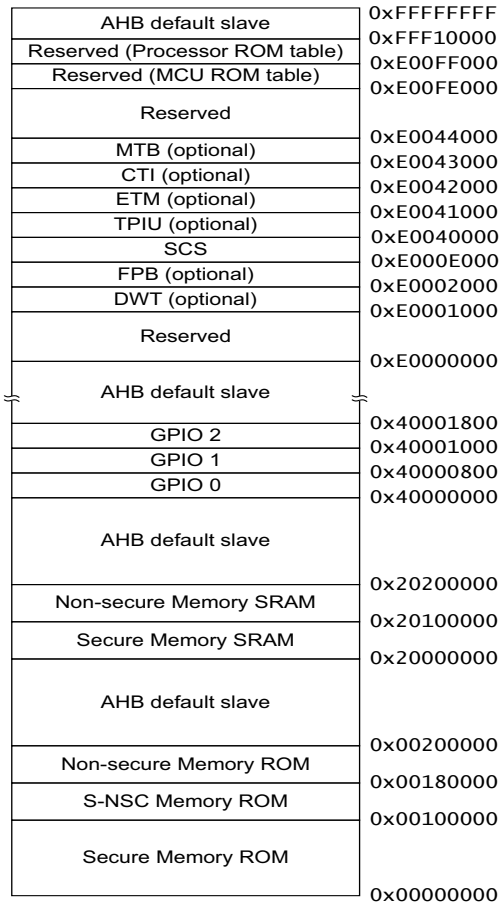


Figure 8-4 GRBIKMCU memory map

The GRBIKMCU memory map is characterized by the following:

- GRBIKMCU instantiates 4MB of physical memory. It is split into two blocks of 2MB:
  - The first 2MB, typically flash in an MCU, is read-only memory that holds the code. It is further split to a Secure code section (1MB) for Secure code, a Secure Non-secure callable section (500KB) for Secure code that can be called from Non-secure and a Non-secure code section (500KB) for Non-secure code.

  - The second 2MB, typically SRAM, is where the stack and heap are located.

- There are three GPIOs, each allocated 2KB of space.

  When the execution testbench includes IOP, the GPIO addresses map to IOP GPIO instead of AHB GPIO.

- The *Private Peripheral Bus* (PPB) space of the Cortex-M23 processor is 1MB.
- The processor level ROM table occupies 4KB of space.
- The optional MCU ROM table occupies 4KB of space..
- 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.

8.10 Execution testbench components

This section describes the components supplied with the execution testbench and instantiated by the `GREBEINTEGRATION_MCU` or `grb_ik_sys` levels.

Note

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

8.10.1 GPIO

The GPIO modules, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_ahb_gpio.v` and `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_iop_gpio.v`, are example general-purpose I/O devices. 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*.

8.10.2 ROM controller

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

8.10.3 ROM

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

Note

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

8.10.4 SRAM controller

The SRAM controller model, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_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.

8.10.5 SRAM

The SRAM model, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_sram.v`, is a synchronous SRAM.

8.10.6 AHB bus interconnect

The AHB bus interconnect module, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_ahb_interconnect.v`, is an example module implementing AHB address decoding and multiplexing.

The single slave port is connected to the AHB-Lite master interface from the `GREBEINTEGRATION` level.

The master ports are connected to the ROM, the RAM, GPIO 0, GPIO 1, GPIO 2, the MCU ROM table, TPIU, ETM, CTI, MTB SFR, MTB SRAM, and the AHB default slave respectively.

8.10.7 IOP bus interconnect

The IOP bus interconnect module, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_iop_interconnect.v`, contains logic for driving **IOMATCH** and **IORDATA** to the processor I/O port. This block exists only if the execution testbench is configured to include the I/O port.

8.10.8 Exclusive monitor

The exclusive monitor module, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_excl_monitor.v` shows an example of global exclusive monitor. Only the SRAM supports the exclusive monitor. All other components return exclusive fail when they try to execute an `STREX`.

8.10.9 IDAU

The IDAU module, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_idau.v` drives the IDAU interface to force the split between Secure and Non-secure regions in the default memory map.

8.10.10 AHB default slave

The AHB default slave, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_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.

8.10.11 WIC

The *Wake-up Interrupt Controller* (WIC) module, `logical/grb_integration/verilog/grb_wic.v`, is an example module that is external to the Cortex-M23 processor and enables the processor to be woken up from deep-sleep.

To use the WIC, you must set the `WIC` configuration parameter. If the `WIC` parameter is set, the processor is configured to include the WIC interface and the WIC module is instantiated in the example integration level `GREBEINTEGRATION.v`.

The WIC includes two parameters `IRQDIS` and `WICLINES` which correspond to the Cortex-M23 processor parameters of the same name.

The example WIC is a synchronously clocked design clocked by the free-running clock **FCLK**. See *WIC interface* on page 4-33 if you plan to build a WIC that is not clocked by **FCLK**.

8.10.12 PMU

The PMU, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_pmu.v`, is an example system power controller that:

- Includes the three distinct power domains and manages the control for:
  - An always-on power domain containing the WIC and part of the DAP.
  - A debug power domain containing the processor debug resources and the rest of the DAP.
  - A system power domain containing the rest of the processor and the NVIC.
- Interfaces with the WIC, to ensure that powerdown and wake-up behaviors are transparent to software.
- Generates clocks for use by the processor, WIC and the memory system compatible with the clocking, power-domain and sleeping requirements.
- Interfaces with the reset controller to meet the power control reset requirements.
- Interfaces with the processor to manage extended sleep and ensure clean powerdown.

8.10.13 Debug driver

The Debug driver module, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_debug_driver.v`, is provided to run tests on a Cortex-M23 processor-based subsystem that is used in the execution testbench to provide Serial Wire or JTAG debug stimulus to `GRBIKMCU`. See Appendix E *Debug Driver*.

8.10.14 Reset controller

The Reset controller, `logical/testbench/execution_tb/verilog/grb_ik_mcu/verilog/grb_ik_rst_ctl.v`, is an example module that controls the various reset signals in accordance with the requirements described in *Resets* on page 4-7.

8.10.15 Downsizer

The Downsizer module, `logical/testbench/execution_tb/verilog/tbench/verilog/grb_32to16_dnsize.v`, is an example 32bit to 16bit AHB-Lite downsizer.

You can use the downsizer example in applications where the Cortex-M23 processor must be interfaced to a 16-bit memory, for example flash memory. The downsizer breaks 32-bit accesses into 16-bit accesses suitable for the narrow memory.

Note

Using 16-bit memory with the 32-bit Cortex-M23 processor incurs a performance penalty. If a 16-bit memory must be used, and the processor might execute code from that memory, the processor should be configured for *Half Word Fetching* (`HWF=1`). This minimizes the performance penalty seen for code fetching.

8.11 Modifying execution testbench tests

The supplied test programs rely on the execution testbench GPIO to report test status, generate interrupts, and monitor status signals. To run the test programs in a custom environment, modify the code to:

- Report test passed or failed.
- Use available peripherals to generate external interrupts.
- Define appropriate exception handlers for the system.

Test programs are supplied as 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.

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 processor. You must adapt the integration test programs to make use of 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 *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-M23 processor that is sufficient to support the supplied tests. See `https://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php` for the full version of the CMSIS and for other embedded software development resources.

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-M23 processor-based device.

Table 8-11 shows the files in the `logical/testbench/execution_tb/tests` and `logical/testbench/execution_tb/tests/CMSIS` directories that constitute the CMSIS.

Table 8-11 CMSIS files

File	Supplied by	Description
`tests/Device/ARM/grbikmcu/Source/ARM/boot_grbikmcu.c`	ARM	This file provides the stack and heap initialization, vector table and default exception handlers. `boot_grbikmcu.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_grbikmcu.c`.
`tests/Device/ARM/grbikmcu/Source/ARM/armclang_grbikmcu.scat`	ARM	Linker script to configure Secure memory regions for the ARM compiler.
`tests/Device/ARM/grbikmcu/Source/ARM/armclang_grbikmcu_ns.scat`	ARM	Linker script to configure Non-secure memory regions for the ARM compiler.

Table 8-11 CMSIS files (continued)

File	Supplied by	Description
tests/CMSIS/Include/core_cm23.h	ARM	Defines the structures for register access of the Cortex-M23 processor. It also contains certain checker for the configuration of processor.
tests/CMSIS/Include/cmsis_armcc.h	ARM	Defines the Cortex-M core register and instruction access functions for ARM compiler version 4/5.
tests/CMSIS/Include/arm_compiler.h	ARM	Defines the macros and the include file in function of the compiler and its version.
tests/CMSIS/Include/tz_context.h	ARM	Defines the Security Extension function.
tests/CMSIS/Include/cmsis_armclang.h	ARM	Defines the Cortex-M core register and instruction access functions for ARM compiler 6.
tests/CMSIS/Include/cmsis_gcc.h	ARM	Defines the Cortex-M core register and instruction access functions for GNU compiler.
tests/Device/ARM/grbikmcu/Source/GCC/sys/custom_file.h	Device vendor	Retargets the file structure.
tests/Device/ARM/grbikmcu/Include/grbikmcu.h	Device vendor	Device-specific file that defines the peripherals for the GRBIKMCU example microcontroller device.
tests/Device/ARM/grbikmcu/Include/ARMCM23.h	Device vendor	Device-specific file that defines the peripherals for the Cortex-M23 processor example device.
tests/Device/ARM/grbikmcu/Include/ARMCM23_TZ.h	Device vendor	Device-specific file that defines the peripherals for the Cortex-M23 processor with Security Extension example device.
tests/Device/ARM/grbikmcu/Source/GCC/startup_grbikmcu.c	Device vendor	Provides the stack and heap initialization, vector table and default exception handlers. This file is provided as an example of a boot file written entirely in C for the GCC compiler.
tests/Device/ARM/grbikmcu/Include/template/partition_ARMCM23.h	Device vendor	Contains an example of CMSIS-Processor initial setup for Secure/Non-secure zones for the Cortex-M23 processor.
tests/Device/ARM/grbikmcu/Include/system_ARMCM23.h	Device vendor	Header file that provides device-specific configuration for the Cortex-M23 processor microcontroller device.

