Cortex™-M0+

Revision: r0p1

Integration and Implementation Manual

Confidential

ARM®

ARM DIT 0032B (ID122112)

Cortex-M0+
Integration and Implementation Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
19 January 2012	A	Confidential	First release for r0p0
21 December 2012	B	Confidential	First release for r0p1

Proprietary Notice

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
Cortex-M0+ Integration and Implementation Manual

Preface

This preface introduces the *Cortex-M0+ Integration and Implementation Manual (IIM)*. It contains the following sections:

About this book

This book is for the Cortex-M0+ 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 *Integration Kit*

Read this for a description of the Cortex-M0+ integration kit.

Chapter 9 *Netlist Dynamic Verification*

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

Chapter 10 *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 *CM0PIKMCU GPIO Integration*

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

Appendix C *SysTick Examples*

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

Appendix D *Debug Driver*

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

Appendix E *Power Intent*

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

Appendix F *Processor Integration Layer*

Read this for a description of the Cortex-M0+ *Processor Integration Layer* (PIL) and how you can use it as a reference for your integration.

Appendix G *Cortex-M0+ CTI*

Read this for a description of the Cortex-M0+ *Cross Trigger Interface* (CTI).

Appendix H *Tarmac Trace Description*

Read this for a description of the Tarmac trace file format that can be used to trace program execution of a Cortex-M0+ processor simulation.

Appendix I *Signal Timing Constraints*

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

Appendix J *IP-XACT*

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

Appendix K *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*.
- *Timing diagrams*.
- *Signals* on page x.

Typographical conventions

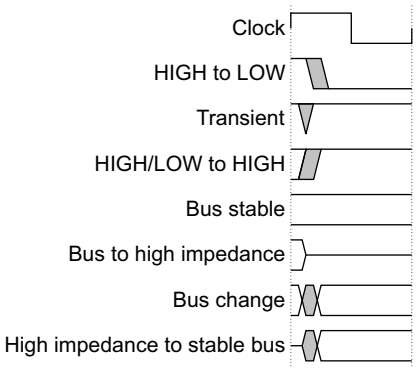
The following table describes the typographical conventions:

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.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *`italic`*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

The figure named *Key to timing diagram conventions* on page x 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-M0+ 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® 3 AHB-Lite™ Protocol* (ARM IHI 0033).
- *ARM CoreSight™ Components Technical Reference Manual* (ARM DDI 0314).
- *ARMv6-M Architecture Reference Manual* (ARM DDI 0419).
- *Cortex-M0+ Technical Reference Manual* (ARM DDI 0484).
- *CoreSight MTB-M0+ Integration and Implementation Manual* (ARM DIT 0031).
- *Cortex-M0+ Release Note* (AT590-DC-06003).

- *Cortex-M0+ Reference Methodology Release Note* (AT590-RM-00002, AT590-RM-70000).
- *CoreSight Architecture Specification* (ARM IHI 0029).
- *CoreSight SoC User Guide* (ARM DUI 0563).
- *ARM Debug Interface v5* (ARM IHI 0031A).
- *ARM Debug Interface Architecture Specification ADIv5.0 to ADIv5.2* (ARM IHI 0031B).

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 0032B.
- 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-M0+ processor. It contains the following sections:

1.1 About the processor

The Cortex-M0+ 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-M0+ 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 for the Cortex-M0+* (MTB-M0+).

The Cortex-M0+ 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. The r0p1 release (and later) also includes an optimized *Cross Trigger Interface* (CTI) component to allow debug cross-triggering with other devices in System-on-Chip implementations.

The Cortex-M0+ processor supports the following levels of hierarchy for integration or implementation:

- Processor component level, `CORTEXM0PLUS`.
- Processor, DAP and *Wakeup Interrupt Controller* (WIC) integration level, `CM0PINTEGRATION`.
- Processor, DAP, WIC, and *Micro Trace Buffer* (MTB) integration level, `CM0PMTBINTEGRATION`.
- Processor, DAP, WIC, MTB, and CTI level, `CORTEXM0PLUSINTEGRATIONCS`.

Figure 1-1 shows the first three levels and the main interfaces of the processor.

Figure 1-2 on page 1-4 shows the `CORTEXM0PLUSINTEGRATIONCS` level and the main interfaces of the processor.

Note

You can modify the `CM0PINTEGRATION` and `CM0PMTBINTEGRATION` levels for your own requirements.

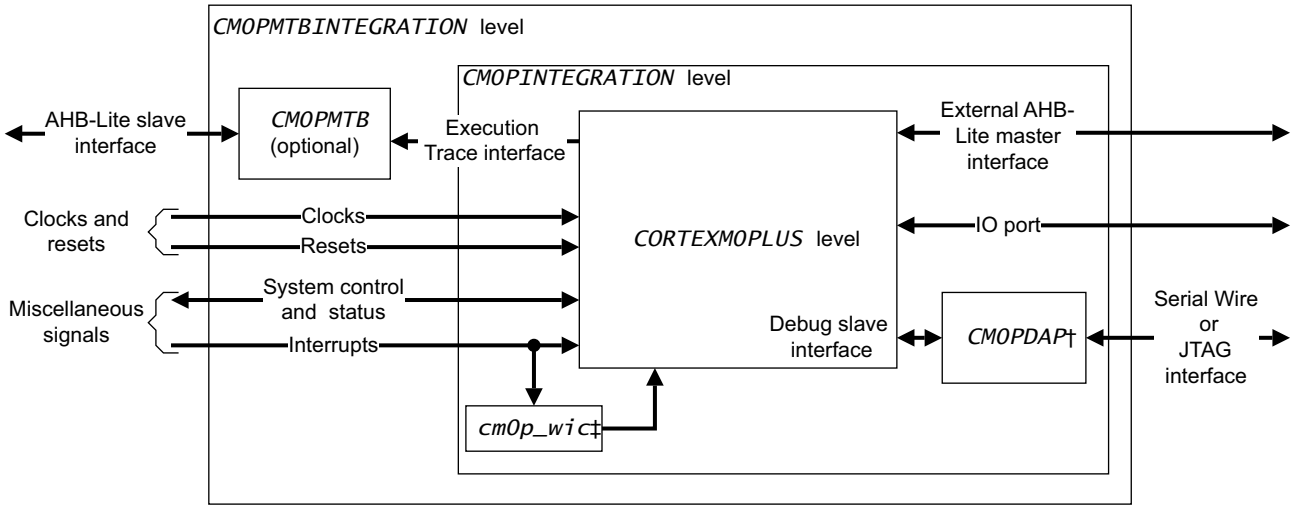


Figure 1-1 Module hierarchy and the main interfaces

CORTEXM0PLUS component level

Contains the basic processor. This level is not modifiable.

CM0PINTEGRATION level

Instantiates the CORTEXM0PLUS component level and provides example integration with the Cortex-M0+ *Debug Access Port* (DAP) and WIC. This level is provided as a working example of a single-processor sub-system. You can modify it to suit your requirements.

CM0PMTBINTEGRATION level

Instantiates the CM0PINTEGRATION component level and provides example integration with the CoreSight Micro Trace Buffer for the Cortex-M0+ processor (CoreSight MTB-M0+). The CoreSight MTB-M0+ is licensed and delivered separately from the Cortex-M0+ processor and DAP. This level is provided as a working example of a single-processor sub-system. You can modify it to suit your requirements.

CORTEXM0PLUSINTEGRATIONCS level

Instantiates the CORTEXM0PLUS component level and provides example integration with the WIC, Cortex-M0+ Cross Trigger Interface and the optional CoreSight MTB-M0+ Micro Trace Buffer. This level is provided as a working example of a processor subsystem that you can integrate into a larger System on Chip device. This level supports integration into large System on Chip designs using the ARM CoreSight SoC tool. See Appendix F *Processor Integration Layer*.

The Cortex-M0+ processor is highly configurable. Although the top-level signals at the CORTEXM0PLUSINTEGRATIONCS, CM0PMTBINTEGRATION, CM0PINTEGRATION and CORTEXM0PLUS levels are independent of configuration, the functionality of these interfaces is dependent on 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 CM0PMTBINTEGRATION, CM0PINTEGRATION or CORTEXM0PLUS component levels. See Chapter 2 *Configuration Guidelines*.

The Cortex-M0+ processor is supplied with an example implementation wrapper:

- CORTEXM0PLUSIMP for the CORTEXM0PLUS level.

This enables you to configure the Cortex-M0+ processor to your requirements and to change the pinout to match the SRPG requirements of your cell library.

Note

- The Cortex-M0+ product includes some example user modifiable Verilog RTL files, including the CM0PMTBINTEGRATION.v and CM0PINTEGRATION.v files described in this section. Files within the cortexm0plus, cm0p_dap and cm0p_cti directories must not be modified.
- If you also licence the CoreSight MTB-M0+, files within the cm0p_mtb directory must not be modified.

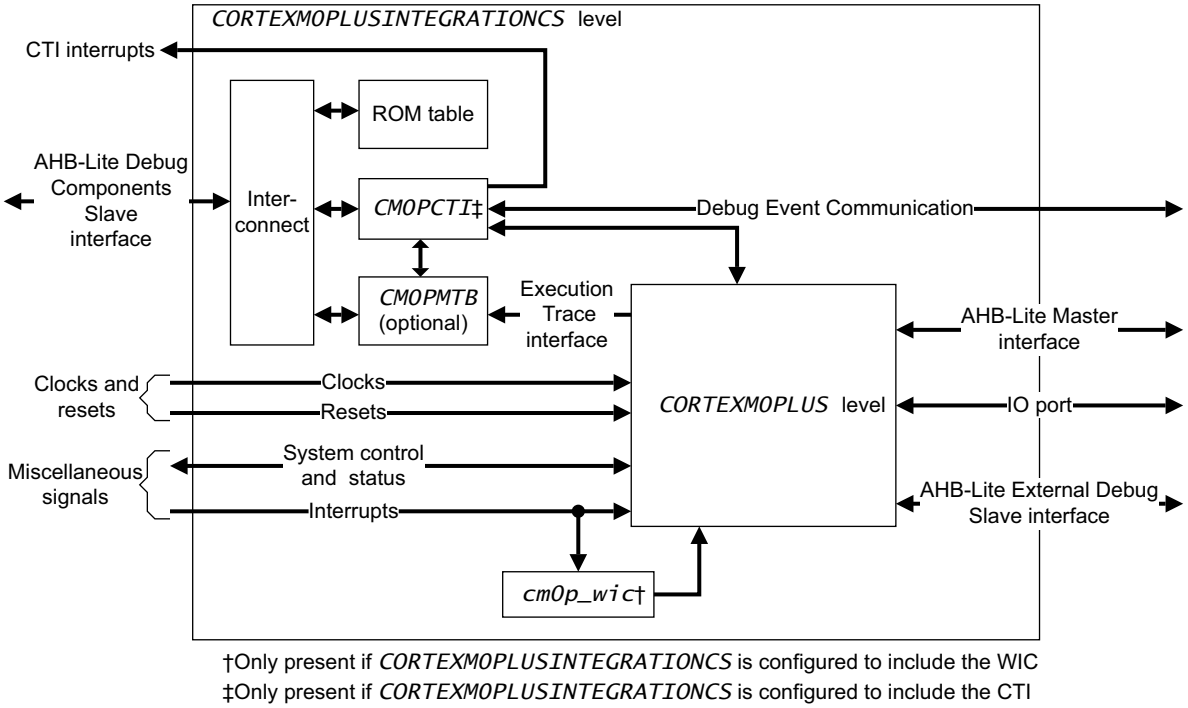


Figure 1-2 CORTEXM0PLUSINTEGRATIONCS module hierarchy and the main interfaces

1.2 About integration and implementation

The flows that you can use to integrate a Cortex-M0+ processor into your SoC can vary depending on the capacity of your tools.

Figure 1-3 shows the integration and implementation flow when you first implement the Cortex-M0+ processor and then integrate the implemented processor into your system.

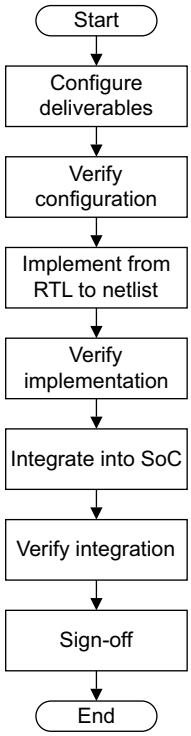


Figure 1-3 Implementation and integration flow

Figure 1-4 on page 1-6 shows the integration and implementation flow when you first integrate the Cortex-M0+ processor into your system and then implement your system.

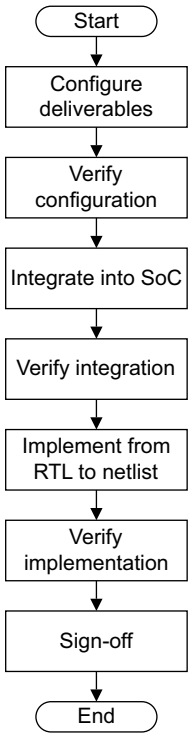


Figure 1-4 Integration and implementation flow

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 additional elements in your design, the main connection is the AHB-Lite bus that conforms to the AMBA 3 AHB-Lite Protocol.

1.2.2 Implementation

Figure 1-5 on page 1-7 shows the implementation process, including the top-level inputs, resources, outputs, and controls and constraints for implementation.

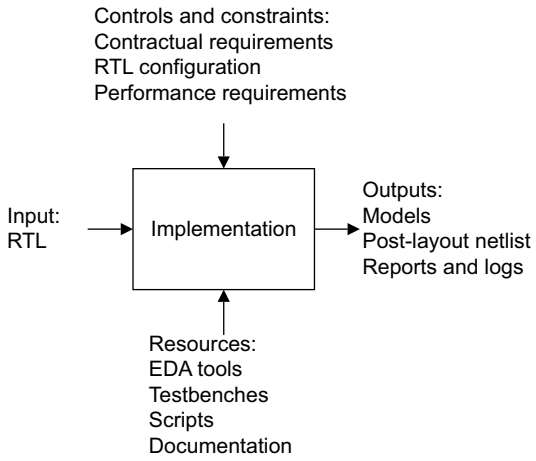


Figure 1-5 Implementation process

For an overview of the implementation process, see:

- *Implementation resources*.
- *Implementation controls and constraints*.
- *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 *Cortex-M0+ Release Note* for a list of deliverables and any specific tool revisions required for implementation.

Note

The *Cortex-M0+ 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 and constraints

Figure 1-5 shows the general controls and constraints that apply to the implementation. You must implement the device in accordance with your contract, see *Implementation obligations* for more information.

Implementation obligations

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 licences granted to Licensee.

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

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

Implementation inputs

The *Cortex-M0+ 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 1-6 on page 1-9 shows the implementation flow.