Table 8-11 CMSIS files (continued)

File	Supplied by	Description
tests/Device/ARM/grbikmcu/Source/system_ARMCM23.c	Device vendor	C file that provides device-specific configuration for he Cortex-M23 processor microcontroller device.
tests/Device/ARM/grbikmcu/Source/GCC/gcc_arm_grbikmcu.ld	Device vendor	Linker script to configure Secure memory regions.
tests/Device/ARM/grbikmcu/Source/GCC/gcc_arm_grbikmcu_ns.ld	Device vendor	Linker script to configure Non-secure memory regions.

Table 8-12 shows the test support files in the logical/testbench/execution_tb/tests directory.

Table 8-12 Test support files

Filename	Description
IKtests.h	This header file describes and defines the allocation of **GPIO** pins used by the GRBIKMCU in the execution testbench. It also declares the function prototypes the execution testbench tests use to communicate with the debug driver.
IKtests.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.
IKConfig.h	This file includes some defines that you must edit to match the implemented configuration of the execution testbench.
Makefile	This file enables you to build the execution testbench tests using the ARM Compiler or GCC toolchain.
retarget_grbikmcu.c	This file implements the functions necessary to retarget the C-library printf() function output to the GRBIKMCU **GPIO** pins.

Chapter 9
Netlist Dynamic Verification

This chapter describes how to test the functionality of your implementation of the processor. It contains the following section:

- *Netlist dynamic verification* on page 9-2.

9.1 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 8 *Execution Testbench* for more information.

Chapter 10
DSM Generation

This chapter describes how to generate a *Design Simulation Model* (DSM). It contains the following sections:

10.1 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 similar to the RTL. But it might be slower when using the Verilog PLI interface depending on the simulator selected.

You can optionally generate a DSM of your configured Cortex-M23 processor for in-house use or to provide to your customers.

A script is provided to allow you to generate and test your Cortex-M23 processor DSM. Generation of both 32-bit and 64-bit Linux DSMs is supported.

The DSM generation script uses the integration kit to validate the model.

10.2 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®-M23 Processor 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`

- Make sure that the 32-bit Google protocol buffers, that are required for tarmac trace, are installed on your system.

  The `grebe_tarmac_decode` binary included in the Cortex-M23 processor deliverables requires external shared libraries to execute, in particular, the Google protocol buffers runtime, `libprotobuf.so`, and other standard libraries including:

  — `libz.so`

  — `libpthread.so`

  — `libm.so`

  — `libstdc++.so`

  Note

  You require 32-bit versions of these libraries. They are usually installed as compatibility libraries on 64-bit Linux distributions.

  If your system does not have the protocol buffers library, you can download and install it. See the *ARM® Cortex®-M23 Processor Release Note* for the version of protocol buffers supported:

  ```
  $ wget http://protobuf.googlecode.com/files/protobuf-<version>.tar.gz
  ```

  You can use `sha1sum` to confirm the checksum of the download:

  ```
  $ sha1sum  protobuf-<version>.tar.gz <version checksum>
  $ tar xzf protobuf-<version>.tar.gz
  $ cd protobuf-<version>
  $ ./configure --with-zlib --prefix=<prefix> CXX='g++ -m32'
  $ make install
  ```

  Then add `<prefix>/lib` to your `LD_LIBRARY_PATH` environment variable.

  Building the protocol buffer library as required for the tarmac flow depends on the `zlib` library and header files. In Red Hat Linux distributions, these are provided as part of the `zlib-devel` package.

- See the *ARM® Cortex®-M23 Processor Release Note* for details of the GCC, Python, and Perl versions required by the DSM generation flow.
- The DSM models have been tested with the simulator versions supported by the Cortex-M23 processor deliverables, as described in the *ARM® Cortex®-M23 Processor Release Note*.

10.3 Building and testing a DSM model

To generate, test, and package a DSM model of your Cortex-M23 processor configuration:

1. Configure the Cortex-M23 processor as described in Chapter 2 *Configuration Guidelines*.
2. Run the execution testbench test programs on the RTL and check they pass, as described in Chapter 8 *Execution Testbench*.
3. Build and test the DSM model using the generation script `BuildGREBE_DSM.pl`, for example:

   ```
   $ cd logical/grebe/dsm

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

   The command-line options that are used in the example generates a DSM using the Cortex-M23 processor configuration specified in `logical/config/grebe_defs.v`.

   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 `GREBE_DSM_<name of your configuration>_README.txt` in the `logs/GREBE_DSM_<config>_linux_<32|64>bit` 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.

10.4 DSM generation script command-line options

The build options are controlled with the `BuildGREBE_DSM.pl` script command-line options described in Table 10-1. You must run the script without modification.

You must specify a vendor name and a configuration name, see Table 10-1.

Table 10-1 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 *DEFAULT* for the default configuration	A configuration name must be specified to generate a DSM model

10.4.1 Simulation options

The simulators that validate your model are shown in Table 10-2.

Table 10-2 DSM simulation options

Command-line options	Description	Notes
`-sim=mti`	A mentor MTI simulator using Verilog DPI	• If no `-sim` option is specified, all simulation options are tested. • 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 DPI	
`-sim=ius`	A cadence IUS simulator using Verilog DPI	

10.4.2 Operating system architecture

The operating system architectures are shown in Table 10-3.

Table 10-3 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	

10.5 Command-line example

Example 10-1 64-bit Linux

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

Generates a DSM for your Cortex-M23 processor configuration specified in `logical/config/grebe_defs.v` that supports the default OS architecture, 64-bit Linux.

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

10.6 If DSM generation fails

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

- Model not generated.

  If the message

  ```
  -I- BuildGREBE_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 8G of memory.

  — If you are using a job submission system, increase the runtime limits of the job.

- `config_check` test fails.

  Check that the processor configuration parameter settings in

  `logical/config/grebe_defs.v`

  match the configuration settings in

  `logical/testbench/execution_tb/tests/IKConfig.h`

- Simulation fails:

  — If you are using a job submission system, check your job runtime limits are sufficient.

  — Check the simulation and compile log files that are stored in `logical/grebe/dsm/logs`

  — If the simulation fails to complete, view the log files for the last simulator option located in `logical/grebe/dsm/logs/GREBE_DSM_<config>_linux_<32|64>bit/sim_logs`

  — Check that all the tests have passed on the RTL.

10.7 Generation directory structure

The DSM generation script and associated files are stored in the directory structure shown in Figure 10-1.

The log files from the DSM generation processes are stored in the `logs` directory.

The final DSM deliverable files are in a tar file stored in the `release` directory.

The `BuildGREBE_DSM.pl` script creates the directory structure shown in Figure 10-1.

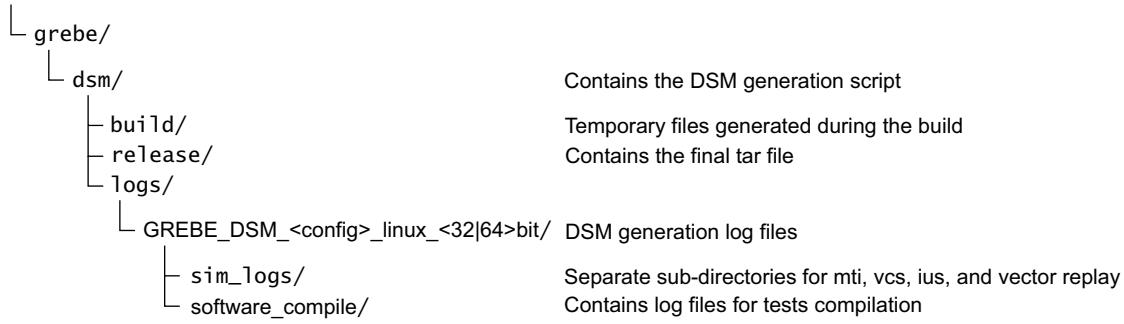


Figure 10-1 Generation directory structure

10.8 Deliverable directory structure

The BuildGREBE_DSM.pl script packages the DSM into a tar file ready for release that has the structure shown in Figure 10-2.

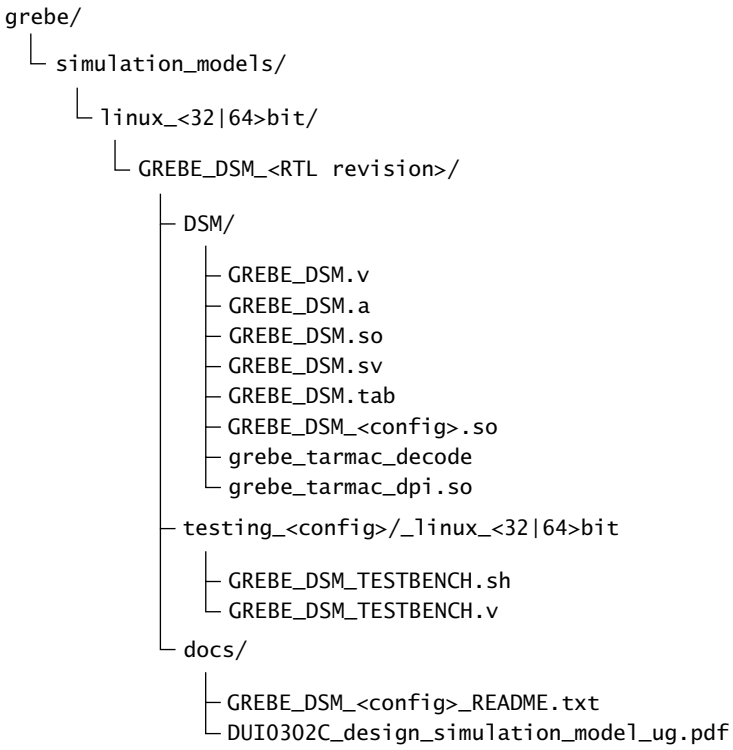


Figure 10-2 Deliverable directory structure

Chapter 11
Sign-off

In addition to your normal ASIC flow sign-off checks, you must satisfy certain verification criteria before you sign off your design. This chapter describes the sign-off criteria. It contains the following sections:

11.1 About sign-off

Figure 11-1 shows the top-level inputs, resources, outputs, and controls and constraints for sign-off.

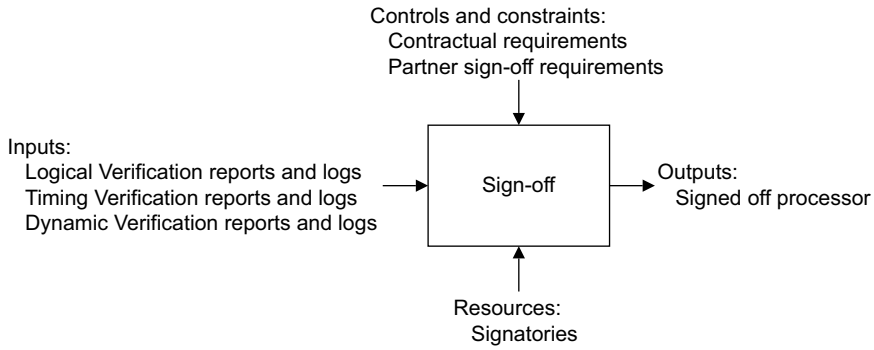


Figure 11-1 Sign-off process

11.2 Obligations for sign-off

Signatories must approve the sign-off of the design in accordance with:

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

See *Implementation controls, constraints, and obligations* on page 1-7 for more information.

11.3 Requirements for sign-off

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

- *Mandatory for sign-off*.
- *Recommended for sign-off*.

11.3.1 Mandatory for sign-off

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

In most cases, you must complete Logical Equivalence Check for a successful sign-off. See the Reference Methodology documents supplied by ARM.

Reports and logs from each of these stages are required for sign-off.

A minimum set of deliverable outputs is required at the end of the implementation. See *Completion of sign-off* on page 11-6.

Note

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

11.3.2 Recommended for sign-off

Table 11-1 shows the recommended stages for sign-off.

Table 11-1 Recommended stages for sign-off

Sign-off stage	Notes
Design Rule Check (DRC)	-
Layout Versus Schematic (LVS)	-
Timing Verification	See the Reference Methodology documents supplied by ARM.
Characterization	
Vector replay with back annotated netlist	If you are using a pre-hardened processor at the `GREBE` or `GREBEINTEGRATION` levels.

11.4 Steps for sign-off

To sign off the processor, you must meet the criteria in each of the following stages in the design flow:

1. *RTL integration*.
2. *Post-synthesis and place-and-route*.
3. *Post place-and-route timing*.

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.

11.4.1 RTL integration

You must verify the RTL deliverables before you begin the synthesis stage by running the supplied integration kit on the configured RTL. This confirms that the RTL has been successfully installed and configured.

Note

- You must use the files provided in `logical/models/cells/generic` for RTL integration.
- Do not use the implementation versions of files for RTL integration. See *Special purpose cells* on page 5-4 for additional information.

11.4.2 Post-synthesis and place-and-route

You must verify the functionality of the final placed-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 supplied by ARM.

11.4.3 Post place-and-route timing

You must verify the timing of the post place-and-route netlist before you sign off the netlist using *Static Timing Analysis* (STA). ARM also recommends that you run:

- All functional vectors appropriate to your configured build.
- Some or all the supplied validation tests on a netlist with back-annotated timing as a final check.

11.5 Completion of sign-off

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

These include:

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

Appendix A
GPIO Programmers Model

This appendix describes the GPIO programmers model. It contains the following section:

- *GPIO programmers model* on page A-2.

A.1 GPIO programmers model

This section describes the GPIO programmers model. It contains the following sections:

- *Data Register - GPIODATA*.
- *Direction Register - GPIODIR* on page A-4.
- *Interrupt Enable Register - GPIOIE* on page A-4.

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 based on input value change.
- The programmable bus interface can either be AHB or the *I/O Port* (IOP).

Table A-1 lists the GPIO registers.

Table A-1 GPIO registers

Address offset	Name	Type	Description
0x00000000-0x000003FF	**GPIODATA**	R/W	Reads the value of the **GPIOIN** pins, or sets the value driven onto **GPIOUT** pins.
0x00000400	**GPIODIR**	R/W	Configures the direction of the I/O. The value is driven on **GPIOEN.** Setting a bit defines that bit as an output.
0x00000410	**GPIOIE**	R/W	Configures the interrupt on input change feature.

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 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 **HADDR[9:2]**, or **IOADDR[9:2]** for the IOP GPIO, 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.

The following examples assume an AHB-Lite GPIO but they equally apply to an IOP GPIO.

Bit set example

To set bit[9] of GPIO 0 Data Register, perform a byte write access to address 0x40000009 with **HWDATA** set to 0x200 and **HSIZE** set to 3'b000.

The mask 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 **HWDATA** set to `0x0` and **HSIZE** set to `0b000`.

The mask 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 **HSIZE** set to `3'b00`.

The mask extracted from the address is `0x02` and applied to the second byte of the GPIO 0 Data register.

HRDATA contains the value of bit[9], all other bits are zeroes.

Figure A-1 shows the structure of the GPIO Data Register.

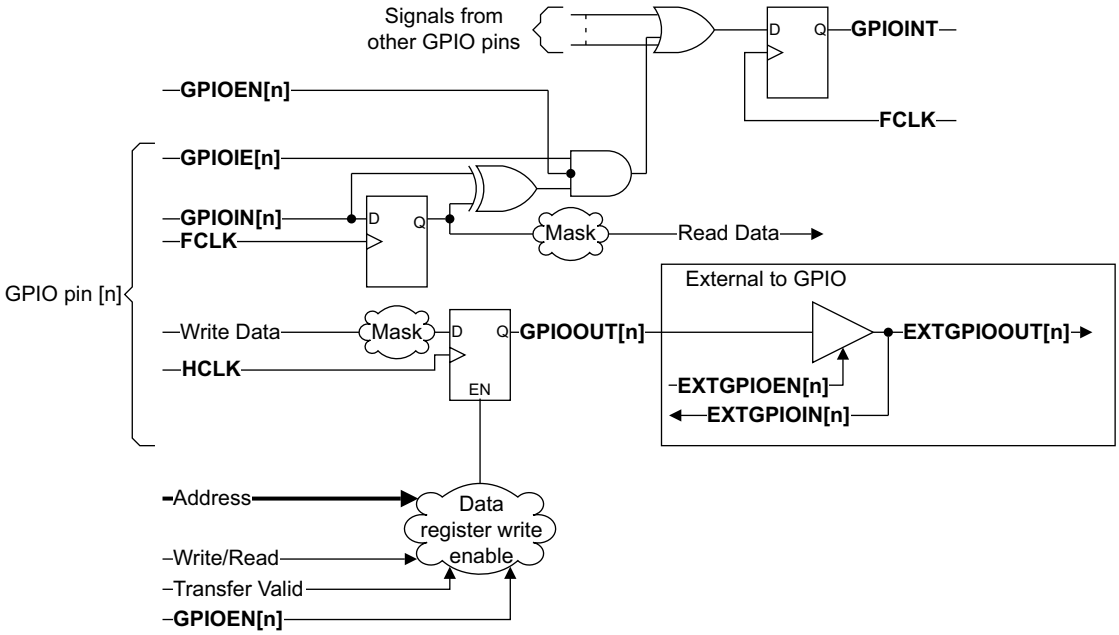


Figure A-1 GPIO Data Register

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 `1'b1` 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
GRBIKMCU GPIO Integration

This appendix describes the integration of the GPIO within the GRBIKMCU example system. It contains the following section:

- *GPIO integration* on page B-2.

B.1 GPIO integration

This section describes the GPIO bit assignments used in the execution testbench. It contains:

- *GPIO 0 bit assignments* on page B-4.
- *GPIO 1 bit assignments* on page B-5.
- *GPIO 2 bit assignments* on page B-6.

The Cortex-M23 processor execution testbench instantiates three GPIOs internally, that is, GPIO 0, 1, and 2. For more information about GPIO, see Appendix A *GPIO Programmers Model*.

Figure B-1 shows the connection of the GPIO interrupt lines.

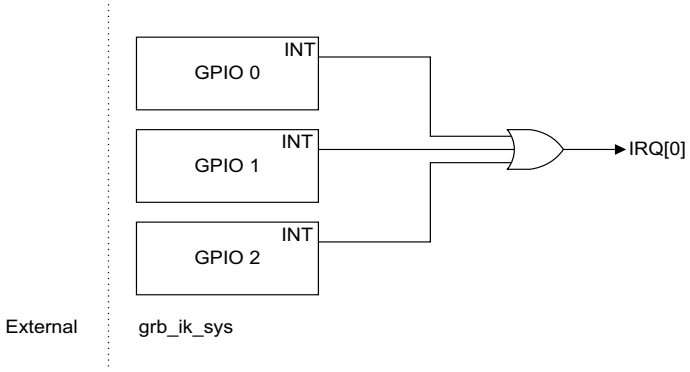


Figure B-1 GPIO interrupt line connections

Figure B-2 on page B-3 shows the GPIO 0 connections.

Note

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

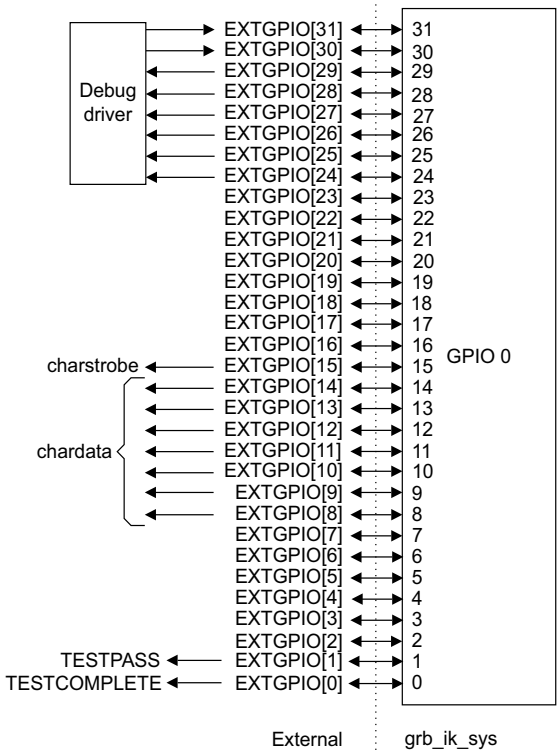


Figure B-2 GPIO 0 connections

Figure B-3 on page B-4 shows the GPIO 1 connections.