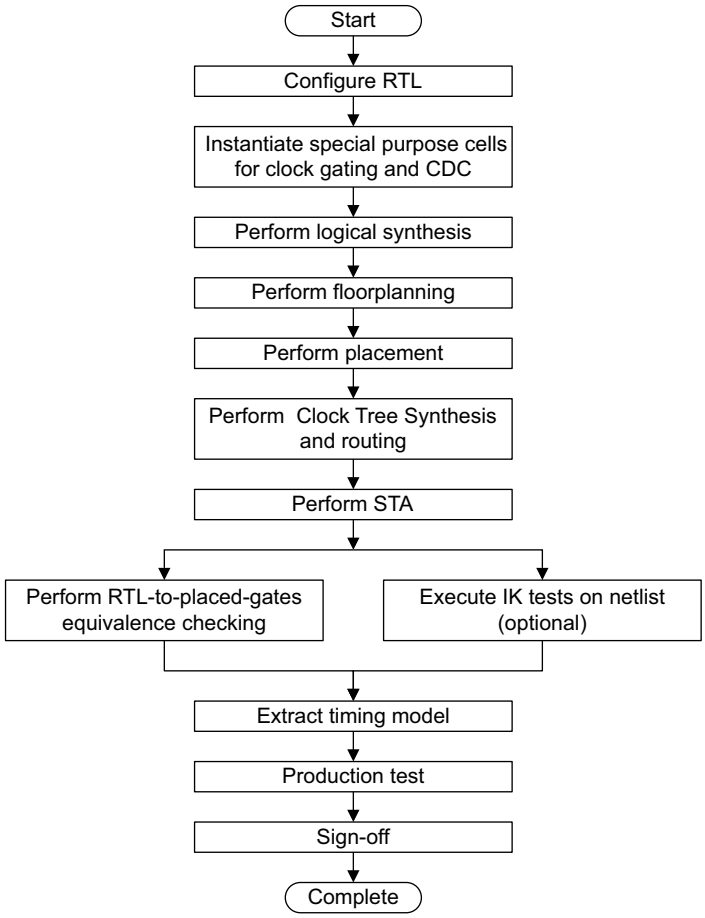


Figure 1-6 Implementation flow

Note

For information on contractual obligations to complete sign-off as part of the completed flow, see *Implementation controls and constraints* 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 10 *Sign-off* for more information.

1.4 Reference data

Before starting, you must ensure the unpacked deliverables are located in the correct directory structure. Figure 1-7 shows the principal directory structure when you unpack the deliverables.

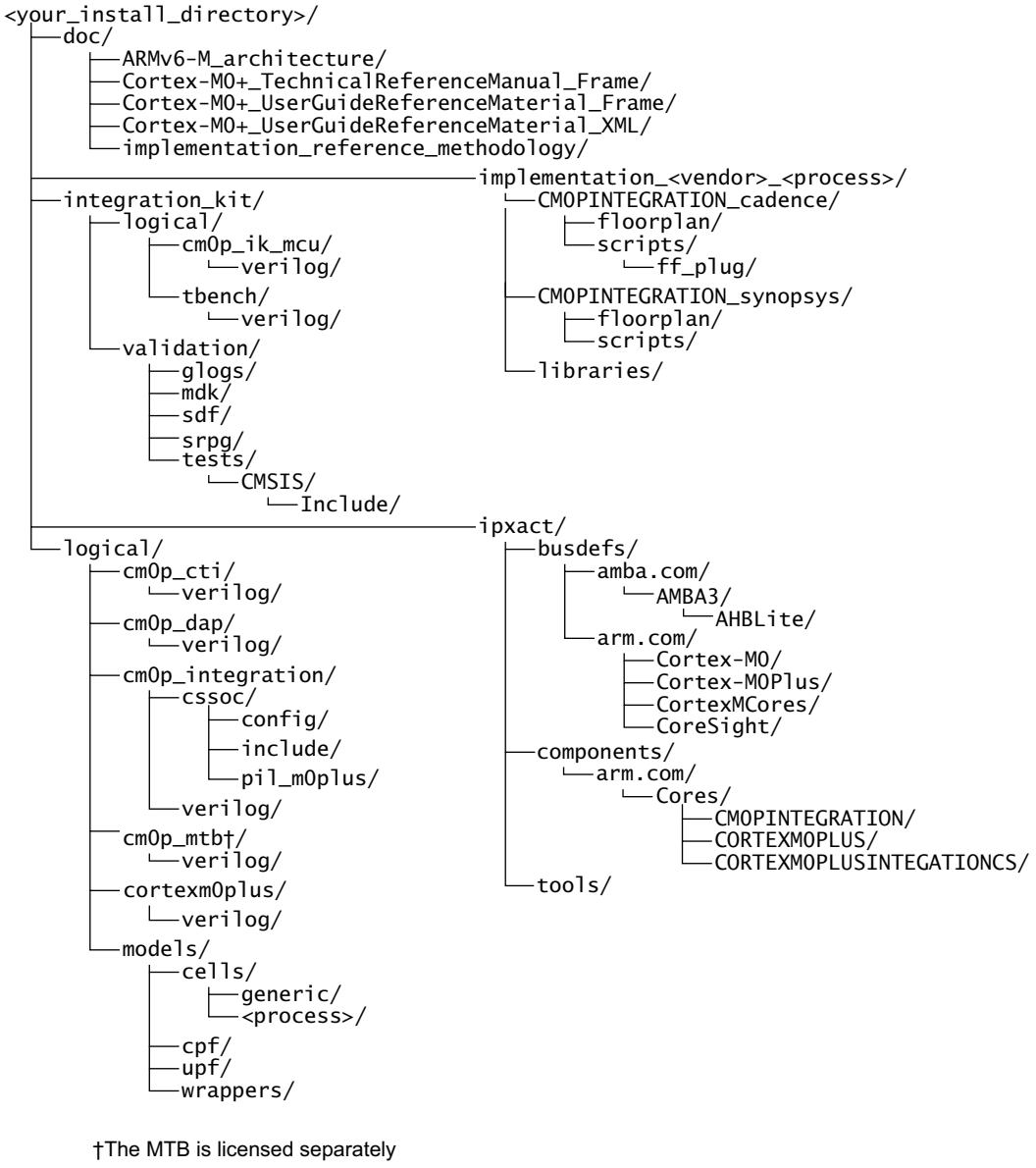


Figure 1-7 Directory structure

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-M0+ RTL includes the following configurable integration levels:

- The CORTEXM0PLUS level implements the Cortex-M0+ processor, the *Nested Vectored Interrupt Controller* (NVIC), breakpoint unit, watchpoint unit and associated debug control.
- The example CM0PINTEGRATION level instantiates a CORTEXM0PLUS level pre-integrated with the WIC, and the Cortex-M0+ Serial Wire or JTAG DAP.
- The example CM0PMTBINTEGRATION level instantiates the CM0PINTEGRATION level pre-integrated with the optional CoreSight MTB-M0+.
- The example CORTEXM0PLUSINTEGRATIONCS level instantiates:
  - CORTEXM0PLUS level pre-integrated with the WIC.
  - An optional CoreSight MTB-M0+.
  - An optional Cortex-M0+ Cross Trigger Interface.
  - A ROM table and associated logic.

  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-M0+ subsystem as part of a larger System on Chip (SoC) design, ARM recommends that you base your design on the example CORTEXM0PLUSINTEGRATIONCS level. If you are building a single-processor design based on Cortex-M0+ that requires the debug features of the processor, ARM recommends that you base your design on either CM0PINTEGRATION or CM0PMTBINTEGRATION. 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 CORTEXM0PLUS level.

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

At all levels, parameters control the configurable options.

The CORTEXM0PLUS level is not modifiable. You can control the parameter values for the CORTEXM0PLUS level by defining them when instantiating the CORTEXM0PLUS level. The file CORTEXM0PLUSIMP.v, in the logical/models/wrappers directory, provides an example wrapper instantiating CORTEXM0PLUS. You can also use this wrapper file to specify additional inputs and outputs to the processor, for example SRPG or scan test signals.

You can control the parameter values for the CM0PINTEGRATION level by either modifying CM0PINTEGRATION.v, or by defining the parameter values when you instantiate this level. This module is located incm0p_integration/verilog.

You can control the parameter values for the CM0PMTBINTEGRATION level by either modifying CM0PMTBINTEGRATION.v, or by defining the parameter values when you instantiate this level. This module is located incm0p_integration/verilog.

You can control the parameter values for the CORTEXM0PLUSINTEGRATIONCS level by either modifying CORTEXM0PLUSINTEGRATIONCS.v, or by defining the parameter values when you instantiate this level, for example when using the ARM CoreSight SoC product. This module is located in cm0p_integration/verilog.

The Reference Methodology deliverables include a copy of the implementation wrappers that might have implementation flow specific modifications. See the *Cortex-M0+ Reference Methodology Release Note* for the location of the implementation wrappers used for implementation. Use these files to replace the files in the `logical/models/wrappers` directory.

2.2 Configuration options

The following sections describe the Cortex-M0+ processor configuration options:

- *CORTEXM0PLUS configuration options*.
- *CM0PINTEGRATION configuration options* on page 2-6.
- *CM0PMTBINTEGRATION configuration options* on page 2-9.
- *CORTEXM0PLUSINTEGRATIONCS configuration options* on page 2-10.

2.2.1 CORTEXM0PLUS configuration options

Table 2-1 shows the configuration options summary for CORTEXM0PLUS.

Table 2-1 CORTEXM0PLUS options summary

Parameter	Default value	Supported values	Description
ACG	1	0, 1	Specifies if internal architectural clock gates are included to minimize dynamic power dissipation: 0 Exclude architectural clock gates. 1 Include architectural clock gates. **Note** If you include architectural clock gating, you must replace the provided model, cm0p_acg.v, with a direct instantiation of a clock gating cell from your target cell library.
AHBSLV	1	0, 1	Specifies the bus protocol implemented on the SLV port. This is a debug port. See Chapter 4 *Functional Integration Guidelines* for additional information: 0 The SLV port implements a Cortex-M0+ DAP specific protocol. 1 The SLV port implements a subset of AHB-Lite. ARM recommends that you set this parameter to 1 if you implement the CORTEXM0PLUS level for integration with any debug infrastructure other than the Cortex-M0+ DAP, for example, the generic CoreSight DAP.
BE	0	0, 1	Specifies the endianness for data transfers: 0 Little-endian. 1 Byte-invariant big-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	1	0, 1	Specifies whether or not the debug extensions are implemented: 0 Exclude debug functionality. 1 Include debug functional. The configuration of debug is controlled additionally by the BKPT and WPT parameters. **Note** Setting the DBG parameter to 0 can result in the **DCLK** input becoming unloaded.
HWF	0	0, 1	Half-word fetching only: **0** Fetch instructions using 32-bit AHB-Lite accesses whenever possible. **1** Fetch instructions using only 16-bit AHB-Lite accesses.

Table 2-1 CORTEXM0PLUS options summary (continued)

Parameter	Default value	Supported values	Description
IOP	0	0, 1	I/O port: 0 Exclude I/O port functionality. 1 Include I/O port functionality.
IRQDIS	0	-	Disables support for individual iterrupts.[a] IRQDIS[i] disables **IRQ[i]**, for example: 32'h00000000 No **IRQ** disabled. 32'h0000FFFF **IRQ[15:0]** disabled.
MPU	0	0, 8	Specifies the number of implemented *Memory Protection Unit* (MPU) regions: 0 Exclude MPU functionality. 8 Include MPU functionality (Eight MPU regions).
NUMIRQ	32	0-32	Specifies the highest interrupt number (NUMIRQ-1) of implemented user interrupts: 0 No functional IRQ lines 1 **IRQ[0]**. 2 **IRQ[1:0]**. **...** 32 **IRQ[31:0]**. NUMIRQ should be used in conjunction with IRQDIS.[a]
RAR	0	0, 1	Specifies whether all synchronous states or only architecturally required states are reset: 0 Only architecturally required state is reset. 1 All state is reset. **Note** • When RAR is 1 all registers in the design can be reset, incurring an area penalty. • When RAR is 0 the registers in the design, that do not require a reset, have no reset.
SMUL	0	0, 1	Specifies the implemented multiplier: 0 Include the fast, single-cycle multiplier. 1 Include the small, 32-cycle multiplier.
SYST	1	0, 1	Specifies whether or not the SysTick timer functionality is included: 0 Exclude the SysTick timer. 1 Include the SysTick timer.
USER	0	0, 1	Unprivileged/Privileged support:[b] 0 Exclude Unprivileged/Privileged support (that is, all accesses are Privileged). 1 Include Unprivileged/Privileged support.
VTOR	0	0, 1	Vector Table Offset Register: 0 Exclude VTOR. 1 Include VTOR.

Table 2-1 CORTEXM0PLUS options summary (continued)

Parameter	Default value	Supported values	Description
WIC	1	0, 1	Specifies whether or not the WIC interface is implemented: 0 Exclude the WIC interface. 1 Include the WIC interface.
WICLINES	34	2-34	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. ... 34 **NMI**, **RXEV,** and **IRQ[31:0]** are supported. **Note** If you exclude the WIC interface, this parameter has no effect. See the WIC parameter description in this table.
WPT	2	0-2	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.

a. NUMIRQ-1 specifies the highest interrupt number. You can disable individual interrupt implementation by setting the associated bit in IRQDIS. This allows IRQs to be implemented in non-contiguous ranges up to a maximum of 32 interrupts for example, if NUMIRQ=6 and IRQDIS=3 **IRQ[5:2]** are implemented.

b. When the MPU is included, the USER parameter is ignored and Unprivileged/Privileged support is automatically included.

2.2.2 CM0PINTEGRATION configuration options

CM0PINTEGRATION instantiates the CORTEXM0PLUS level and some additional components. See Figure 1-1 on page 1-2. It includes configuration options available at the CORTEXM0PLUS 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 CM0PINTEGRATION options summary

Parameter	Default value	Supported values	Description
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-M0+ processor. If you implement any additional CoreSight components, such as the CoreSight MTB-M0+, or want debug tools to be able to uniquely identify your system, you must override this value. See *CoreSight ROM tables* on page 4-34. **Note** If you exclude debug, this parameter has no effect. See the DBG parameter description in Table 2-1 on page 2-4.