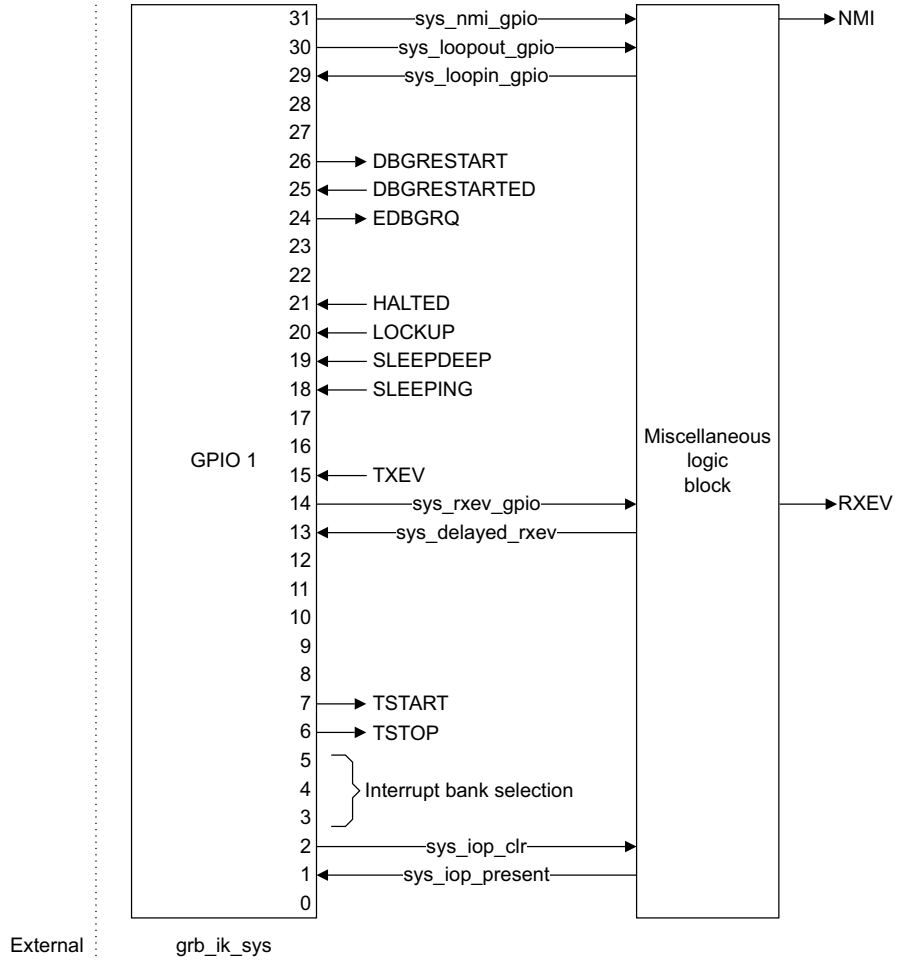


Figure B-3 GPIO 1 connections

Figure B-4 shows the GPIO 2 connections.

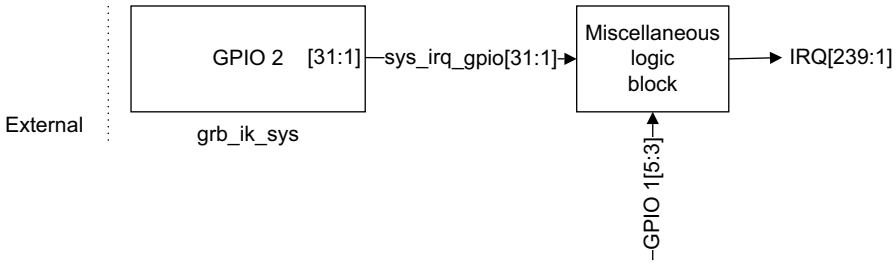


Figure B-4 GPIO 2 connections

B.1.1 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 GRBIKMCU, with its signals routed to the top level of the GRBIKMCU.

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.

Table B-1 shows the GPIO 0 bit assignments in the Cortex-M23 processor execution testbench.

Table B-1 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.

B.1.2 GPIO 1 bit assignments

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

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

Table B-2 shows the bit assignments.

Table B-2 GPIO 1 bit assignments

Bit	Receive input from	Send output to	Connection information
[31]	-	**NMI**	Drives the **NMI** pin of the Cortex-M23 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-M23 processor.
[25]	**DBGRESTARTED**	-	Connects to the **DBGRESTARTED** pin of the Cortex-M23 processor.
[24]	-	**EDBGRQ**	Drives the **EDBGRQ** pin of the Cortex-M23 processor.
[23:22]	-	-	Not used.
[21]	**HALTED**	-	Connects to the **HALTED** pin of the Cortex-M23 processor.
[20]	**LOCKUP**	-	Connects to the **LOCKUP** pin of the Cortex-M23 processor.

Table B-2 GPIO 1 bit assignments (continued)

Bit	Receive input from	Send output to	Connection information
[19]	**SLEEPDEEP**	-	Connects to the **SLEEPDEEP** pin of the Cortex-M23 processor.
[18]	**SLEEPING**	-	Connects to the **SLEEPING** pin of the Cortex-M23 processor.
[17:16]	-	-	Not used.
[15]	**TXEV**	-	Connects to the **TXEV** pin of the Cortex-M23 processor.
[14]	-	**RXEV**	Drives the **RXEV** pin of the Cortex-M23 processor.
[13]	Miscellaneous logic block	-	Delayed version of the **RXEV** pin of the Cortex-M23 processor.
[12:8]	-	-	Not used.
[7]	-	**TSTART**[a]	Connects to **TSTART** pin of CoreSight MTB-M23.
[6]	-	**TSTOP**[a]	Connects to **TSTOP** pin of CoreSight MTB-M23.
[5:3]	-	Interrupt driving module	Banking of Cortex-M23 processor interrupts: **000** GPIO 2 drives interrupts [31:1] **001** GPIO 2 drives interrupts [63:32] **010** GPIO 2 drives interrupts [95:64] **011** GPIO 2 drives interrupts [127:96] **100** GPIO 2 drives interrupts [159:128] **101** GPIO 2 drives interrupts [191:160] **110** GPIO 2 drives interrupts [223:192] **111** GPIO 2 drives interrupts [239:224]
[2]	-	Miscellaneous logic block	Connects to I/O Port sticky bit clear. Asserting this pin clears the sticky bit.
[1]	Miscellaneous logic block	-	Connects to sticky bit indicating presence of I/O Port on the Cortex-M23 processor.
[0]	-	-	Not used.

a. The TSTART and TSTOP inputs are also connected to the CTI, and CTI can drive them, see Appendix G *CTI*.

B.1.3 GPIO 2 bit assignments

GPIO 2 is used internally to the example GRBIKMCU to drive **IRQ[239:1]** for testing purposes only.

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

Table B-3 shows the GPIO 2 bit assignments.

Table B-3 GPIO 2 bit assignments

Bit	Receive input from	Send output to	Connection information
[31:0]	-	Miscellaneous logic block	Drives the corresponding **IRQ** pin of the Cortex-M23 processor, after a delay, according to the coding of bits[5:3] in Table B-2 on page B-5. **IRQ** is not driven according to this coding scheme.

Appendix C
CoreSight SoC

This appendix describes the location of the CoreSight SoC-400 configuration files for the Cotrtex-M23 processor. It contains the following section:

- *Location of the Cortex-M23 processor CoreSight SoC-400 support files* on page C-2.

C.1 Location of the Cortex-M23 processor CoreSight SoC-400 support files

The Cortex-M23 processor deliverables provide a header file for use with the CoreSight SoC-400 product, as described in the following table.

Table C-1 Location of CoreSight SoC-400 header file

Filename	Description	Location
`cssoc_grebe.h`	Header file containing the description of the Cortex-M23 processor CoreSight components and associated function calls to support the integration of a Cortex-M23 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 the reference file is listed in the following table.

Table C-2 Location of ETM trace comparison tool

Filename	Description	Location
`cssoc_grebe_trace_i.ref`	A CoreSight SoC-400 Cortex-M23 trace reference file for the ETM. The configuration parameter ETM is set to 1.	`logcial/testbench/integration_cssoc`

Appendix D
SysTick Examples

This appendix provides some examples of setting up SysTick timing. It contains the following section:

D.1 SysTick examples

Example D-1 shows 25MHz **SCLK** with no reference provided.

Example D-1 25MHz SCLK, no reference provided

```
CFGSTCALIB[25]   = 1'b1; // no reference provided
CFGSTCALIB[24]   = 1'b0; // no skew
CFGSTCALIB[23:0] = (25MHz x 10mS) - 1
                 = 249,999 decimal
                 = 24'h03D08F
```

Example D-2 shows 14.31818MHz **SCLK** with no reference provided.

Example D-2 14.31818MHz SCLK, no reference provided

```
CFGSTCALIB[25]   = 1'b1; // no reference provided
CFGSTCALIB[24]   = 1'b1; // skew
CFGSTCALIB[23:0] = (14.31818MHz x 10mS) - 1
                 = 143,180.8 decimal
                 = 143,181 with skew
                 = 24'h022F4D
```

Example D-3 shows 1MHz **SCLK** with **STCLKEN** enabled once every four cycles.

Example D-3 1MHz SCLK, STCLKEN enabled once every four cycles

```
CFGSTCALIB[25]   = 1'b0; // reference provided
CFGSTCALIB[24]   = 1'b0; // no skew
CFGSTCALIB[23:0] = (1MHz x 10mS / 4) - 1
                 = 2,499 decimal
                 = 24'h0009C3
```

Example D-4 shows an 32.768kHz asynchronous example.

Example D-4 32.768kHz asynchronous example

```
CFGSTCALIB[25]   = 1'b0; // reference provided
CFGSTCALIB[24]   = 1'b1; // skew
CFGSTCALIB[23:0] = (32768 x 10mS) - 1
                 = 326.68 decimal
                 = 327 with Skew
                 = 24'h000147
```

Appendix E
Debug Driver

This appendix describes the debug driver that is supplied with the execution testbench. It contains the following section:

E.1 Debug driver

The debug driver block is a testbench component that uses a GPIO and software to control the DAP by driving the SW or JTAG pins of GRBIKMCU. The debug driver contains an instantiation of the GREBEINTEGRATION_MCU level, memories, and a GPIO. Tests running on GRBIKMCU can use pins **EXTGPIO[31:24]** to request the debug driver to initiate one of several predetermined operations which are enumerated in header file debugdriver_functions.h. This arrangement enables the execution testbench to test GRBIKMCU even when its processor is sleeping.

The debug driver always executes the binary file debugdriver.bin, regardless of the test that is run on GRBIKMCU. The debugdriver.bin binary must be built from the supplied source code. See *Test program compilation using the ARM or GCC Compiler* on page 8-13 for information on how to compile and build the execution testbench tests.

Table E-1 lists the software source files used by the debug driver located in logical/testbench/execution_tb/.

Table E-1 Debug driver source files

File	Description
tests/Device/ARM/grbikmcu/Source/ARM/boot_grbikmcu.c	This file provides the stack and heap initialization, vector table and default exception handlers. boot_grbikmcu.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_grbikmcu.c.
tests/Device/ARM/grbikmcu/Source/ARM/armclang_grbikmcu.scat	Linker script to configure Secure memory regions for the ARM compiler.
tests/Device/ARM/grbikmcu/Source/GCC/gcc_arm_grbikmcu_ns.ld	Linker script to configure Non-secure memory regions.
tests/Device/ARM/grbikmcu/Source/GCC/gcc_arm_grbikmcu.ld	Linker script to configure Secure memory regions.
tests/Device/ARM/grbikmcu/Source/system_ARMCM23.c	C file that provides device-specific configuration for he Cortex-M23 processor microcontroller device.
tests/Device/ARM/grbikmcu/Include/system_ARMCM23.h	Header file that provides device-specific configuration for the Cortex-M23 processor microcontroller device.
tests/Device/ARM/grbikmcu/Include/template/partition_ARMCM23.h	Contains an example of CMSIS-Processor initial setup for Secure/Non-secure zones for the Cortex-M23 processor.
tests/Device/ARM/grbikmcu/Source/GCC/startup_grbikmcu.c	Provides the stack and heap initialization, vector table and default exception handlers. This file is provided as an example of a boot file written entirely in C for the GCC compiler.
tests/Device/ARM/grbikmcu/Include/ARMCM23_TZ.h	Device-specific file that defines the peripherals for the Cortex-M23 processor with Security Extension example device.
tests/Device/ARM/grbikmcu/Include/ARMCM23.h	Device-specific file that defines the peripherals for the Cortex-M23 processor example device.
tests/Device/ARM/grbikmcu/Include/grbikmcu.h	Device-specific file that defines the peripherals for the GRBIKMCU example microcontroller device.
tests/Device/ARM/grbikmcu/Source/GCC/sys/custom_file.h	Retargets the file structure.

Table E-1 Debug driver source files (continued)

File	Description
tests/CMSIS/Include/cmsis_gcc.h	Defines the Cortex-M core register and instruction access functions for GNU compiler.
tests/CMSIS/Include/cmsis_armclang.h	Defines the Cortex-M core register and instruction access functions for ARM compiler 6.
tests/CMSIS/Include/tz_context.h	Defines the Security Extension function.
tests/CMSIS/Include/arm_compiler.h	Defines the macros and the include file in function of the compiler and its version.
tests/CMSIS/Include/cmsis_armcc.h	Defines the Cortex-M core register and instruction access functions for ARM compiler version 4/5.
tests/CMSIS/Include/core_cm23.h	Defines the structures for register access of the Cortex-M23 processor. It also contains certain checker for the configuration of processor.
tests/Device/ARM/grbikmcu/Source/ARM/armclang_grbikmcu_ns.scat	Linker script to configure Non-secure memory regions for the ARM compiler.
tests/IKtests.h	This header file describes and defines the allocation of **GPIO** pins used by the GRBIKMCU in the execution testbench. It also declares the function prototypes the execution testbench tests use to communicate with the debug driver.
tests/IKtests.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.
tests/IKConfig.h	This file includes some defines that you must edit to match the implemented configuration of the execution testbench.
tests/Makefile	This file enables you to build the execution testbench tests using the ARM Compiler or GCC toolchain.
tests/retarget_grbikmcu.c	This file implements the functions necessary to retarget the C-library printf() function output to the GRBIKMCU **GPIO** pins.
tests/debugdriver.h	Contains various defines used by the debug driver block, including the GPIO pin allocations.
tests/debugdriver.c	This is the main source code file for the debug driver block. It includes the routines for communicating with GRBIKMCU through the GPIO and the routines to drive the GRBIKMCU Serial Wire or JTAG interface.
tests/debugdriver_functions.h	Contains a C enum representing the functions that the debug driver makes available to GRBIKMCU.

Figure E-1 on page E-4 shows the debug driver.

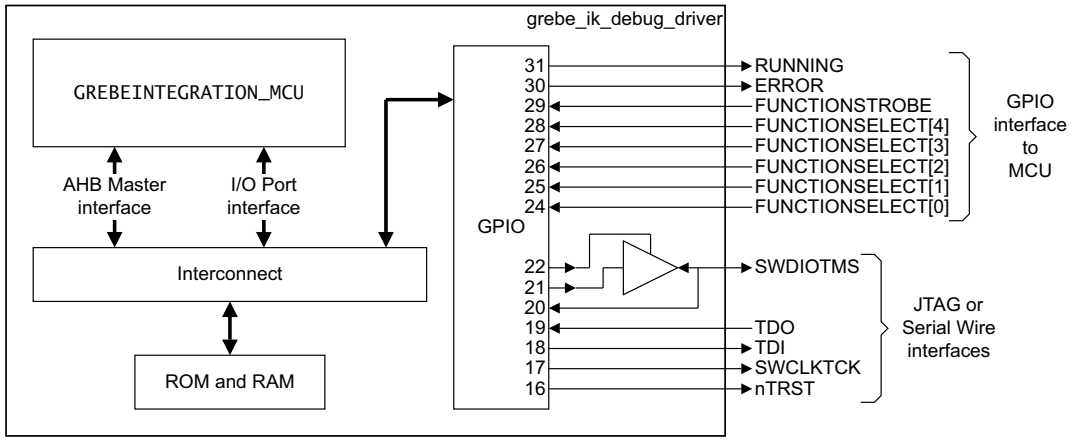


Figure E-1 Debug driver

Note

- During netlist simulations, the netlist replaces the RTL in both GRBIKMCU and the debug driver.
- 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. For example, if you change the SRPG pins to match your requirements, ensure that these pins are tied inactive in the debug driver module.

Appendix F
Power Intent

This appendix describes the optional State Retention and Power Gating features of the Cortex-M23 processor. It contains the following section:

- *Cortex-M23 processor power intent* on page F-2.

F.1 Cortex-M23 processor power intent

This section describes how you divide the logic into power domains, to enable you to power down as much of the processor as possible, in each low-power mode. It also specifies:

- Which power domain interfaces require isolation cells.
- The signals required to control the isolation cells and power switches.
- The state retention requirements.

It contains the following sections:

F.1.1 Power intent specification

The *Unified Power Format* (UPF) file specifies the power intent for the Cortex-M23 processor. The UPF file is located in the `logical/power_intent/upf` directory. You should use the file format required by your EDA tools. Synthesis tools use the power intent specification to insert cells automatically into the design, from your technology library.

The types of cells that are inserted are:

- Isolation cells.
- State retention flip-flops.
- Power switches

Also, you can use the UPF files, with the RTL, to validate the power intent specification of a SoC:

- Statically, using rule checking tools.
- Dynamically, using power-aware simulation tools.

Power-aware simulation tools insert technology-independent models of the SRPG cells into their databases, to model the behavior of the power intent specification. The synthesis scripts, provided by ARM, support SRPG implementation at the `GREBEINTEGRATION_MCU` level. The execution testbench supports power-aware simulation. See Chapter 8 *Execution Testbench*.

This appendix describes the power intent at the `GREBEINTEGRATION_MCU` level, `logical/power_intent/upf/GREBEINTEGRATION_MCU.upf`, because the synthesis scripts provided by ARM support this level.

The information in this section is only an example of how you might implement the power gating features. You can modify the power intent files provided, to meet the requirements of your cell library, implementation of your processor, and SoC. ARM recommends that you perform any modification on a copy of a file kept, for example, in `logical/power_intent/upf/power_intent_<tech>`.

F.1.2 Power domains

The processor contains three power domains: TOP, PD_SYS, and PD_DBG. Figure F-1 on page F-4 shows the domains and the location of isolation cells and switch cells that can be inserted in the design by your EDA tools.