Table 2-2 CM0PINTEGRATION options summary (continued)

Parameter	Default value	Supported values	Description
HALTEV	0	0, 1	Specifies if the DAP includes halt event signalling support: 0 Exclude halt event signalling support. 1 Include halt event signalling support. If you include this feature, the Debug Port (DP) Event Status Register is implemented in the Cortex-M0+ 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.
JTAGnSW	0	0, 1	Specifies the external debug protocol implemented by the Cortex-M0+ DAP: 0 Implement Serial Wire. 1 Implement JTAG. **Note** • If you implement the JTAG protocol, the Cortex-M0+ DAP returns the DPIDR value 0x0BC11477 unless modified by an *Engineering Change Order* (ECO). The JTAG IDCODE scan chain returns 0x0BA01477. • If you implement the Serial Wire protocol, the Cortex-M0+ DAP returns a DPIDR value that is dependent on 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.
MPU	0	0, 8	At this level additionally specifies if the DAP includes Cacheable/Bufferable attribute support to allow it to generate accesses with the same range of attributes as the processor with an MPU: 0 Exclude Cacheable/Bufferable attribute support. 8 Include Cacheable/Bufferable attribute support. **Note** If you exclude debug, this parameter has no additional effect at this level. See the DBG parameter description in Table 2-1 on page 2-4.
RAR	0	0, 1	At this level additionally specifies whether the DAP resets all registers or only architecturally required registers: 0 Only architecturally required state is reset. 1 All state is reset. **Note** • When RAR is 1 all registers in the design can be reset, incurring an area penalty. • When RAR is 0 the registers in the design, that do not require a reset, have no reset. • If you exclude debug, this parameter has no additional effect at this level. See the DBG parameter description in Table 2-1 on page 2-4.

Table 2-2 CM0PINTEGRATION options summary (continued)

Parameter	Default value	Supported values	Description
SWMD	0	0, 1	DAP Serial Wire Multi-drop Support: 0 Implement Serial Wire protocol version 1. This enables a debugger to connect to a system with a single Serial Wire device. 1 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-M0+ DAP has a DPIDR value of 0x0BC11477 unless modified by an ECO. • If you implement Serial Wire protocol version 2 (SWMD = 1), the Cortex-M0+ DAP has a DPIDR value of 0x0BC12477 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.
TARGETID	0	-	This parameter defines the contents of the Target Identification register in the Cortex-M0+ DAP that uniquely identifies the system that the Cortex-M0+ 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-M0+ 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** • 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.

Table 2-2 CM0PINTEGRATION options summary (continued)

Parameter	Default value	Supported values	Description
USER	0	0, 1	At this level additionally specifies Unprivileged/Privileged support in the DAP: 0 Exclude Unprivileged/Privileged support (that is, all accesses are Privileged). 1 Include Unprivileged/Privileged support. **Note** If you exclude debug, this parameter has no additional effect at this level. See the DBG parameter description in Table 2-1 on page 2-4.

2.2.3 CM0PMTBINTEGRATION configuration options

CM0PMTBINTEGRATION instantiates the CM0PINTEGRATION level and some additional components. See Figure 1-1 on page 1-2. It includes configuration options available at the CM0PINTEGRATION 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 CM0PMTBINTEGRATION options summary

Parameter	Default value	Supported values	Description
ACG	1	0, 1	At this level additionally specifies if an internal architectural clock gate, connected to the CoreSight MTB-M0+, is included to minimize dynamic power dissipation: 0 Exclude architectural clock gate. 1 Include architectural clock gate. **Note** If you include architectural clock gating, you must replace the provided model, cm0p_acg.v, with a direct instantiation of a clock gating cell from your target cell library.
AWIDTH	32	5-32	Specifies the CoreSight MTB-M0+ SRAM address width
MTB	1	0, 1	Specifies whether or not the CoreSight MTB-M0+ is included: 0 Exclude the CoreSight MTB-M0+. 1 Include the CoreSight MTB-M0+.
USER	0	0,1	At this level additionally specifies Unprivileged/Privileged support in the MTB: 0 Exclude Unprivileged/Privileged support (that is, all accesses are Privileged). 1 Include Unprivileged/Privileged support.

2.2.4 CORTEXM0PLUSINTEGRATIONCS configuration options

CORTEXM0PLUSINTEGRATIONCS instantiates the CORTEXM0PLUS level and some additional components. See Figure 1-2 on page 1-4. It includes configuration options available at the CORTEXM0PLUS level and new configuration options that control behavior at this level. Table 2-4 shows the new configuration options available at this level, and configuration options that have additional behavior at this level.

Table 2-4 CM0PLUSINTEGRATIONCS options summary

Parameter	Default value	Supported values	Description
ACG	1	0, 1	At this level additionally specifies if an internal architectural clock gate, connected to the CoreSight MTB-M0+, is included to minimize dynamic power dissipation: 0 Exclude architectural clock gate. 1 Include architectural clock gate. **Note** If you include architectural clock gating, you must replace the provided model, cm0p_acg.v, with a direct instantiation of a clock gating cell from your target cell library.
AWIDTH	32	5-32	Specifies the CoreSight MTB-M0+ SRAM address width, and additionally defines the size of the SRAM space that is decoded by the Debug Component Slave AH-Lite address decoder within this level.
MTB	0	0, 1	Specifies whether or not the CoreSight MTB-M0+ is included: 0 Exclude the CoreSight MTB-M0+. 1 Include the CoreSight MTB-M0+.
USER	0	0,1	At this level additionally specifies Unprivileged/Privileged support in the MTB: 0 Exclude Unprivileged/Privileged support (that is, all accesses are Privileged). 1 Include Unprivileged/Privileged support.
CTI	1	0,1	Specifies whether or not the Cortex-M0+ CTI is included: 0 Exclude the Cortex-M0+ CTI. 1 Include the Cortex-M0+ CTI.

Chapter 3
Key Integration Points

This chapter describes the key integration points 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-M0+ processor into your SoC design. You must read this chapter in conjunction with the rest of the information in this book, and the information referred to in *Additional reading* on page x.

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 CORTEXM0PLUS component level key integration tasks.

Table 3-1 CORTEXM0PLUS 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 **HRESETn** and **DBGRESETn**[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. • WIC interface.	See *Interfaces* on page 4-9
4.	Tie off the CoreSight ROM table base address.[b]	See Chapter 2 *Configuration Guidelines*
5.	Verify your design using the integration kit.	See Chapter 8 *Integration Kit*

a. Tie LOW if your Cortex-M0+ configuration does not include debug.
b. This is only applicable if you have configured the Cortex-M0+ processor to include debug.

Table 3-2 lists the CM0PINTEGRATION level key integration tasks.

Table 3-2 CM0PINTEGRATION level key integration tasks

Key task		Description
1.	Connect the **FCLK**, **SCLK**, **HCLK**, **DCLK**[a], and **SWCLKTCK**[a] clocks correctly.	See *Clocks* on page 4-3
2.	Connect the **PORESETn**, **HRESETn**, **DBGRESETn**[a], and **nTRST**[b] 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.	See *Interfaces* on page 4-9
4.	Tie off the CoreSight ROM table base address.[c]	See Chapter 2 *Configuration Guidelines*
5.	Verify your design using the integration kit.	See Chapter 8 *Integration Kit*

a. Tie LOW if your Cortex-M0+ configuration does not include debug.
b. Tie LOW if your configuration does not include JTAG. Tie HIGH or tie to **PORESETn**, if you do not use **nTRST** and your configuration includes JTAG.
c. This is only applicable if you have configured the Cortex-M0+ processor to include debug.

Table 3-3 lists the CM0PMTBINTEGRATION level key integration tasks.

Table 3-3 CM0PMTBINTEGRATION level key integration tasks

Key task		Description
1.	Connect the **FCLK**, **SCLK**, **HCLK**, **DCLK**[a], and **SWCLKTCK**[a] clocks correctly.	See *Clocks* on page 4-3
2.	Connect the **PORESETn**, **HRESETn**, **DBGRESETn**[a], and **nTRST**[b] 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.	See *Interfaces* on page 4-9
4.	Tie off or connect the CoreSight MTB-CM0+ interface inputs appropriately:	See the *CoreSight MTB-M0+ Integration and Implementation Manual*
5.	Tie off the CoreSight ROM table base address.[c]	See Chapter 2 *Configuration Guidelines*
6.	Verify your design using the integration kit.	See Chapter 8 *Integration Kit*

a. Tie LOW if your Cortex-M0+ configuration does not include debug.

b. Tie LOW if your configuration does not include JTAG. Tie HIGH or tie to **PORESETn**, if **you do not use nTRST** and your configuration includes JTAG.

c. This is only applicable if you have configured the Cortex-M0+ processor to include debug.

Table 3-4 lists the CM0PMTBINTEGRATION level key integration tasks.

Table 3-4 CORTEXM0PLUSINTEGRATIONCS 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 **PORESETn**, **HRESETn** and **DBGRESETn**[a] [b]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.	See *Interfaces* on page 4-9
4.	Tie off or connect the CoreSight-M0+ CTI interface inputs appropriately:	See Appendix G *Cortex-M0+ CTI*

Table 3-4 CORTEXM0PLUSINTEGRATIONCS level key integration tasks (continued)

Key task		Description
5.	Tie off or connect the CoreSight MTB-CM0+ interface inputs appropriately:	See the *CoreSight MTB-M0+ Integration and Implementation Manual*
6.	Tie off the CoreSight ROM table base address.[c]	See Chapter 2 *Configuration Guidelines*
7.	Verify your design. If you have licenced the ARM CoreSight SoC product, you can use it to produce integration tests.	-

a. Tie LOW if your Cortex-M0+ configuration does not include debug.

b. Tie LOW if your configuration does not include JTAG. Tie HIGH or tie to **PORESETn**, if **you do not use nTRST** and your configuration includes JTAG.

c. This is only applicable if you have configured the Cortex-M0+ 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-M0+ processor clocks in your SoC design in:

- *CORTEXM0PLUS level clocks* on page 4-4.
- *CM0PINTEGRATION and CM0PMTBINTEGRATION level clocks* on page 4-4.
- *Clock gating combinations* on page 4-6.

Figure 4-1 shows the Cortex-M0+ clock domains.

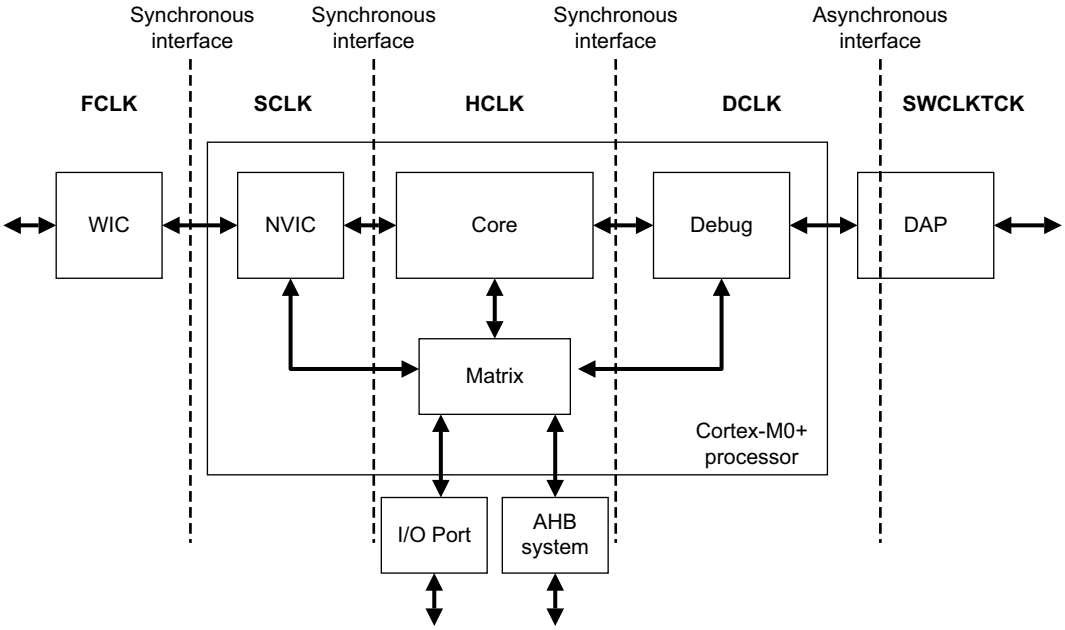


Figure 4-1 Cortex-M0+ clock domains

4.2.1 CORTEXM0PLUS level clocks

Table 4-1 shows the clocks at the CORTEXM0PLUS level of hierarchy.

Table 4-1 CORTEXM0PLUS 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-M0+ processor is configured without debug.

4.2.2 CM0PINTEGRATION and CM0PMTBINTEGRATION level clocks

Table 4-2 shows the clocks of the CM0PINTEGRATION and CM0PMTBINTEGRATION levels of hierarchy.

Table 4-2 CM0PINTEGRATION and CM0PMTBINTEGRATION level clocks

Name	Direction	Description	Connection information
FCLK	Input	Free running clock that clocks the WIC.	**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.

Table 4-2 CM0PINTEGRATION and CM0PMTBINTEGRATION level clocks (continued)

Name	Direction	Description	Connection information
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 `CM0PINTEGRATION` level to detect the presence of an external debugger requesting a connection.
SWCLKTCK	Input	Clock for the JTAG or Serial Wire interface.	**SWCLKTCK** is typically driven by an external debugger and is completely asynchronous to **FCLK**, **HCLK**, **SCLK**, and **DCLK**.

The example PMU module provided, `cm0p_pmu.v`, meets the clocking requirements at the `CM0PINTEGRATION` and `CM0PMTBINTEGRATION` levels.

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** and **SWCLKTCK** must be tied LOW if the Cortex-M0+ processor is configured without debug.

4.2.3 CORTEXM0PLUSINTEGRATIONCS level clocks

Table 4-2 on page 4-4 shows the clocks of the `CORTEXM0PLUSINTEGRATIONCS` level.

Table 4-3 CORTEXM0PLUSINTEGRATIONCS level clocks

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

Table 4-3 CORTEXM0PLUSINTEGRATIONCS 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.

The example PMU module provided, `cm0p_pmu.v`, meets the clocking requirements at the `CORTEXM0PLUSINTEGRATIONCS` 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.

4.2.4 Clock gating combinations

Figure 4-2 shows the legal clock gating combinations.

Gated \ Ungated	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-M0+ processor resets.

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

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

4.3.1 CORTEXM0PLUS level resets

Table 4-4 shows the resets at the `CORTEXM0PLUS` level of hierarchy.

Table 4-4 CORTEXM0PLUS level resets

Name	Direction	Description	Connection information
HRESETn	Input	Reset for the processor system domain and the AHB system	Deassert **HRESETn** synchronously to **SCLK**. Assert **HRESETn** on power-on. Assert **HRESETn** for at least two **HCLK** cycles.
DBGRESETn	Input	Reset for the processor debug domain	Deassert **DBGRESETn** synchronously to **SCLK**. Assert **DBGRESETn** on power-on. Assert **DBGRESETn** for at least two **DCLK** cycles. Tie **DBGRESETn** LOW when no debugger is connected.

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

- Assert **HRESETn** synchronously to **SCLK**, and only when **HREADY** is HIGH. ARM recommends that **HRESETn** is driven from a register clocked by **SCLK** and enabled by **HREADY**.
- Do not assert **DBGRESETn** 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 `CORTEXM0PLUS` level.