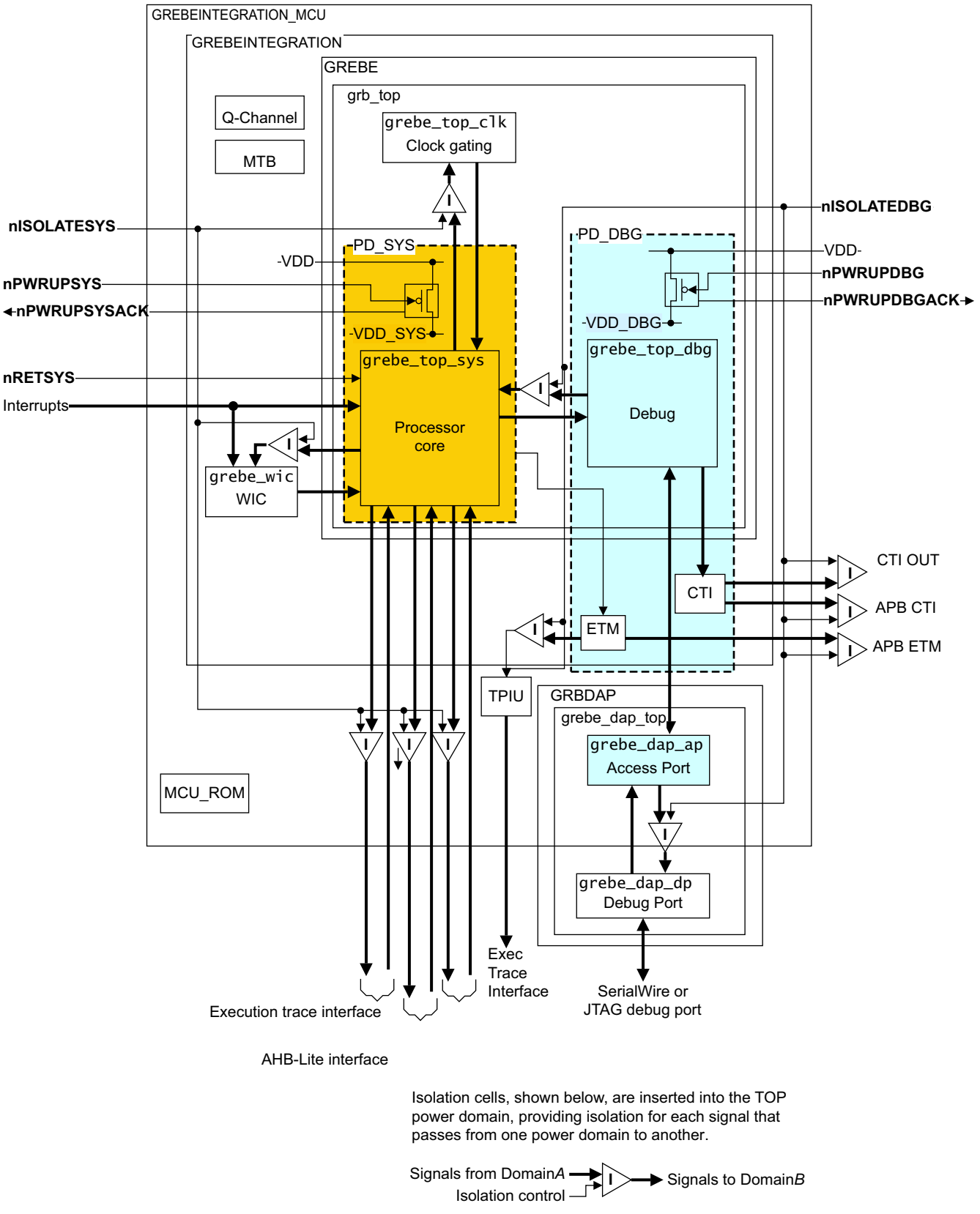


Figure F-1 Cortex-M23 processor power domains

In Figure F-1 on page F-4 blocks that are not colored are in the TOP power domain. The colored blocks show the PD_SYS and PD_DBG power domains. Also, the switch cells are shown for:

- VDD_SYS, that powers the PD_SYS power domain.
- VDD_DBG, that powers the PD_DBG power domain.

Figure F-1 on page F-4 does not show the connection of:

- Every block to ground, VSS.
- Each block in the TOP power domain, including the isolators, to VDD.
- Each block in the PD_SYS power domain to VDD_SYS.
- Each block in the PD_DBG power domain to VDD_DBG.

Table F-1 describes which blocks are supplied by each domain.

Table F-1 Power domains

Domain name	Supply signal name	Comments
TOP	VDD	The TOP domain is always on. It contains the DAP debug port, clock gates, WIC, Q-Channel, TPIU, MCU_ROM, APB_interconnect, and MTB.
PD_SYS	VDD_SYS	The PD_SYS domain is the largest domain and contains the majority of the processor core logic. This domain must not be powered down if the PD_DBG domain is powered up.
PD_DBG	VDD_DBG	The PD_DBG domain covers the debug logic in the processor core, the access port logic in the DAP, CTI, ETM, and DBG_misc. This domain is normally powered up when a debugger is connected.

All power domains connect to the VSS ground signal.

Note

You must place the clock gating logic in the TOP domain, not in the PD_SYS domain, because PD_SYS is configured to use state retention, but some technology libraries do not have state retention support in their clock gating cells.

F.1.3 Low-power modes

The Cortex-M23 processor power intent supports three low-power modes Run, Sleep and Debug. See the Low-power modes section in the *ARM® Cortex®-M23 Processor Technical Reference Manual* for more information. Table F-2 shows the state of the power supply voltages for each domain, in all three modes. Power on reset must only occur in Run or Debug mode, otherwise any transition between modes is permitted.

Table F-2 Low-power modes

Low-power mode	VDD	VDD_SYS	VDD_DBG	Description
Run	on	on	off	Run mode, normal operation.
Sleep	on	off	off	Sleep mode, main processor logic powered down.
Debug	on	on	on	Debug mode.

F.1.4 Power control signals

The example power intent files use SRPG control signals included in the corresponding level. You can modify the signal names and sense to suit the requirements of your SRPG implementation and technology library.

You must connect a SoC specific power management controller to either these ports and the following signals according to the chosen power management policy:

Legacy interface

WAKEUP, **WICSENSE**, **WICENREQ**, **WICENACK**, **SLEEPHOLDREQN**, **SLEEPHOLDACKN**, **SLEEPING**, **SLEEPDEEP**, **CDBGPWRUPREQ**, and **CDBGPWRUPACK**.

Q-Channel interface

WAKEUP, **WICENREQ**, **WICENACK**, **WICSENSE**, **SLEEPING**, **SLEEPDEEP**, **SYSQREQN**, **SYSQACCEPTN**, **SYSQDENY**, **SYSQACTIVE**, **DBGQREQN**, **DBGQACCEPTN**, **DBGQDENY**, and **DBGQACTIVE**.

Note

SYSQACTIVE uses **WAKEUP** and **SLEEPING** signals to trigger the **SYSQACTIVE** hint.

These interfaces are mutually exclusive and cannot be used at the same time. See *Power Management Unit - Q-Channel Interface* on page 4-36.

Table F-3 describes the power control signals and Figure F-1 on page F-4 shows how to use them in the SRPG implementation.

Table F-3 Power Control Signals

Signal	Direction	Active sense	Description
nPWRUPSYS	Input	HIGH	PD_SYS domain power down control.
nPWRUPSYSACK	Output	HIGH	PD_SYS domain power down acknowledge.
nISOLATESYS	Input	LOW	PD_SYS domain isolate control.
nRETSYS	Input	LOW	PD_SYS domain retention control.
nPWRUPDBG	Input	HIGH	PD_DBG domain power down control.
nPWRUPDBGACK	Output	HIGH	PD_DBG domain power down acknowledge.
nISOLATEDBG	Input	LOW	PD_DBG domain isolate control.

F.1.5 Signal isolation

The power intent uses a policy of placing isolation cells on output ports of domains that can be powered down. Figure F-1 on page F-4 shows the location of the isolation cells. Where isolation cells are inferred, Figure F-1 on page F-4 shows only one isolation cell on each interface. However, an isolation cell is automatically inserted into every signal in the direction shown in the figure. By default, the isolation cells clamp the output signals LOW, but some signals must be clamped HIGH, as shown in Table F-4.

Table F-4 Clamp HIGH output signals

Domain	Module name	Output signal
PD_SYS	`grb_top_sys`	**sleeping_o**
		sleep_deep_o

Table F-4 Clamp HIGH output signals (continued)

Domain	Module name	Output signal
PD_DBG	grb_top_dbg	**dbg_restarted_o**
	DAP	**slvprot_o[0]**
	DBG_misc	**preadyc_o**
		preadyt_o
		afready_tpiu_o
		pslverrt_o (if ETM is present)
		pslverrc_o (if CTI is present)

F.1.6 State retention

To support the Sleep low-power mode, your implementation is required to retain only the state of the registers in the processor core, in the PD_SYS domain. ARM recommends that you implement state retention by replacing the registers with state retention registers. Your implementation is not required to retain the state of the debug registers in the PD_DBG domain because they are programmed by an external Debugger after the PD_DBG domain is powered up.

Table F-5 shows the state retention information.

Table F-5 State retention information

Domain	Mode when state is retained	Retention supply signal name	Save signal name	Save sense	Restore signal name	Restore sense
PD_SYS	Sleep	**VDD**	**SYSRETAINn nRETSYS**	HIGH	**nRETSYS**	LOW

F.1.7 IEEE 1801

IEEE 1801 is the industry standard power intent specification format and the current version is IEEE 1801-2009. The UPF files provided comply with the IEEE 1801-2009 standard.

Appendix G
CTI

This appendix describes the Cortex-M23 processor *Cross Trigger Interface* (CTI). It contains the following section:

- *CTI* on page G-2.

G.1 CTI

The CTI is an implementation of the CoreSight Cross Trigger Interface architecture version CTI v2 which has been optimized for use with the Cortex-M23 processor and the optional CoreSight MTB-M23 Micro Trace Buffer or the optional CoreSight ETM-M23 Embedded Trace Macrocell as follows:

- Fully synchronous clocking only (synchronous to the processor clock, **HCLK**).
- Four channel inputs and output.
- One trigger input, six trigger outputs.
- Direct signal interfacing to the processor and MTB.
- APB programming interface.

The CTI is supplied in `logical/grb_cti/verilog/GRBCTI.v`.

If you require the CTI channel interface to be clocked asynchronously to the processor, you may add an appropriate bridge, or replace the CTI with the CoreSight CTI plus associated interfacing logic and APB bridge if required. Contact ARM if you need further information.

Figure G-1 shows the CTI top level.

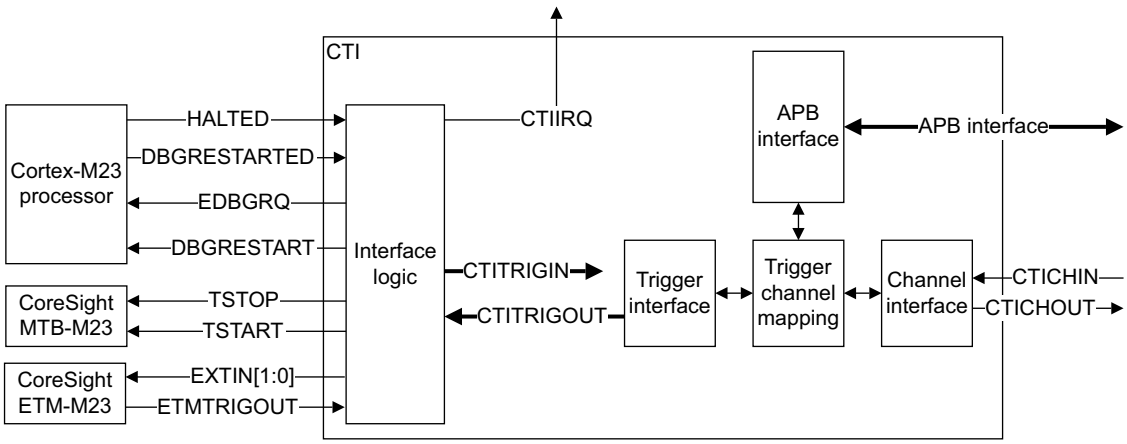


Figure G-1 CTI

G.1.1 Connecting the processor to the CTI

The CTI is an implementation of the CoreSight CTI architecture that is optimized for use with the Cortex-M23 processor and the CoreSight MTB-M23 or CoreSight ETM-M23.

Table G-1 shows the CTI input triggers when using MTB.

Table G-1 CTI input triggers when using MTB

Signal	Function	Connection	RTL signal name
CTITRIGIN[0]	Cross-Halt	From core	`core_halted_i`
CTITRIGIN[1]	**CMPMATCH0**	From DWT	`dwt_cmpmatch_i[0]`
CTITRIGIN[2]	**CMPMATCH1**	From DWT	`dwt_cmpmatch_i[1]`
CTITRIGIN[3]	**CMPMATCH2**	From DWT	`dwt_cmpmatch_i[2]`

Table G-1 CTI input triggers when using MTB (continued)

Signal	Function	Connection	RTL signal name
CTITRIGIN[4]	**CMPMATCH3**	From DWT	dwt_cmpmatch_i[3]
CTITRIGIN[5]	-	-	-
CTITRIGIN[6]	-	-	-
CTITRIGIN[7]	-	-	-

Table G-2 shows the CTI input triggers when using ETM.

Table G-2 CTI input triggers when using ETM

Signal	Function	Connection	RTL signal name
CTITRIGIN[0]	Cross-Halt	From core	core_halted_i
CTITRIGIN[1]	**CMPMATCH0**	From DWT	dwt_cmpmatch_i[0]
CTITRIGIN[2]	**CMPMATCH1**	From DWT	dwt_cmpmatch_i[1]
CTITRIGIN[3]	**CMPMATCH2**	From DWT	dwt_cmpmatch_i[2]
CTITRIGIN[4]	**ETMTRIGOUT[0]**	From ETM	etm_cti_extout_i[0]
CTITRIGIN[5]	0	-	-
CTITRIGIN[6]	-	-	-
CTITRIGIN[7]	-	-	-

Table G-3 shows the CTI input triggers when using neither MTB nor ETM.

Table G-3 CTI input triggers when using neither MTB nor ETM

Signal	Function	Connection	RTL signal name
CTITRIGIN[0]	Cross-Halt	From core	core_halted_i
CTITRIGIN[1]	-	-	-
CTITRIGIN[2]	-	-	-
CTITRIGIN[3]	-	-	-
CTITRIGIN[4]	-	-	-
CTITRIGIN[5]	-	-	-
CTITRIGIN[6]	-	-	-
CTITRIGIN[7]	-	-	-

Table G-4 shows the CTI output triggers when using MTB.

Table G-4 CTI output triggers when using MTB

Signal	Function	Connection	RTL signal name
CTITRIGOUT[0]	**EDBGRQ**	To core	cti_edbgrq_o
CTITRIGOUT[1]	**DBGRESTART**	To core	cti_dbg_restart_o[a]
CTITRIGOUT[2]	**CTIIRQ[0]**	To top-level[b]	cti_ext_irq_o[0]
CTITRIGOUT[3]	**CTIIRQ[1]**	To top level[b]	cti_ext_irq_o[1]
CTITRIGOUT[4]	**TSTART** event	To MTB	cti_mtb_tstart_o
CTITRIGOUT[5]	**TSTOP** event	To MTB	cti_mtb_tstop_o
CTITRIGOUT6]	-	-	-
CTITRIGOUT[7]	-	-	-

a. The CTI also has the nvic_dbg_restarted_i signal from core in addition to cti_dbg_restart_o. This completes the dbg_restart/dbg_restarted handshake

b. **CTIIRQ** crosses GREBEINTEGRATION and GREBEINTEGRATION_MCU.

Table G-5 shows the CTI output triggers when using ETM.

Table G-5 CTI output triggers when using ETM

Signal	Function	Connection	RTL signal name
CTITRIGOUT[0]	**EDBGRQ**	To core	cti_edbgrq_o
CTITRIGOUT[1]	**DBGRESTART**	To core	cti_dbg_restart_o[a]
CTITRIGOUT[2]	**CTIIRQ[0]**	To top-level[b]	cti_ext_irq_o[0]
CTITRIGOUT[3]	**CTIIRQ[1]**	To top level[b]	cti_ext_irq_o[1]
CTITRIGOUT[4]	**EXTIN[0]**	To ETM	cti_etm_extin_o[0]
CTITRIGOUT[5]	**EXTIN[1]**	To ETM	cti_etm_extin_o[1]
CTITRIGOUT6]	-	-	-
CTITRIGOUT[7]	-	-	-

a. The CTI also has the nvic_dbg_restarted_i signal from core in addition to cti_dbg_restart_o. This completes the dbg_restart/dbg_restarted handshake

b. **CTIIRQ** crosses GREBEINTEGRATION and GREBEINTEGRATION_MCU.

Table G-6 shows the CTI output triggers when using neither MTB nor ETM.

Table G-6 CTI output triggers when using neither MTB nor ETM

Signal	Function	Connection	RTL signal name
CTITRIGOUT[0]	**EDBGRQ**	To core	cti_edbgrq_o
CTITRIGOUT[1]	**DBGRESTART**	To core	cti_dbg_restart_o[a]
CTITRIGOUT[2]	**CTIIRQ[0:1]**	To top-level[b]	cti_ext_irq_o[0]

Table G-6 CTI output triggers when using neither MTB nor ETM (continued)

Signal	Function	Connection	RTL signal name
CTITRIGOUT[3]	**CTIIRQ[0:1]**	To top level[b]	cti_ext_irq_o[1]
CTITRIGOUT[4]	-	-	-
CTITRIGOUT[5]	-	-	-
CTITRIGOUT6]	-	-	-
CTITRIGOUT[7]	-	-	-

a. The CTI also has the nvic_dbg_restarted_i signal from core in addition to cti_dbg_restart_o. This completes the dbg_restart/dbg_restarted handshake

b. **CTIIRQ** crosses GREBEINTEGRATION and GREBEINTEGRATION_MCU.

Note

- After the processor is halted using **CTITRIGOUT[0]**, the **EDBGRQ** signal remains HIGH. The debugger must write to CTIINTACK to clear the halting request before restarting the processor.
- After asserting the CTI interrupt signal, the *Interrupt Service Routine* (ISR) must clear the **CITIRQ** interrupt request by writing to the CTI Interrupt Acknowledge, CTIINTACK.

Appendix H
Signal Timing Constraints

This appendix describes the timing constraints on the processor signals. It contains the following section:

H.1 Signal timing constraints

The timing constraints for signals are classified according to the percentage of the clock period that is available for external logic:

- 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.

Table H-1 shows the AHB master interface timing constraints.

Note

The constraints are at `GREBEINTEGRATION_MCU` level

Table H-1 AHB master interface

Signal	Timing constraint	Direction
nHRESET	50	Input
HTRANS[1:0]	50	Output
HSIZE[2:0]	60	Output
HWRITE	60	Output
HPROT[6:0]	50	Output
HNONSEC	40	Output
HBURST[2:0]	60	Output
HADDR[31:0]	50	Output
HWDATA[31:0]	60	Output
HRDATA[31:0]	50	Input
HREADY	60	Input
HRESP	60	Input
HEXCL	40	Output
HEXOKAY	60	Input
HMASTER	60	Output

Table H-2 shows the AHB interface extensions timing constraints.

Table H-2 AHB interface extensions

Signal	Timing constraint	Direction
SPECHTRANS	60	Output
CODENSEQ	40	Output
CODEHINT [3:0]	20	Output
DATAHINT[1:0]	60	Output
CURRNS	60	Output

Table H-3 shows the TPIU interface extensions timing constraints.

Table H-3 TPIU interface extensions

Signal	Timing constraint	Direction
TRACECLK	50	Output
TRACEDATA[3:0]	40	Output
TRACESWO	40	Output
SWOACTIVE	40	Output
TPIUACTV	40	Output
TPIUBAUD	40	Output
nTRESET	50	Input

Table H-4 shows the I/O port interface output timing constraints.

Table H-4 I/O port output timing constraints

Signal	Timing constraint	Direction
IOTRANS	50	Output
IOWRITE	60	Output
IOADDR[31:0]	60	Output
IOSIZE[1:0]	60	Output
IOPRIV	60	Output
IOMASTER	60	Output
IOWDATA[31:0]	60	Output
IONONSEC	60	Output

Table H-5 shows the combinatorial paths from processor outputs to inputs.

Table H-5 I/O port combinatorial paths

Start	End	Timing constraint
IOCHECK[31:0]	**IOMATCH**	10
IOADDR[31:0]	**IORDATA[31:0]**	20
IOTRANS	**IORDATA[31:0]**	10
IOSIZE[1:0]	**IORDATA[31:0]**	20
IOPRIV	**IORDATA[31:0]**	20
IOMASTER	**IORDATA[31:0]**	20
IONONSEC	**IORDATA[31:0]**	20

Table H-6 shows the WIC interface timing constraints.

Table H-6 WIC interface

Signal	Timing constraint	Direction
WICENACK	60	Output
WAKEUP	60	Output
WICSENSE[241:0]	60	Output
WICENREQ	60	Input

Table H-7 shows the Sleep control interface timing constraints.

Table H-7 Sleep-control interface

Signal	Timing constraint	Direction
SLEEPING	60	Output
SLEEPDEEP	60	Output
SLEEPHOLDACKn	60	Output
SLEEPHOLDREQn	60	Input

Table H-8 shows the Interrupts and events timing constraints.

Table H-8 Interrupts and events

Signal	Timing constraint	Direction
TXEV	60	Output
RXEV	60	Input
IRQ[239:0]	60	Input
NMI	60	Input

Table H-9 shows the status timing constraints.

Table H-9 Status

Signal	Timing constraint	Direction
LOCKUP	60	Output
HALTED	60	Output

Table H-10 shows the debug timing constraints.

Table H-10 Debug

Signal	Timing constraint	Direction
DBGRESTARTED	60	Output
DBGRESTART	60	Input
EDBGRQ	60	Input
NIDEN	10	Input
DBGEN	10	Input
SPIDEN	10	Input
SPNIDEN	10	Input
nDBGRESET	50	Input

Table H-11 shows the configuration timing constraints.