4.3.2 CM0PINTEGRATION and CM0PMTBINTEGRATION level resets

Table 4-5 shows the resets of the CM0PINTEGRATION and CM0PMTBINTEGRATION levels of hierarchy.

Table 4-5 CM0PINTEGRATION and CM0PMTBINTEGRATION level resets

Name	Direction	Description	Connection information
HRESETn	Input	Reset for the processor system domain and the AHB system.	Deassert **HRESETn** synchronously to **FCLK**. Assert **HRESETn** on power-on. Assert **HRESETn** for at least two **HCLK** cycles.
DBGRESETn	Input	Reset for the processor debug domain.	Deassert **DBGRESETn** synchronously to **FCLK**. Assert **DBGRESETn** on power-on. Assert **DBGRESETn** for at least two **DCLK** cycles.
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.
PORESETn	Input	Power-on reset for the JTAG or Serial Wire DP logic.	Deassert **PORESETn** synchronously to **FCLK**. External debuggers cannot connect when **PORESETn** is asserted.

The example reset controller provided meets the reset requirements at the CM0PIKMCU level. See *CM0PIKMCU level* on page 8-25 for more information.

4.3.3 CORTEXM0PLUSINTEGRATIONCS level resets

Table 4-6 shows the resets of the CORTEXM0PLUSINTEGRATIONCS level.

Table 4-6 CORTEXM0PLUSINTEGRATIONCS level resets

Name	Direction	Description	Connection information
HRESETn	Input	Reset for the processor system domain and the AHB system.	Deassert **HRESETn** synchronously to **FCLK**. Assert **HRESETn** on power-on. Assert **HRESETn** for at least two **HCLK** cycles.
DBGRESETn	Input	Reset for the processor debug domain.	Deassert **DBGRESETn** synchronously to **FCLK**. Assert **DBGRESETn** on power-on. Assert **DBGRESETn** for at least two **DCLK** cycles.
PORESETn	Input	Power-on reset for the JTAG or Serial Wire DP logic.	Deassert **PORESETn** synchronously to **FCLK**. External debuggers cannot connect when **PORESETn** is asserted.

There are no reset synchronizers within the CORTEXM0PLUSINTEGRATIONCS level.

4.4 Interfaces

This section describes the interfaces of the Cortex-M0+ processor in:

4.4.1 External AHB-Lite interface

This interface is present at the `CORTEXM0PLUS`, `CM0PINTEGRATION`, `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS` 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 *AMBA 3 AHB-Lite 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.
HPROT[3:0]	Output	Connect to the slaves through the bus infrastructure[a].
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.
HMASTLOCK	Output	
HWDATA[31:0]	Output	
HRDATA[31:0]	Input	
HREADY	Input	
HRESP	Input	

a. The **HPROT[1]** bit is tied HIGH if the processor is configured to exclude USER.

AHB-Lite interface transaction types

Table 4-8 shows the Cortex-M0+ 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 half-word load/store	10	1	0	001
Core byte load/store	10	1	0	000
Debug word read/write	10	1	1	010
Debug half-word 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[1] indicates the privilege level of the transaction. For implementations without User support, the signal is always HIGH, indicating Privileged. **HPROT[3:2]** derivation is based on the configuration of the *Memory Protection Unit* (MPU) and debug interface, or for implementations without the MPU, 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 *ARMv6-M Architecture Reference Manual* for more information.

Table 4-9 shows the **HPROT[3:2]** memory types when an MPU is either not present or is disabled.

Table 4-9 Memory types for HPROT[3:2]

Address	Memory type	HPROT[3:2]	Recommended usage
0xF0000000 - 0xFFFFFFFF	Device XN[a]	01	CoreSight ROM tables[b].
0xE0000000 - 0xEFFFFFFF	Reserved	-	System control space.
0xA0000000 - 0xDFFFFFFF	Device XN	01	Peripherals.
0x80000000 - 0x9FFFFFFF	Normal WT[c]	10	Off chip RAM.
0x60000000 - 0x7FFFFFFF	Normal WBWA[d]	11	Off chip RAM.
0x40000000 - 0x5FFFFFFF	Device XN	01	Peripherals.
0x20000000 - 0x3FFFFFFF	Normal WBWA	11	On chip RAM.
0x00000000 - 0x1FFFFFFF	Normal WT[c]	10	ROM or flash.

a. XN means eXecute Never.

b. See *CoreSight ROM tables* on page 4-34.

c. WT means write-through.

d. WBWA means write-back-write-allocate.

AHB-Lite byte lane definition

The Cortex-M0+ processor AHB interface uses the byte lanes defined for a little-endian system in the *AMBA 3 AHB-Lite Protocol Specification*, regardless of the endianness configuration of the processor. 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]	-	-	-
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`.
- **HMASTLOCK** is always LOW because the Cortex-M0+ processor can not generate any locked transactions.
- Instruction fetches are performed as 32 bit word, and 16 bit halfword accesses, if you implement the processor with `HWF=1`. Instruction fetches are performed as 16 bit halfword accesses, only if you implement the processor with `HWF=0`.

4.4.2 AHB interface extensions

This interface is present at the `CORTEXM0PLUS`, `CM0PINTEGRATION`, `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS` levels.

You can use AHB interface extensions in conjunction 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
HMASTER	Output	Enables the system to distinguish between processor initiated and debugger initiated accesses: **LOW** Processor access. **HIGH** Debugger access.
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 address phase does not contain a data-transfer from the processor. **CODEHINT[2]** Indicates that either the current state of the processor, or the next instruction to be executed, does not result in sequential execution. For example, if the next instruction is an unconditional branch, or there is an interrupt pending. **CODEHINT[1]** Indicates that the next instruction to be executed is a currently unresolved conditional backward branch. **CODEHINT[0]** Indicates that the next instruction to be executed is a currently unresolved conditional forward 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[0]** Indicates, in an AHB or I/O port address phase, that the current data load is for an exception vector. **DATAHINT[1]** Indicates, for an AHB address phase generated by the processor, that the current data load was generated by a PC relative load.
SHAREABLE	Output	Indicates that an AHB transaction is shareable: **LOW** Non-shareable. **HIGH** Shareable.
SPECHTRANS	-	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. 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.

AMBA Considerations

You must abide by the rules of AMBA 3 AHB-Lite irrespective of the AHB interface extensions.

4.4.3 AHB memory considerations

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

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

4.4.4 I/O port

This interface is present at the `CORTEXM0PLUS`, `CM0PINTEGRATION`, `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS` levels.

The Cortex-M0+ 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 *CORTEXM0PLUS 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 4-11.

The Cortex-M0+ integration kit provides an example general purpose I/O block suitable for connection to the I/O port interface. See *Testbench structure* on page 8-23.

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.
IOSIZE[1:0]	Output	I/O port transaction size: **0b00** Byte, 8-bit. **0b01** Halfword, 16-bit. **0b10** Word, 32-bit. **0b11** Never used.

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

Name	Direction	Description
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.

Figure 4-3 shows two single read accesses. In this example, the IO device asserts **IOMATCH** and the processor responds, in the same cycle, by asserting **IOTRANS**.

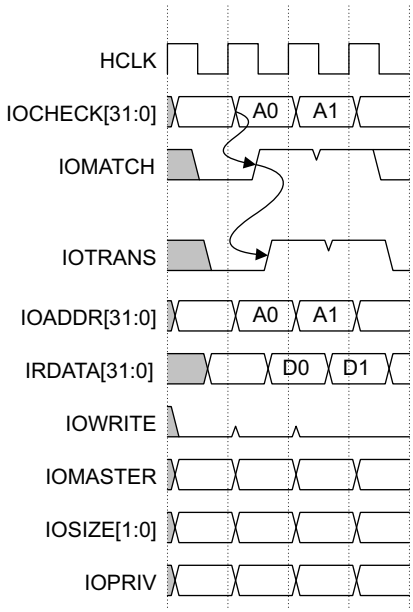


Figure 4-3 IO port read timing single read

Figure 4-4 on page 4-16 shows two single write accesses. In this example, the IO device asserts **IOMATCH** and the processor responds, in the same cycle, by asserting **IOTRANS**.

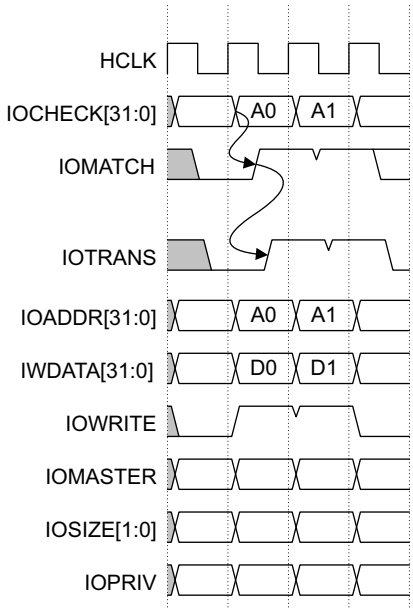


Figure 4-4 IO port write timing single writes

Figure 4-5 shows a multiple write access. In this example, the processor asserts IOTRANS in the cycle after the IO device asserts IOMATCH.

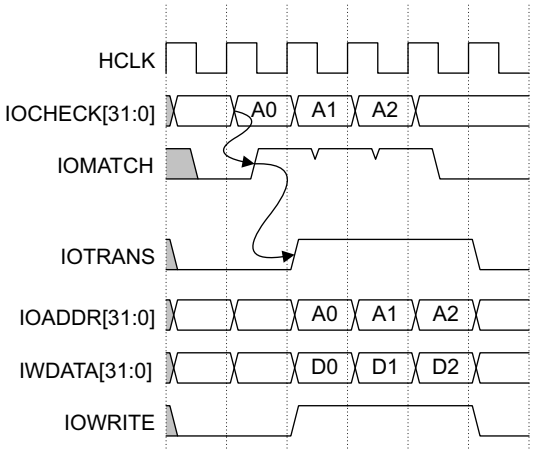


Figure 4-5 IO port write timing store multiple

4.4.5 Interrupt interface

This interface is present at the `CORTEXM0PLUS`, `CM0PINTEGRATION`, `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS` levels.

Table 4-13 shows the signals of the external interrupt interface, and describes how you must connect the signals in your SoC design.

Table 4-13 Interrupt signals

Name	Direction	Description	Connection information
IRQ[31:0]	Input	External interrupt signals.	When the processor is implemented with fewer than 32 external interrupts, the unused **IRQ** inputs must be tied LOW. When correctly configured by software, the Cortex-M0+ 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 `CM0PINTEGRATION`, `CM0PMTBINTEGRATION`, or `CORTEXM0PLUSINTEGRATIONCS` 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-M0+ processor takes an NMI exception in response to **NMI** being held HIGH, or being HIGH for a single cycle of: • **FCLK** for the `CM0PINTEGRATION`, `CM0PMTBINTEGRATION`, or `CORTEXM0PLUSINTEGRATIONCS` level configured to include WIC support for the **NMI**. • **SCLK** for all other cases.

The Cortex-M0+ processor does not implement synchronizers for the **IRQ[31:0]** and **NMI** inputs. If you want to use asynchronous interrupts, you must implement external synchronizers to reduce the possibility of metastability issues.

4.4.6 External debugger interface

The external debugger interface is only present at the `CM0PINTEGRATION` and `CM0PMTBINTEGRATION` levels.

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

Although you can configure `CM0PINTEGRATION` and `CM0PMTBINTEGRATION` 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 CM0PDAP. 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*.
- The Serial Wire interface, see *Serial Wire signals* on page 4-18.

JTAG signals

Table 4-14 on page 4-18 lists the JTAG interface signals.

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

Table 4-14 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 4-15 lists the Serial Wire signals.

Table 4-15 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 4-16 lists the DAP signals.

Table 4-16 Miscellaneous DAP signals

Name	Direction	Description	Connection information
HALTED	Input	Processor Halted.	Connect to Cortex-M0+ processor **HALTED** output.

Table 4-16 Miscellaneous DAP signals (continued)

Name	Direction	Description	Connection information
DEVICEEN	Input	Enable debugger access to system through SLV interface.	Tie this signal HIGH not to use debug authentication to authenticate debugger access to devices on the SLV bus. Tie this signal LOW to permanently disable debugger access. Connect this signal to your own logic, such as that connected to the Cortex-M0+ 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 CM0PDAP 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 CM0PDAP with multi-drop, tie this input to 0b0000.
SWDETECT	Output	Serial Wire protocol detected.	HIGH for one cycle of **SWCLKTCK** when a Serial Wire CM0PDAP is implemented, and a Serial Wire line reset sequence is attempted while the CM0PDAP is not in dormant state. Optionally, connect to your own logic. For example, to disable a JTAG test device when a Serial Wire CM0PDAP is implemented without multi-drop. See Figure 4-6.

Figure 4-6 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.

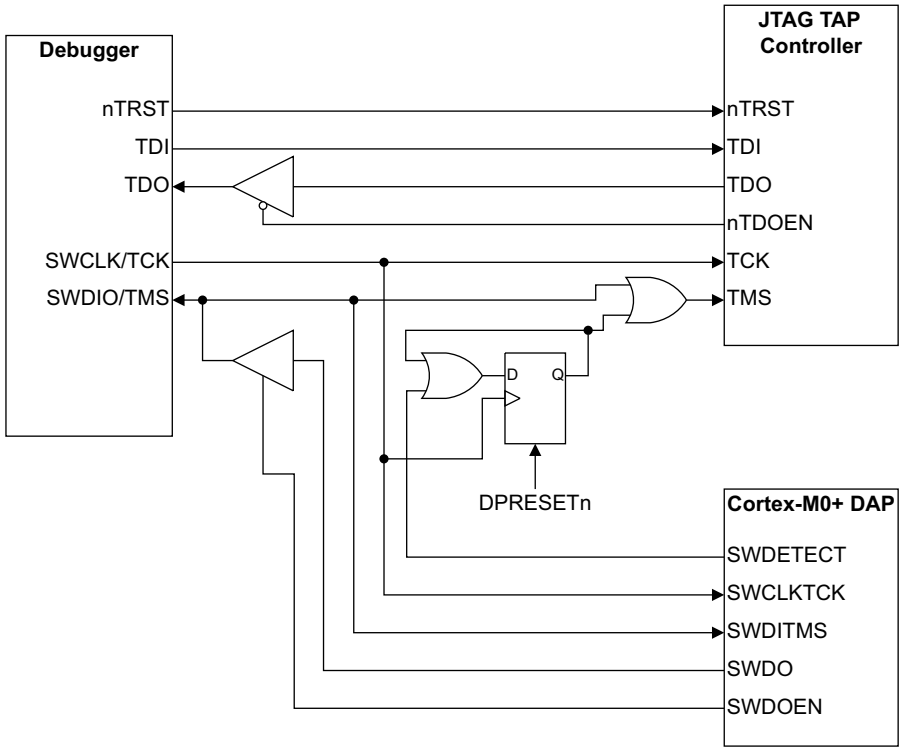


Figure 4-6 Example SWDETECT logic

4.4.7 Debug slave interface

The debug slave interface is only present at the CORTEXM0PLUS level. It enables debugger access to processor control resources and system memory.