Table H-11 Configuration

Signal	Timing constraint	Direction
CFGSECEXT	60	Input
CFGSTCALIB[25:0]	60	Input
CFGSTCALIBNS[25:0]	60	Input
ECOREVNUM[19:0]	60	Input
INSTANCEID[3:0]	40	Input
INITVTOR	30	Input
INITVTORNS	30	Input

Table H-12 shows the miscellaneous signals timing constraints.

Table H-12 Miscellaneous signals

Signal	Timing constraint	Direction
nPORESET	50	Input
SYSRESETREQ	90	Output
STCLKEN	60	Input
STCLKENNS	60	Input
CPUWAIT	60	Input
IRQLATENCY[7:0]	60	Input
GATEHCLK	60	Output
DFTRSTDISABLE	10	Input
DFTCGEN	20	Input

Table H-13 shows the JTAG and Serial Wire interface timing constraints.

Table H-13 JTAG and Serial Wire interface

Signal	Timing constraint	Direction
nTRST	50	Input
DEVICEEN	50	Input
TDI	60	Input
TDO	60	Output
nTDOEN	60	Output
SWDITMS	60	Input
SWDO	60	Output
SWDOEN	60	Output
SWDETECT	60	Output

Table H-14 shows the PMU interface timing constraints.

Table H-14 PMU interface

Signal	Timing constraint	Direction
CDBGPWRUPREQ[a]	60	Output
CDBGPWRUPACK[b]	-	Input

a. **CDBGPWRUPREQ** is clocked by **SWCLKTCK**

b. **CDBGPWRUPACK** is asynchronous

Table H-15 shows the APB interface timing constraints.

Table H-15 APB interface

Signal	Timing constraint	Direction
PSEL	40	Input
PENABLE	40	Input
PADDR[11:2]	40	Input
PSTRB[3:0]	50	Input
PPROT[2:0]	50	Input
PREADY	50	Output
PWRITE	50	Input
PWDAT[31:0]	50	Input
PRDATA[31:0]	50	Output
PSLVERR	30	Output

Table H-16 shows the MTB AHB interface timing constraints.

Table H-16 MTB AHB interface

Signal	Timing constraint	Direction
MTBSRAMBASE[31:0]	20	Input
HADDRM[31:0]	40	Input
HPROTM[3:0]	40	Input
HSIZEM[2:0]	40	Input
HTRANSM[1:0]	40	Input
HWDATAM[31:0]	40	Input
HBURSTM[2:0]	40	Input
HSELRAMM	40	Input
HSELSFRM	40	Input
HWRITEM	40	Input
HREADYM	40	Input
HRDATAM[31:0]	40	Output
HREADYOUTM	40	Output
HRESPM	40	Output

Table H-17 shows the ETM Top Interface interface timing constraints.

Table H-17 ETM Top interface

Signal	Timing constraint	Direction
ETMFIFOFULLEN	50	Input
ETMPWRUP	50	Output
ETMEN	50	Output
TSVALUEB[47:0]	50	Input
TSCLKCHANGE	50	Input

Table H-18 shows the SRAM MTB signals timing constraints.

Table H-18 SRAM MTB interface

Signal	Timing constraint	Direction
RAMHCLK	1.8	Output
RAMRD[31:0]	1.5	Input
RAMAD[MTBAWIDTH-3:0]	1.8	Output

Table H-18 SRAM MTB interface (continued)

Signal	Timing constraint	Direction
RAMWD[31:0]	1.8	Output
RAMCS	1.8	Output
RAMWE[3:0]	1.8	Output

Table H-19 shows the Miscellaneous MTB signals timing constraints.

Table H-19 Miscellaneous MTB interface

Signal	Timing constraint	Direction
TSTART	50	Input
TSTOP	50	Input

Table H-20 shows the Cross Trigger Interface signals timing constraints.

Table H-20 CTI interface

Signal	Timing constraint	Direction
CTICHIN[3:0]	60	Input
CTICHOUT[3:0]	60	Output
CTIIRQ[1:0]	60	Output

Table H-21 shows the External SAU signals timing constraints.

Table H-21 External SAU interface

Signal	Timing constraint	Direction
IDAUADDR{A,B}	65	Output
IDAUNS{A,B}	40	Input
IDAUNSC{A,B}	40	Input
IDAUID{A,B}	40	Input
IDAUIDV{A,B}	40	Input
IDAUNCHK{A,B}	30	Input

Note

The External SAU interface is fully combinatorial and timing constraints cannot be realistic. When possible, you need to consider synthesis at a level that includes all the logic connected to the interface instead of setting constraints.

Table H-22 shows the State-retention and power-gating interface.

Table H-22 State-retention and power-gating (SRPG) interface

Signal	Timing constraint	Direction
SYSQREQn	60	Input
SYSQACCEPTn	60	Output
SYSQDENY	60	Output
SYSQACTIVE	60	Output
DBGQREQn	60	Input
DBGQACCEPTn	60	Output
DBGQDENY	60	Output
DBGQACTIVE	60	Output

Appendix I
IP-XACT

This appendix describes the location and configuration of the IP-XACT files in the following sections:

I.1 Location of the IP-XACT files

Table I-1 shows the IP-XACT version IEEE 1685.2009 XML core description files that are included for the following levels of hierarchy.

Table I-1 IP-XACT files

Verilog module	IP-XACT description	Description
GREBE.v	GREBE/GREBE.xml	The processor top level.

The .xml files supplied represent the configurable, but unconfigured, descriptions of the named levels of hierarchy. You can find the files in the ipxact/components/arm.com/ directory.

I.2 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 `shared/ipxact/busdefs` directory.

Figure 1-6 on page 1-12 shows the shared directory structure that contains the IP-XACT bus definitions and abstraction definitions that are referenced from the core description.

Appendix J
Revisions

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

Table J-1 Issue A

Change	Location	Affects
First release	-	-

Table J-2 Differences between issue A and issue B

Change	Location	Affects
Updated the directory structure	Figure 1-6 on page 1-12	r1p0
Updated the `GREBE` configurations options	*GREBE configuration options* on page 2-4	r1p0
Added the **nSYSPORESET** and updated the reset descriptions	*Resets* on page 4-7	r1p0
Added DWT, ASIC level, trace port, and core ETM signals	*Interfaces* on page 4-10	r1p0
Updated the **HPROT[6:2]** values	Table 4-9 on page 4-12	r1p0
Updated the **CODEHINT[3:0]** description	Table 4-11 on page 4-14	r1p0
Added the **CURRNS** signal	Table 4-11 on page 4-14 and Table H-2 on page H-2	r1p0
Updated the I/O port signals figures	Figure 4-3 on page 4-18, Figure 4-4 on page 4-18, and Figure 4-5 on page 4-19	r1p0

Table J-2 Differences between issue A and issue B (continued)

Change	Location	Affects
Updated the external SAU interface	*External Secure Attribution Unit Interface* on page 4-19	r1p0
Updated the debug authentication section	*Debug authentication* on page 4-26	r1p0
Updated the **EDBGRQ** and **IRQLATENCY[7:0]** signals	*Miscellaneous signals* on page 4-27	r1p0
Updated the Q-Channel interface	*Power Management Unit - Q-Channel Interface* on page 4-36	r1p0
Updated the security considerations	*Security Considerations* on page 4-48	r1p0
Updated the CoreSight section	*CoreSight* on page 4-50	r1p0
Added clock considerations for implementation	*Clock considerations* on page 5-4	r1p0
Added the `security` and `exclusive` tests	Table 8-1 on page 8-5	r1p0
Updated the test program compilation using the ARM or GCC Compiler section	*Test program compilation using the ARM or GCC Compiler* on page 8-13	r1p0
Updated the execution testbench limitations	*Execution testbench limitations* on page 8-2	r1p0
Updated the running the testbench section	*Running the testbench* on page 8-15	r1p0
Added DSM.	*Running the testbench with the -dsm option* on page 8-16 and Chapter 10 *DSM Generation*	r1p0
Updated the VCD generation section	*VCD generation* on page 8-18	r1p0
Added the exclusive monitor and the IDAU in the testbench structure	*Testbench structure* on page 8-21 and *Execution testbench components* on page 8-25	r1p0
Updated the memory map	*GRBIKMCU memory map* on page 8-23	r1p0
Updated the CMSIS files	Table 8-11 on page 8-28	r1p0
Various updates on the GPIO integration	Appendix B *GRBIKMCU GPIO Integration*	r1p0
Updated the power intent specification	*Power intent specification* on page F-2	r1p0
Updated the CTI	Appendix G *CTI*	r1p0

Table J-3 Differences between issue B and issue C

Change	Location	Affects
Changed product name to Cortex-M23	-	r1p0
Updated the directory structure	Figure 1-6 on page 1-12	r1p0
Updated the `BASEADDR` and `TARGETID` descriptions in `GREBEINTEGRATION_MCU`	Table 2-3 on page 2-9	r1p0

Table J-3 Differences between issue B and issue C (continued)

Change	Location	Affects
Added **nSYSPORESET**	*Key integration tasks* on page 3-3 and Table 4-6 on page 4-8	r1p0
Updated the clock domains diagram	Figure 4-1 on page 4-3	r1p0
Updated the **TRACECLKIN** connection information	Table 4-3 on page 4-5	
Added the **DBGACTIVE** and **DBGTRCENA** signals	Table 4-23 on page 4-27	r1p0
Updated the Power Management Unit - Q-Channel Interface section	*Power Management Unit - Q-Channel Interface* on page 4-36	r1p0
Updated the execution testbench figure	Figure 8-3 on page 8-21	r1p0
Updated the commands in the Test program compilation using the ARM or GCC Compiler section	*Test program compilation using the ARM or GCC Compiler* on page 8-13	r1p0
Updated the command in the Running the testbench section	*Running the testbench* on page 8-15	r1p0
Updated the execution testbench figure	Figure 8-3 on page 8-21	r1p0
Updated the CMSIS files	Table 8-11 on page 8-28	r1p0
Updated the DSM simulation options	Table 10-2 on page 10-6	r1p0
Added Appendix C	Appendix C *CoreSight SoC*	r1p0
Updated the debug driver source files	Table E-1 on page E-2	r1p0
Updated the power domains figure	Figure F-1 on page F-4	r1p0
Updated the Power control signals section	*Power control signals* on page F-5	r1p0