Table 4-17 shows the signals for the debug slave interface on the Cortex-M0+ processor. These signals must be connected to either the Cortex-M0+ 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-M0+ 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 4-17 Slave Port interface signals

Name	Direction	Connection information
SLVRDATA[31:0]	Output	Connect to the **SLVRDATA** port of the Cortex-M0+ DAP, or the **HRDATA** port of a CoreSight AHB-AP.
SLVREADY	Output	Connect to the **SLVREADY** port of the Cortex-M0+ DAP, or the **HREADY** port of a CoreSight AHB-AP.
SLVRESP	Output	Connect to the **SLVRESP** port of the Cortex-M0+ 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.
SLVADDR[31:0]	Input	Connect to the **SLVADDR** port of the Cortex-M0+ DAP, or the **HADDR** port of a CoreSight AHB-AP.
SLVTRANS[1:0]	Input	Connect to **SLVTRANS** port of the Cortex-M0+ DAP or the **HTRANS** port of a CoreSight AHB-AP.
SLVWRITE	Input	Connect to the **SLVWRITE** port of the Cortex-M0+ DAP, or the **HWRITE** port of a CoreSight AHB-AP.
SLVWDATA[31:0]	Input	Connect to the **SLVWDATA** port of the Cortex-M0+ DAP, or the **HWDATA** port of a CoreSight AHB-AP.
SLVSIZE[1:0]	Input	Connect to the **SLVSIZE** port of the Cortex-M0+ DAP, or the **HSIZE[1:0]** port of a CoreSight AHB-AP.
SLVPROT[3:0]	Input	Connect to the **SLVPROT** port of the Cortex-M0+ DAP, or the **HPROT[3:0]** port of a CoreSight AHB-AP.

4.4.8 Debug authentication

You can use the **DBGEN** and **NIDEN** inputs to disable trace and prevent the **EDBGRQ** output going active.

Table 4-18 lists the debug authentication signals and describes how to connect them.

Table 4-18 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.

The **DBGEN** and **NIDEN** inputs must be synchronous to the **HCLK** input, and only be modified on any **HCLK** where **HREADY** is HIGH. ARM recommends that if **DBGEN** and **NIDEN** are not static, they are registered through a register clocked by **HCLK** and enabled by **HREADY**.

4.4.9 Miscellaneous signals

These signals are present at the `CORTEXM0PLUS`, `CM0PINTEGRATION`, `CM0PMTBINTEGRATION, and CORTEXM0PLUSINTEGRATIONCS` levels unless otherwise stated.

Table 4-19 shows the miscellaneous signals present on the processor, and describes how to connect the signals.

Table 4-19 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.
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. This signal is not present at the `CORTEXM0PLUSINTEGRATIONCS` level
DGBRESTARTED[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. This signal is not present at the `CORTEXM0PLUSINTEGRATIONCS` level
ECOREVNUM[19:0][c] **ECOREVNUM[27:0]**[d] **ECOREVNUM[31:0]**[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-M0+.	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.

Table 4-19 Miscellaneous signals (continued)

Name	Direction	Description	Connection information
EDBGRQ[a]	Input	External debug request.	This signal can be asserted by a debug agent in the system, for example the CoreSight MTB-M0+, to request that the core enters Debug state. The Cortex-M0+ 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 *ARMv6-M Architecture Reference Manual* for more information.
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 `CM0PINTEGRATION` and `CM0PMTBINTEGRATION` 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 Cortex-M0+ DAP if you are using the Cortex-M0+ DAP **HALTED** input and if you are using the DAP Halt Event Signalling feature. If you are using this signal to connect to a CTI, contact ARM for connection information.
IRQLATENCY[7:0]	Input	The Cortex-M0+ 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, 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. For non-zero wait state memory, zero jitter can be achieved with a higher value on **IRQLATENCY**. **IRQLATENCY** is sampled synchronously to **SCLK**.

Table 4-19 Miscellaneous signals (continued)

Name	Direction	Description	Connection information
LOCKUP	Output	When HIGH, indicates that the processor is in the architected Lockup state, as the result of an unrecoverable exception. See the *ARMv6-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 **HRESETn**. If your system executes instructions from a programmable memory, for example flash, after power up, you must consider how that memory is programmed. The processor might enter Lockup state very quickly if the memory is uninitialized. If **HRESETn** 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 **HRESETn** because of **LOCKUP**. If you require entry into Lockup state to reset the system, your code must enable the functionality in your watchdog unit.
RXEV	Input	A HIGH level on this input causes the ARM v6-M architecture defined Event Register to be set in the Cortex-M0+ 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.
SLEEPDEEP	Output	Active only when **SLEEPING** is HIGH. Indicates that the **SLEEPDEEP** bit in the NVIC is set to 1.	Use this signal in your Power Management Unit, to determine when the processor is requesting DEEPSLEEP.
SLEEPHOLDACKn[f]	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.	Use this to cleanly power off the processor or safely shut down PLLs for deeper levels of sleep.
SLEEPHOLDREQn[f]	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.	Use this, in conjunction with **SLEEPHOLDACKn**, to cleanly power off the processor or safely shut down PLLs for deeper levels of sleep.

Table 4-19 Miscellaneous signals (continued)

Name	Direction	Description	Connection information
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.	Use this signal in your PMU to control power in your system.
SLVSTALL	Input	Drive HIGH to stall accesses on the SLV port.	If you are not using this signal, tie it LOW.
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.	Factor this signal into your reset controller. See *Resets* on page 4-7.
TXEV	Output	A single **SCLK** cycle HIGH level is generated on this output every time an `SEV` instruction is executed on the Cortex-M0+ processor.	Use this to signal an event to other ARM processors.

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 `CORTEXM0PLUS`.
d. This bus is **[27:0]** for `CM0PINTEGRATION`.
e. This bus is **[31:0]** for `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS`.
f. 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.

4.4.10 SysTick signals

These signals are present at the `CORTEXM0PLUS`, `CM0PINTEGRATION`, `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS` levels.

The ARMv6-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 **STCALIB** signals are non-functional.

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

In the Cortex-M0+ 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 4-25 shows an example asynchronous clock generating **STCLKEN**.

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

Table 4-20 shows the **STCALIB[25:0]** values required for the software developer using the SysTick calibration register in the Cortex-M0+ NVIC memory map.

Table 4-20 SysTick signals

Name	Direction	Description	Connection information
STCALIB[25]	Input	NOREF Indicates that no alternative reference clock source has been integrated.	Tie HIGH if **STCLKEN** has been tied off.
STCALIB[24]	Input	SKEW	Tie this LOW if the system timer clock, the external reference clock, or **SCLK** as indicated by **STCALIB[25]**, can guarantee an exact multiple of 10ms. Otherwise, tie this signal HIGH.
STCALIB[23:0]	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.

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

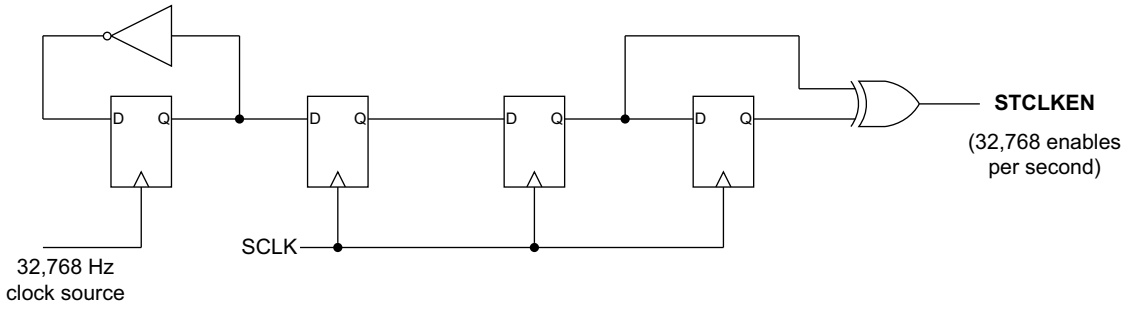


Figure 4-7 Asynchronous SysTick clock example

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

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

4.4.11 WIC interface

Table 4-21 shows the WIC interface signals.

Table 4-21 WIC interface signals

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[31:0]	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.

The WIC interface is only functional if the WIC is included. The interface signals are only present at the `CORTEXM0PLUS` level.

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

Figure 4-8 shows the WIC mode enable sequence. It includes:

- The **WICENREQ** and **WICENACK** PMU interface signals that are present at the `CM0PINTEGRATION`, `CM0PMTBINTEGRATION`, and `CORTEXM0PLUSINTEGRATIONCS` levels. See *Power Management Unit interface* on page 4-27.
- The **WICDSREQn** and **WICDSACKn** WIC signals that are present at the `CORTEXM0PLUS` level only.

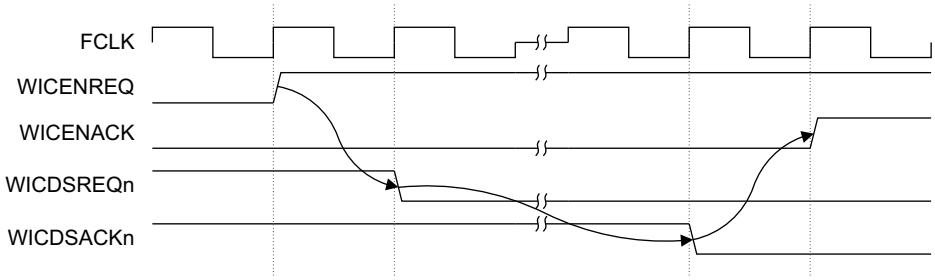


Figure 4-8 WIC mode enable sequence

Figure 4-9 on page 4-27 shows the power down timing sequence. It includes:

- The **SLEEPING** and **SLEEPDEEP** signals. See *Miscellaneous signals* on page 4-21.
- The WIC interface signals that are present at the `CORTEXM0PLUS` level only. See Table 4-21
  - **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 CORTEXM0PLUS, CM0PINTEGRATION, and CORTEXM0PLUSINTEGRATIONCS levels. See Table 4-22 on page 4-28
  — **WAKEUP.**
  — **WICSENSE.**
- The **SYSISOLATEn**, **SYSRETAINn**, and **SYSPWRDOWN** signals that are present for particular libraries only. See *Chapter 2 Configuration Guidelines*.

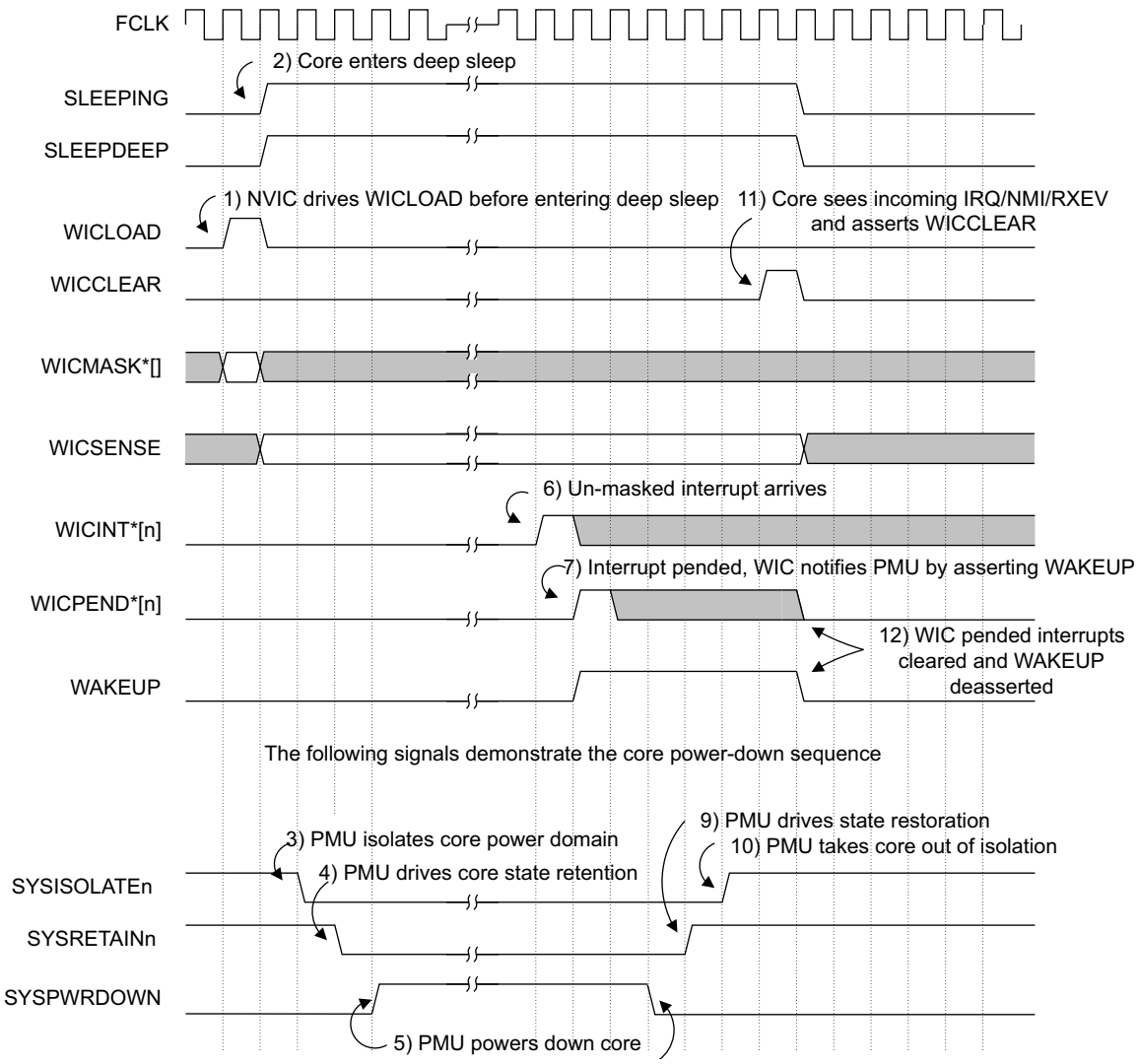


Figure 4-9 Power down timing sequence

4.4.12 Power Management Unit interface

The PMU interface is only present at the CM0PINTEGRATION, CM0PMTBINTEGRATION and CORTEXM0PLUSINTEGRATIONCS levels.

Table 4-22 shows the PMU interface signals.

Table 4-22 PMU interface signals

Name	Direction	Description
WAKEUP[a]	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 cause the WIC to generate the **WAKEUP** signal.
WICENREQ[ab]	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**.
CDBGPWRUPREQ[c]	Output	Active HIGH signal that indicates an external debugger request to the PMU to power-up the debug domain in readiness for a debugging session. This signal must be synchronized before use.
CDBGPWRUPACK[c]	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**.

a. Only functional if the WIC interface is included.
b. Tie LOW if the WIC interface is not included.
c. Not present on the `CORTEXM0PLUSINTEGRATIONCS` level.

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**.

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 4-26 and Figure 4-9 on page 4-27 show how the **WAKEUP**, **WICSENSE**, **WICENREQ**, and **WICENACK** signals are sequenced.

4.4.13 Execution trace

The execution trace signals are only present at the `CORTEXM0PLUS` level. These output signals enable you to connect to the optional CoreSight MTB-M0+ component to provide execution trace. The CoreSight MTB-M0+ is licensed and delivered separately from the Cortex-M0+ processor. See the *CoreSight MTB-M0+ Integration and Implementation Manual* for details of the CoreSight MTB-M0+.

Table 4-23 shows the execution trace interface signals.

Table 4-23 Execution trace signals

Name	Direction	Description
ATOMIC	Output	Instruction address is not instruction related.

Table 4-23 Execution trace signals (continued)

Name	Direction	Description
IAEX[30:0]	Output	Instruction address[31:1] in execute.
IAEXEN	Output	Instruction address enable.
IAEXSEQ	Output	Instruction address is sequential.

4.4.14 Test interface

Table 4-24 shows the functions of the test interface pins.

Table 4-24 Test interface pin functions

Name	Direction	Description
DFTRSTDISABLE	Input	`CM0PINTEGRATION` and `CM0PMTBINTEGRATION` 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. **DFTSE** also bypasses any architectural clock-gate cell instantiations within the Cortex-M0+ processor.

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.

4.4.15 MTB SRAM interface

The MTB SRAM interface is only present at the `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS` levels. See *CoreSight MTB-M0+ Integration and Implementation Manual*.

Table 4-25 MTB SRAM interface

Name	Direction	Description
RAMHCLK	output	MTB RAM clock, optionally gated.
RAMRD	input	MTB RAM read-data.
RAMAD[`AWIDTH`**-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.

4.4.16 MTB trace control

The MTB trace control interface is present at the CM0PMTBINTEGRATION and CORTEXM0PLUSINTEGRATIONCS levels. See *CoreSight MTB-M0+ Integration and Implementation Manual*.

Table 4-26 MTB trace control signals

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

Table 4-27 MTB miscellaneous signals

Name	Direction	Description
SRAMBASEADDR[31:0]	input	Location of MTB RAM in memory map[a]
MTBSRAMBASEADDR[31:0]	input	Location of MTB RAM in memory map[b]

a. **SRAMBASEADDR** is present only at CM0PMTBINTEGRATION.

b. **MTBSRAMBASEADDR** is present only at CORTEXM0PLUSINTEGRATIONC.

4.4.17 MTB AHB interface

The MTB AHB interface is present at the CM0PMTBINTEGRATION level. See *CoreSight MTB-M0+ Integration and Implementation Manual*.

Table 4-28 MTB AHB Interface

Name	Direction	Description
HADDRMTB[31:0]	input	AHB address into MTB.
HPROTMTB[3:0]	input	AHB properties into MTB.
HSIZEMTB[2:0]	input	AHB transaction size into MTB.
HTRANSMTB[1:0]	input	AHB transaction to MTB.
HWDATAMTB[31:0]	input	AHB write-data for MTB.
HSELRAM	input	AHB selects MTB RAM.
HSELSFR	input	AHB selects MTB control registers.
HWRITEMTB	input	AHB write to MTB.
HREADYMTB	input	AHB ready for MTB.
HRDATAMTB[31:0]	output	AHB read-data from MTB.
HREADYOUTMTB	output	AHB ready out of MTB.
HRESPMTB	output	AHB error response from MTB.

4.4.18 Cross trigger interface

The cross trigger interface is present at the CORTEXM0PLUSINTEGRATIONCS level.**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. See also Appendix F *Processor Integration Layer*.

Table 4-29 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.

4.4.19 External debug slave interface

The *External Debug Slave* (EDS) interface is present at the CORTEXM0PLUSINTEGRATIONCS level.

Table 4-30 External debug slave interface

Name	Direction	Description
HADDREDS[31:0]	input	AHB address to EDS.
HBURSTEDS[2:0]	input	AHB burst type to EDS.
HMASTLOCKEDS	input	AHB locked status to EDS.
HPROTEDS[3:0]	input	AHB protection control to EDS.
HSIZEEDS[2:0]	input	AHB transfer size to EDS.
HTRANSEDS[1:0]	input	AHB transfer type to EDS.
HWDATAEDS[31:0]	input	AHB write-data to EDS.
HWRITEEDS	input	AHB write/not read to EDS.
HRDATAEDS[31:0]	output	AHB read-data from EDS.
HREADYEDS	output	AHB ready/advance from EDS.
HRESPEDS	output	AHB error response from EDS.

4.4.20 Debug component slave interface

The *Debug Component Slave* (DCS) interface is present at the CORTEXM0PLUSINTEGRATIONCS level.

Table 4-31 Debug component slave interface

Name	Direction	Description
HADDRDCS[31:0]	input	AHB address to DCS.
HBURSTDCS[2:0]	input	AHB burst type to DCS.
HMASTLOCKDCS	input	AHB locked status to DCS.

Table 4-31 Debug component slave interface (continued)

Name	Direction	Description
HPROTDCS[3:0]	input	AHB protection control to DCS.
HSIZEDCS[2:0]	input	AHB transfer size to DCS.
HTRANSDCS[1:0]	input	AHB transfer type to DCS.
HWDATADCS[31:0]	input	AHB write-data to DCS.
HWRITEDCS	input	AHB write/not read to DCS.
HRDATADCS[31:0]	output	AHB read-data from DCS.
HREADYDCS	output	AHB ready/advance from DCS.
HRESPDCS	output	AHB error response from DCS.

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-M0+ processor may be useful when considering how to prevent unauthorized debug access to all or part of your system.

See *Interfaces* on page 4-9 for additional information on the signals described here.

4.5.1 SLVPROT and HPROTEDS

The **SLVPROT** input to the Cortex-M0+ debug slave port is used to indicate the privilege level of the access being made, either into the processor SCS space or through the processor to the AHB-Lite master port or IOP. **SLVPROT** is named **HPROTEDS** at the `CORTEXM0PLUSINTEGRATIONCS` level.

The ARMv6-M architecture requires that the processor SCS space 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. This behavior is not affected by the value of the `USER` configuration parameter.

Unprivileged accesses to the SCS space 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 SLVSTALL

The **SLVSTALL** input signal to the Cortex-M0+ processor can be used to delay SLV port accesses. You could use this signal to delay debug accesses so that they did not occur during the execution of code that you want to protect.

A SLV access that is stalled by **SLVSTALL** completes when **SLVSTALL** is deasserted. How the access completes is then determined by **SLVPROT** as described above.

4.5.3 HPROT and IOPRIV

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

4.5.4 HMASTER and IOMASTER

The **HMASTER** signal on the AHB-Lite master and **IOMASTER** signal on the IO 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 wish to prevent SLV or processor accesses to some or all memories and peripherals.

4.6 CoreSight

The Cortex-M0+ processor, Cortex-M0+ CTI, and CoreSight MTB-M0+ are CoreSight compliant devices. Read the following sections if you intend building a CoreSight compliant debug system based on these components.

4.6.1 CoreSight ROM tables

The Cortex-M0+ 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-M0+ 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 *CoreSight Architecture Specification* for more information. The Cortex-M0+ internal ROM table provides pointers to the memory-mapped NVIC, MPU and debug resources. See the *Cortex-M0+ Technical Reference Manual* for more information about the Cortex-M0+ ROM table and its functionality. This ROM table is fixed in the processor memory map at address `0xE00FF000` and is not modifiable.

If you build a system that includes additional CoreSight compliant debug components, such as the CoreSight MTB-M0+ or Cortex-M0+ CTI, you must include one, or more, additional ROM tables within your system to enable a debugger to locate those additional components. ARM recommends that you include a system level ROM table that includes your own JEP106 ID and part number values, to enable debuggers to identify your system.

The Integration Kit includes an example system level ROM table located in the memory map at address `0xF0100000`, and a test to check correct ROM table structure in the system. See Chapter 8 *Integration Kit* for more information:

- If you are integrating the `CM0PINTEGRATION` or the `CM0PMTBINTEGRATION` level, the Cortex-M0+ DAP is preconfigured to point to the processor internal ROM table at `0xE00FF000`. To change this, you must set the `BASEADDR` parameter to the value: `{ADDRESS[31:12], 12'h003}` where `ADDRESS` is the 4KB aligned base address of your system level ROM table and `BASEADDR` bits [1:0] indicate that the pointer is present and in 32-bit format. In the Integration Kit example, `BASEADDR` is set to `0xF0100003` to point to the system level ROM table instantiated outside of the integration level. The IK system level ROM table contains entries that point to the Cortex-M0+ processor and, optionally, the CoreSight MTB-M0+.
- If you are integrating the `CORTEXM0PLUS` level, see the *ARM CoreSight Components Technical Reference Manual* for information about how to configure your ROM tables to ensure that the Cortex-M0+ ROM table is appropriately referenced by your Debug Port or by your own system level ROM table.
- The `CORTEXM0PLUSINTEGRATIONCS` level includes a ROM table at address `0xF0000000` that contains pointers to the Cortex-M0+ processor, Cortex-M0+ CTI and optional CoreSight MTB-M0+, and identifies this combination of components as a standard ARM subsystem. If you are integrating the `CORTEXM0PLUSINTEGRATIONCS` level, see the *ARM CoreSight Components Technical Reference Manual* for information about how to configure your ROM tables to ensure that the integration level ROM table is appropriately referenced by your DP or by your own system level ROM table.

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 whilst still allowing debug slave port. See Table 4-19 on page 4-21.

Chapter 5
Key Implementation Points

This chapter describes the key implementation points you must consider when you implement the Cortex-M0+ processor. It contains the following sections:

Note

This chapter references steps 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-M0+ processor. Read this chapter in conjunction 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 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 integration level, `CM0PINTEGRATION`. b. The integration level, `CM0PMTBINTEGRATION`. c. The integration level, `CORTEXM0PLUSINTEGRATIONCS`. d. The processor component level, `CORTEXM0PLUS`. e. A higher level in your SoC that includes the Cortex-M0+ 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 E *Power Intent*.
5.	If you require SRPG, select appropriate UPF or CPF file and library cells for power gating.	See Appendix E *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 Integration Kit tests on the netlist with SDF annotation.	See *Running RunIK with the -netlist option* on page 8-19.
15.	Perform sign-off in accordance with the agreed criteria and your sign-off obligations.	See Chapter 10 *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 10-4.

5.3 Other considerations for implementation

This section describes points that you must consider when implementing your design that are not covered by the configuration options described in Chapter 2 *Configuration Guidelines*.

5.3.1 Architectural clock gating

The Cortex-M0+ 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 `cm0p_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 `cm0p_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 `cm0p_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/cm0p_acg.v` file contains a configurable module that you can use for simulation and logical equivalence checking. Do not use this version for synthesis.

5.3.2 Special purpose cells

The Cortex-M0+ processor and its associated example system components have a number of 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 a number of 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 *Cortex-M0+ Reference Methodology Release Note* for the location of these files.

For each individual cell, there is a reference version 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-M0+ processor and its associated example system components.

Table 5-2 Cortex-M0+ special purpose cells

Component	Cells
CM0PDAP[a]	cm0p_dap_cdc_capt_sync.v
	cm0p_dap_cdc_comb_and.v
	cm0p_dap_cdc_comb_and_addr.v
	cm0p_dap_cdc_comb_and_data.v
	cm0p_dap_cdc_send.v
	cm0p_dap_cdc_send_addr.v
	cm0p_dap_cdc_send_data.v
	cm0p_dap_cdc_send_reset.v
	cm0p_dap_jt_cdc_comb_and.v
	cm0p_dap_sw_cdc_capt_reset.v
	cm0p_dap_sw_cdc_capt_sync.v
CM0PINTEGRATION	cm0p_dbg_reset_sync.v[b]
System components	cm0p_rst_send_set.v
	cm0p_rst_sync.v
	cm0p_pmu_acg.v
	cm0p_pmu_cdc_send_reset.v
	cm0p_pmu_cdc_send_set.v
	cm0p_pmu_sync_reset.v
	cm0p_pmu_sync_set.v
CORTEXM0PLUS CM0PMTBINTEGRATION	cm0p_acg.v

a. This is included if you implement CM0PINTEGRATION 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
cm0p_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.
cm0p_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-M0+ DAP with multi-drop.
cm0p_dap_cdc_send cm0p_dap_cdc_send_reset cm0p_dap_cdc_send_addr cm0p_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 CM0PINTEGRATION 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.
cm0p_dap_cdc_comb_and cm0p_dap_cdc_comb_and_addr cm0p_dap_cdc_comb_and_data cm0p_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-M0+ DAP is implemented.

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

Module	Description
cm0p_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 cm0p_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-M0+ DAP without multi-drop.
cm0p_dap_sw_cdc_capt_sync	Logically, this modules 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 cm0p_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.
cm0p_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 cm0p_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
cm0p_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.
cm0p_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.
cm0p_pmu_sync_set	-	

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

Module	Parameter	Description
cm0p_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 CM0PINTEGRATION 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.
cm0p_pmu_cdc_send_reset	-	
cm0p_pmu_cdc_send_set	-	
cm0p_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 cm0p_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.

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 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 `CM0PINTEGRATION` 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.

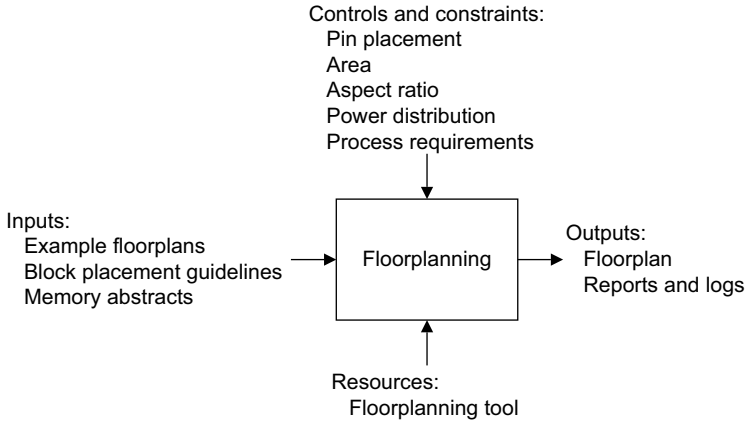


Figure 6-1 Floorplan process

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 together 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.

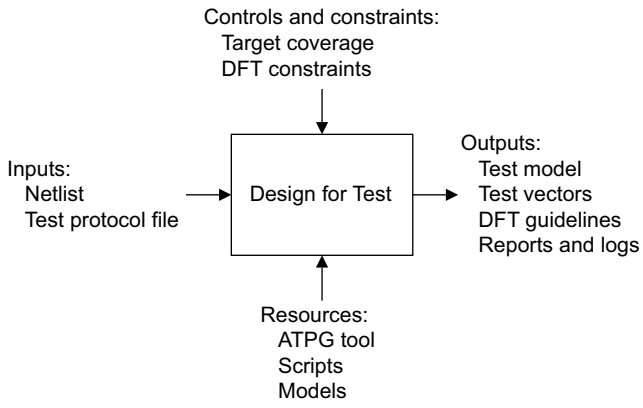


Figure 7-1 Design for Test process

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

Drive **DFTSE** HIGH during scan testing, 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 `CM0PINTEGRATION`, `CM0PMTBINTEGRATION` and `CORTEXM0PLUSINTEGRATIONCS` levels.

The `CORTEXM0PLUS` level does not implement reset synchronizers.

7.4 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 and force enable clock gate cells. 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.

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
Integration Kit

This chapter describes the Cortex-M0+ *Integration Kit* (IK). It contains the following sections:

8.1 About the IK

The Cortex-M0+ Integration Kit (IK) is provided as a reference to enable easy integration of the Cortex-M0+ processor into your system, and contains tests to check that you have performed this integration correctly.

The IK instantiates the supplied `CM0PMTBINTEGRATION` level of hierarchy. The IK tests work with all valid configuration options of the Cortex-M0+ processor, DAP and optional CoreSight MTB-M0+. The IK RTL supports most of these configuration options, and you can modify it to suit your requirements, see *Modifying the IK RTL for your SoC* on page 8-23, *Modifying IK tests* on page 8-30 and *IK limitations* for more information.

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

The tests supplied with the IK 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 IK is based on a simple example microcontroller, `CM0PIKMCU`, that instantiates the `CM0PMTBINTEGRATION` 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-M0+ 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 IK, in which the example microcontroller `CM0PIKMCU` is the device under test. Figure 8-3 on page 8-23 shows a more detailed block diagram of the `CM0PIKMCU` and the IK testbench. Figure D-1 on page D-3 shows a more detailed block diagram of the debug driver, `cm0p_ik_debug_driver`, that provides Serial Wire or JTAG stimulus in the IK testbench.

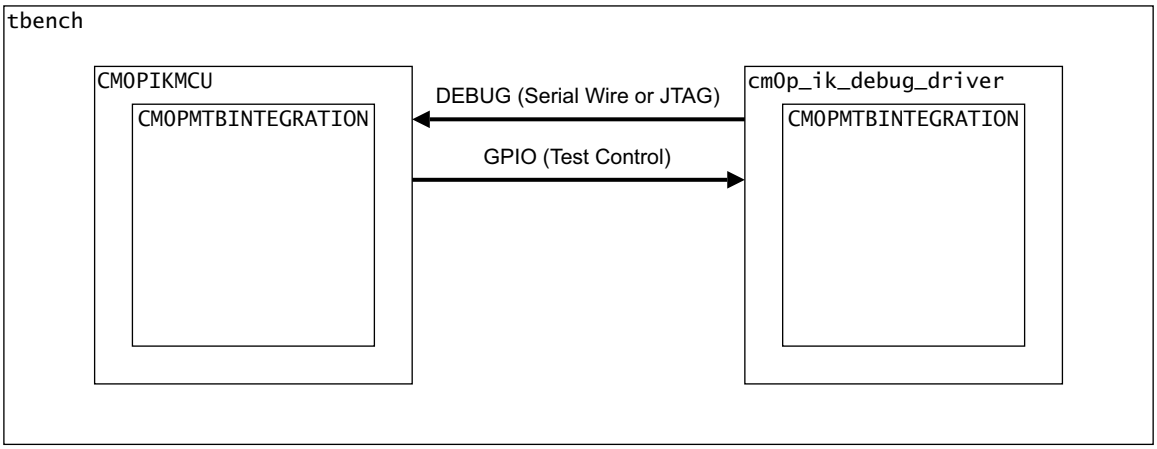


Figure 8-1 Cortex-M0+ IK overview

8.1.1 IK limitations

The Cortex-M0+ IK is designed to be a simple, representative example of a basic Microcontroller. As such, the IK has some known limitations and is not a replacement for other testing. See Chapter 10 *Sign-off* for details of your Sign-Off obligations.

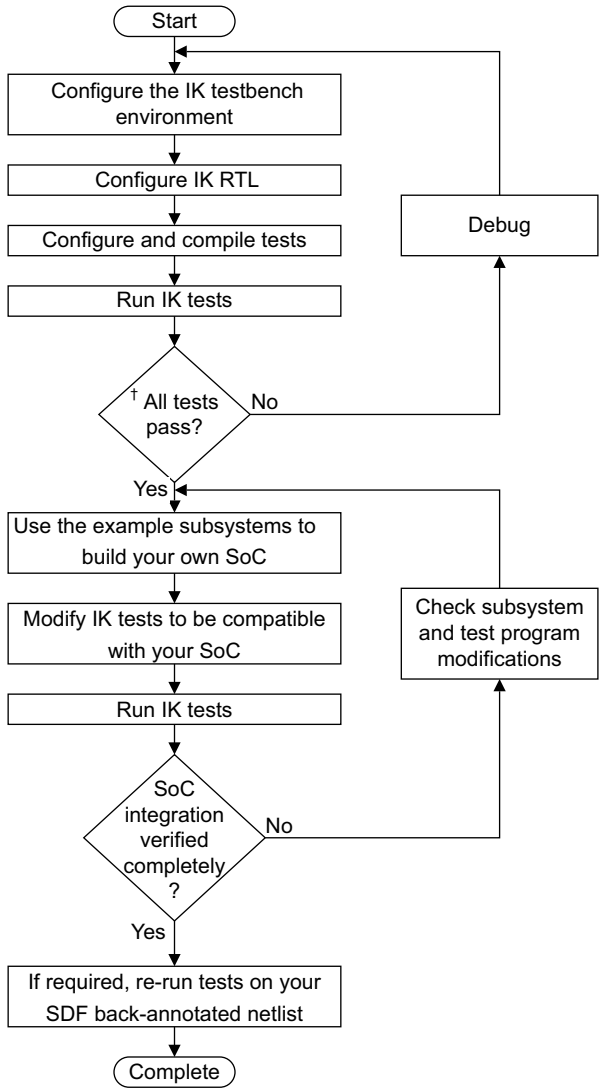
The IK tests do not exhaustively test the Cortex-M0+ processor I/O because not all I/O pins have an effect that is visible to program code, and some pins are tied to static values within the `CM0PIKMCU` example system. This means that you must not use the passing of all IK tests as the only sign-off criteria for successful processor integration.

IK limitations include the following:

- Some implementation parameters do not have a program-visible effect, so cannot be tested in the IK. For example ACG, RAR, AHBSLV, HWF.
- Some tests are timing sensitive to test the effect of implementation parameters such as SMUL, WIC, WICLINES.
- Some tests require the presence of at least one external interrupt pin.
- Some pins are tied-off to static values because the CM0PIKMCU is designed to be a simple example, so are not tested. For example **IRQLATENCY**, **STCALIB**. For simplicity, the Debug Authentication signals **DBGEN** and **NIDEN** are also tied off in CM0PIKMCU.
- The IK is designed to test a single processor subsystem such as a microcontroller. The IK does not support testing at the CORTEXM0PLUSINTEGRATIONCS level of hierarchy.

8.2 Cortex-M0+ IK flow

The Cortex-M0+ IK is intended as a platform from which you can develop a SoC incorporating the Cortex-M0+ processor. Figure 8-2 shows the Cortex-M0+ IK flow.



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

Figure 8-2 Cortex-M0+ IK flow

8.3 Test overview

Table 8-1 shows the IK test programs in the `integration_kit/validation/tests` directory.

Table 8-1 IK 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**, **DEBUGRESTART**, and **DEBUGRESTARTED**. **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 `Makefile`, edit the value of ITERATIONS there. • If you build tests using MDK-ARM™, right-click **CM0PIKMCU - Cortex-M0+** within the dhry project to open the target options, then edit the value of ITERATIONS under the **C/C++** tab.
`ext_debug_check`	This test reads and checks the CPU 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.
`hellow`	The processor reads and checks its CPU 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-M0+ processor and maximize power consumption. Run this test on your netlist to get power values.
`mpu`	This test sets up a number of 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.
`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.

Table 8-1 IK test programs (continued)

Test program	Description
`romtable`	This test checks that it is possible for a debugger to autodetect the Cortex-M0+ processor. The test uses the DAP to locate the architecturally-defined ROM table to locate the processor. If you have included system level ROM tables, the content of these is displayed, but not checked. **Note** • 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.
`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.

8.4 Configuring the testbench

This section describes how to configure the testbench.

The testbench instantiates the `CM0PIKMCU` level of the IK. See *Testbench structure* on page 8-23 for more information. The testbench supports simulation of one of:

- The Verilog RTL source.
- A netlist.
- A *Design Simulation Model* (DSM).

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

Table 8-2 shows the Verilog command files in the `integration_kit/logical/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.
`ovl.vc`	Used for OVL assertions. Defines the path to your Accellera OVL library installation. Modify this file to enable assertions in your simulations.
`rtl.vc`	Used for RTL simulations. Defines the paths to the processor Verilog files. Modify this file to include the CoreSight MTB-M0+ in your simulations.
`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`	Used for DSM simulations. Defines the path to your DSM model and any required Verilog source files. Modify this file to simulate a DSM model.

The Cortex-M0+ processor RTL and Cortex-M0+ DAP RTL include conditionally instantiated *Open Verification Library* (OVL) assertion checkers. These dynamic checks can help you to diagnose problems with your system or test code. The IK testbench also includes conditionally instantiated OVL based protocol checker modules for the debug slave, IO and AHB-Lite interfaces. To run simulations with these checkers and protocol checkers enabled, you must uncomment the appropriate lines in the `logical/tbench/verilog/ovl.vc` file and ensure the path to the OVL library is correct for your environment. You can download the OVL library from `http://www.accellera.org`.

Note

The debug slave protocol checker is supplied in `integration_kit/logical/tbench/verilog/cm0p_slvpc.v`.

The IO port protocol checker is supplied in `integration_kit/logical/tbench/verilog/cm0p_ioppc.v`.

You can obtain the AMBA3 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_CM0PIK_SDF`, in the file `integration_kit/logical/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 want do not want SDF annotation from netlist simulation.
- The Verilog define `ARM_CM0PIK_NETLIST_IODELAY`, in the file `integration_kit/logical/tbench/verilog/tbench.v` points to the appropriate port delay SDF file. You can find example port delay SDF files in `integration_kit/validation/sdf`.
- The Verilog define `ARM_CM0PIK_NETLIST_SDF` points to the SDF file from synthesis flow.
- The Verilog define `ARM_CM0PIK_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 3 ns for the default 50MHz clock. The IK provides three port delay SDF files for use with `CM0PIMP`, `CM0PINTEGRATION` and `CM0PMTBINTEGRATION` levels. They can be found in `integration_kit/validation/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 IK.

8.4.2 State retention power gating considerations

To run UPF or CPF RTL simulations you must define `ARM_SRPG_ON` in `integration_kit/logical/tbench/verilog/rtl.vc`.

To run SRPG-netlist simulations you must define `ARM_SRPG_ON` in `integration_kit/logical/tbench/verilog/netlist.vc`

You must ensure that `ARM_SRPG_ON` is not defined if you do not use UPF or CPF in RTL simulations or SRPG in netlist simulations.

8.5 Configuring the IK RTL

This section describes how you can configure the IK RTL.

8.5.1 Configuring the CM0PMTBINTEGRATION level

The IK instantiates the CM0PMTBINTEGRATION level of hierarchy in both CM0PIKMCU, 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-M0+ processor.

To configure the CM0PMTBINTEGRATION instance in CM0PIKMCU, you must edit the instantiation within the file integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_ik_sys.v.

To configure the CM0PMTBINTEGRATION level, see *CM0PMTBINTEGRATION configuration options* on page 2-9 for more information.

8.5.2 Configuring the IK

The IK uses Verilog defines to control the generation of logic outside of the CM0PMTBINTEGRATION level and to capture system specific values for parameters.

Table 8-3 lists the defines in integration_kit/logical/tbench/verilog/cm0p_ik_defs.v. You must modify the values to match your configuration of the CM0PMTBINTEGRATION level.

Table 8-3 IK defines

Define	Description
ARM_CM0PIK_BASEADDR	Specifies the location of the ROM table base address for the Cortex-M0+ DAP. Changing this define does not modify the location of the system level ROM table in the system. You must only modify this define if you are also modifying the CM0PIKMCU memory map.
ARM_CM0PIK_BE	Specifies the endianness of the Cortex-M0+ processor in CM0PIKMCU and in the debug driver.
ARM_CM0PIK_MTB	Specifies whether the IK supports the CoreSight MTB-M0+. Set this define to 1 if you have licensed the CoreSight MTB-CM0+ and are including it in the system.
ARM_CM0PIK_IOP	Specifies whether the IK supports Cortex-M0+ processor's I/O Port. Set this define to 0 to instantiate an AHB GPIO. Set this define to 1 to instantiate an IOP GPIO. The GPIO is instantiated in both CM0PIKMCU and the debug driver.
ARM_CM0PIK_MTBAWIDTH	Specifies the address width of the CoreSight MTB-M0+ RAM. This define is used to instantiate and connect the SRAM to the MTB, if present. This define does not influence the AHB Decoder, that always decodes 4kB of the memory map to the MTB RAM region. You must modify the AHB Decoder to use an AWIDTH value of more than 12, that is 4kB.

Note

The IK requires these parameters to be set appropriately so that the GPIO, CoreSight MTB-M0+ SRAM and the Cortex-M0+ DAP are correctly configured to support the CM0PMTBINTEGRATION level. The instantiation of CM0PMTBINTEGRATION in integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_ik_sys.v uses these defines. They are also used elsewhere in the IK to configure other IK logic.

If you build a system that includes debug, and you include a CoreSight system level ROM table, you must pass your own JEP-106 ID, part number and revision values into the instantiation of `cm0p_ik_ahb_cs_rom_rom_table`. See *CoreSight ROM tables* on page 4-34. Edit file `integration_kit/logical/cm0p_ik_mcu/logical/verilog/cm0p_ik_sys.v` to modify the instantiation.

8.5.3 Setting the implementation pin options

The IK uses Verilog defines to capture system specific values for implementation pin options. Table 8-4 lists the defines in `integration_kit/logical/tbench/verilog/cm0p_ik_defs.v`. You must modify the values to match your configuration of the `CM0PMTBINTEGRATION` level.

Table 8-4 Implementation pin defines

Define	Description
`ARM_CM0PIK_STCALIB`	SysTick calibration, see Figure 4-7 on page 4-25 for an asynchronous SysTick clock example.
`ARM_CM0PIK_IRQLATENCY`	Interrupt Latency. See *Miscellaneous signals* on page 4-21.
`ARM_CM0PIK_INSTANCEID`	Instance ID for the DAP, see *Miscellaneous DAP signals* on page 4-18.

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-M0+ processor.

The tests have been developed and tested using ARM Compiler 5 running under Linux, and using Keil™ MDK-ARM under Windows. You can download an evaluation version of the:

- ARM Compiler 5, from `http://www.arm.com`.
- Keil MDK-ARM from `http://www.keil.com`.

Note

See the *Cortex-M0+ 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 IK test configuration

The IK 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 a number of `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 `IKConfig.h`.

IK Processor configuration options

Table 8-5 shows the IK processor configuration options.

Table 8-5 IK processor configuration options

Name	Description
`EXPECTED_BE`	Expected value of BE parameter. See Table 2-1 on page 2-4. **Note** • If the Cortex-M0+ processor is in big-endian configuration, you must also compile the IK tests as big-endian. • If you are compiling tests using: — ARM Compiler and the Makefile, edit the `COMPILE_BE` variable — 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_DBG`	Expected value of `DBG` parameter. See Table 2-1 on page 2-4.

Table 8-5 IK processor configuration options (continued)

Name	Description
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	Expected value of MPU parameter. See Table 2-1 on page 2-4. This applies to the Cortex-M0+ processor and the Cortex-M0+ DAP.
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_USER	Expected value of USER parameter. See Table 2-1 on page 2-4. This applies to the Cortex-M0+ processor, the Cortex-M0+ DAP and the CoreSight MTB-M0+.
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 2-1 on page 2-4.
EXPECTED_WPT	Expected value of WPT parameter. See Table 2-1 on page 2-4.

IK processor tie-off options

Table 8-6 shows the IK processor tie-off options.

Table 8-6 IK processor tie-off options

Name	Description
EXPECTED_STCALIB	The expected value of **STCALIB[25:0]**. Ensure this configuration option matches the tied-off value of the corresponding signal. See *SysTick signals* on page 4-24 for information about how to determine this value for your design.

IK DAP configuration options

Table 8-7 shows the IK DAP configuration options.

Table 8-7 IK DAP configuration options

Name	Description
EXPECTED_BASEADDR	The expected value of the CM0PDAP CoreSight Component pointer. Ensure this configuration option matches the tied-off value of the corresponding signal. See *CoreSight ROM tables* on page 4-34 for information about how to determine this value for your design.
EXPECTED_HALTEV	Expected value of HALTEV parameter. See Table 2-2 on page 2-6.
EXPECTED_JTAGnSW	Expected value of JTAGnSW parameter. See Table 2-2 on page 2-6.
EXPECTED_SWMD	Expected value of SWMD parameter. See Table 2-2 on page 2-6.

Table 8-7 IK DAP configuration options (continued)

Name	Description
EXPECTED_TREVISION	Expected value of the TREVISION field of the TARGETID parameter. See Table 2-2 on page 2-6.
EXPECTED_TPARTNO	Expected value of the TPARTNO field of the TARGETID parameter. See Table 2-2 on page 2-6.
EXPECTED_TDESIGNER	Expected value of the TDESIGNER field of the TARGETID parameter. See Table 2-2 on page 2-6.

IK DAP tie-off options

Table 8-8 shows the IK DAP tie-off table options.

Table 8-8 IK DAP tie-off options

Name	Description
EXPECTED_INSTANCEID	Expected value of INSTANCEID[3:0]. Ensure this configuration option matches the tied-off value of the corresponding signal. See *Miscellaneous DAP signals* on page 4-18.

IK CoreSight MTB-M0+ configuration options

Table 8-9 shows the IK CoreSight MTB-M0+ configuration options. If you have not licensed the CoreSight MTB-CM0+, 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 IK CoreSight MTB-M0+ configuration options

Name	Description
EXPECTED_MTB	Expected value of the MTB parameter. See Table 2-3 on page 2-9.
EXPECTED_MTB_AWIDTH	Expected value of the CoreSight MTB-M0+ AWIDTH parameter. See Table 2-3 on page 2-9.
EXPECTED_MTB_BASEADDR	Expected base address of the CoreSight MTB-M0+ special function registers. **Note** This configuration value does not correspond to a Verilog parameter. However, the IK tests require its value. The AHB decoder logic in your system determines its value. In the IK, the value is 0xF0200000.

IK system ROM table

Table 8-10 shows the IK System ROM table configuration options.

Table 8-10 IK system ROM table configuration options

Name	Description
EXPECTED_CUST_JEP_ID	Expected value of cm0p_ik_ahb_cs_rom_table JEPID parameter. This value is your own JEDEC JEP-106 identity code value. See *CoreSight ROM tables* on page 4-34.
EXPECTED_CUST_JEP_CONT	Expected value of cm0p_ik_ahb_cs_rom_table JEPCONTINUATION parameter. This value is your own JEDEC JEP-106 continuation code value. See *CoreSight ROM tables* on page 4-34.
EXPECTED_CUST_PART	Expected value of cm0p_ik_ahb_cs_rom_table PARTNUMBER parameter. This value is the part number value that enables you to identify your system. See *CoreSight ROM tables* on page 4-34.
EXPECTED_CUST_REV	Expected value of cm0p_ik_ahb_cs_rom_table REVISION parameter. This value identifies the revision of your system, as defined by your part number value. See *CoreSight ROM tables* on page 4-34.

IK system ROM table tie-off options

Table 8-12 shows the IK System ROM table tie-off options.

Table 8-11 IK system ROM table tie-off options

Name	Description
EXPECTED_CUST_REVAND	Expected value of cm0p_ik_ahb_cs_rom_table ECOREVNUM[3:0]. Ensure that this configuration option matches the tied-off value of the corresponding signal. See *CoreSight ROM tables* on page 4-34.

IK ECO and revision tie-off options

Table 8-12 shows the IK processor tie-off options.

Table 8-12 IK ECO and revision tie-off options

Name	Description
EXPECTED_ECOREVNUM	Expected value of ECOREVNUM[31:0]. See Table 4-19 on page 4-21.

8.6.2 Test program compilation using the ARM 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 Makefile in the integration_kit/validation/tests directory to suit your environment, for example:

- The name of the ARM compiler.
- The name of the linker.
- Compiler and linker options.
- Test configuration options.

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

- Binary .elf files for use with a debugger.
- Binary .bin files for memory initialization.
- ASCII .inc files that other C program files might include.
- ASCII .rcf (ARM ROM Code) files for use with ARM ROM models.

To compile a test:

1. Change to the integration_kit/validation/tests directory by typing:

   ```
   cd validation/tests
   ```

2. Use the Makefile in integration_kit/validation/tests to compile the test programs. The command can be either:

   `make <test_program>` For individual tests.

   `make all` For all tests.

   where:

<test_program>	Selects an individual test program to compile. See Table 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 hellow.c, type:

   ```
   make hellow
   ```

   Note

   - ARM 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
     ```

8.6.3 Test program compilation using Keil MDK-ARM

This section describes how to compile the test programs using the Keil MDK-ARM tools.

The integration_kit/validation/mdk directory contains a multi-project workspace that includes project files for each of the IK tests and the debug driver. Use the following procedure to compile the test programs:

1. Open the project workspace:

   ```
   Project -> Open Project -> IntegrationKit.uvmpw
   ```

2. Configure the tests:

   Locate the IKConfig.h header file within any of the projects and make any necessary changes.

3. Build the test binaries:

   ```
   Project -> Batch Build -> Select All, Build
   ```

The test creates intermediate build files in the mdk directory. Test programs compile into the tests directory. The project files compile:

- Binary .axf files for use with a debugger.
- Binary .bin files for memory initialization.
- ASCII .inc files that might be included in other C program files.

Note

The Keil MDK-ARM text compilation flow does not generate .rcf files.

8.7 Running IK tests

The testbench is run from the integration_kit/validation directory. The RunIK script is provided as an example script to compile the testbench and simulate a specified test. This section contains:

- *RunIK script usage and options*.
- *Example RunIK output* on page 8-18.
- *Running RunIK with the -dsm option* on page 8-18.
- *Running RunIK with the -netlist option* on page 8-19.
- *Simulation logs* on page 8-20.
- *Modifying the RunIK script* on page 8-20.
- *Running RunIK with the -upf or -cpf options* on page 8-19.

8.7.1 RunIK script usage and options

To run the RunIK script, use the following format:

```
RunIK <options> <testname>
```

where <options> are:

-build	This switch is optional. Use it to compile the IK Verilog, including the testbench, before running the test or tests.
-make	This switch is optional. Use this switch to call make to compile the tests using the ARM Compiler tools, if these tools are present. If you do not have the ARM Compiler tools, compile the tests before you run this script.
-all	Use this switch to run all IK tests.
-dsm	Use a DSM for the CORTEXM0PLUS level.
-h[elp]	Use this switch to display RunIK usage and options.
-netlist	Use this switch to run the IK with a netlist.
-mti	Use the MTI simulator. The default is MTI.
-nc	Use the NC simulator.
-vcs	Use the VCS simulator.
-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.
-upf	Use UPF for simulation.
-cpf	Use CPF for simulation.
-64	Use 64-bit simulation mode.

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

```
RunIK -vcs -build hellow
```

The RunIK script creates a directory within integration_kit/validation, corresponding to the specified simulator:

- MTI.

- VCS.
- NC.

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

Example RunIK output

The RunIK script prints a summary at the end unless it is used with the -noexec switch. This is also available in file integration_kit/validation/logs/summary.txt.

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

```
Processor - Cortex-M0+
Simulator - MTI
64-bit    - FALSE
Source    - RTL

------------------------------------------------------------
                 Summary of RunIK
------------------------------------------------------------
hellow           |        PASS |           - |
config_check     |        PASS |           - |
dhry             |        PASS |           - |
max_power        |        PASS |           - |
interrupt        |        PASS |           - |
reset            |        PASS |           - |
sleep            |        PASS |           - |
debug            |        PASS |           - |
romtable         |        FAIL |           - |
mpu              |        PASS |        SKIP |
vtor             |        PASS |        SKIP |
ppb_iop_check    |        PASS |        SKIP |
ext_debug_check  |        PASS |           - |
------------------------------------------------------------
TOTAL PASSES :: 12
TOTAL FAILURES :: 1
TOTAL OVL FAILURES :: 0
TOTAL KILLED :: 0
TOTAL TESTS :: 13
------------------------------------------------------------
```

If OVL assertions are enabled, an OVL_FATAL or OVL_ERROR in the test causes RunIK to report the test as a FAIL. OVL_WARNINGs, in addition to test WARNINGs, are reported in the fourth summary column.

Note

In the example, the romtable test reports a FAIL because the values of the system ROM table parameters are not valid. Three tests report SKIP because the default configuration of the processor does not include the associated features.

8.7.2 Running RunIK with the -dsm option

Ensure that the DSM is set up correctly to run RunIK with the -dsm option. See the documentation in the DSM package for more information about DSM setup. An example file integration_kit/validation/dsmdotcshrc is provided to set up the DSM. Source the dsmdotcshrc to set the environment variables required for DSM simulation. In addition, ensure that the file integration_kit/logical/tbench/verilog/dsm.vc points to the CORTEXM0PLUS DSM.

8.7.3 Running RunIK with the -netlist option

To run RunIK with the -netlist option, ensure that:

- The file integration_kit/logical/tbench/verilog/netlist.vc points to the netlist at the required level.
- The instantiation of the netlist in the IK does not take parameters. You have to remove any parameters passed to the netlist instance.

  Note

  The debug driver block also instantiates the netlist.

- The file integration_kit/logical/tbench/verilog/netlist.vc points to the appropriate technology libraries for netlist simulation.
- The rest of the IK 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.

Note

A netlist generated with the processor parameter RAR set to 0 has the potential to propagate X's. This happens when the netlist synthesis is aggressive resulting in reordering of logic to meet the timing constraints. Such a netlist must be fixed for X-propagation, if it has to be simulated successfully in the IK. X's can be removed by either modifying the cell library primitives to appropriately deal with X's or by forcing a value that is not X through the simulator to the signal that is going X.

8.7.4 Running RunIK with the -upf or -cpf options

Before simulating the RTL or netlist with UPF or CPF ensure that:

- SRPG signals are included.
- The RunIK script points to the appropriate UPF or CPF file.

  Note

  The default UPF file is srpg/CM0PIKMCU.upf and the default CPF file is srpg/CM0PIKMCU.cpf.

- The mvtools suite is on your path for UPF simulation.
- Verilog define ARM_SRPG_ON in integration_kit/logical/tbench/verilog/rtl.vc is defined.

Note

The following two Verilog defines might need to be defined in integration_kit/logical/tbench/verilog/rtl.vc to work around tool issues.

- ARM_CPF_PWRDOWNACK if using CPF.
- ARM_UPF_MVTOOLS_VCS if using UPF and VCS.

8.7.5 VCD generation

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

- Verilog define `ARM_CM0PIK_VCD` in `integration_kit/logical/tbench/verilog/tbench.vc` is defined.
- Verilog defines `ARM_CM0PIK_VCD_START` and `ARM_CM0PIK_VCD_STOP` in `integration_kit/logical/tbench/verilog/tbench.vc` are set appropriately.
- Verilog define `ARM_CM0PIK_VCD_FILE` in `integration_kit/logical/tbench/verilog/tbench.v` is defined.

8.7.6 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_CM0PIK_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 nonexistent feature of the Cortex-M0+ processor, it passes but prints the following message in the simulation log:

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

The simulation log files are generated in `integration_kit/validation/logs`.

Note

If you enable OVL assertions, any OVL messages that appear in the test logs do not modify these status messages. The presence of OVL messages in the test logs is reported in the RunIK summary information.

8.7.7 Modifying the RunIK script

You can modify the `RunIK` script to include any extra simulator options you want to use.

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

8.8 Debugging failing tests

This section describes what you can do to help diagnose problems when a test or 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 `hellow`. 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 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.

If you have enabled OVL checks, you might see messages that indicate RTL misconfiguration or X value propagation. The latter is normally the result of accessing uninitialized memory or a malfunctioning memory subsystem. See *Configuring the testbench* on page 8-7 for information on how to enable OVL checks.

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 commenting out the definition of `CM0PIKMCU_PRINTF` in `IKConfig.h` before compiling the tests. You might want to do this to reduce the runtime of the tests.

If a test is failing, enable message printing by defining `CM0PIKMCU_PRINTF`, for example by uncommenting it in `IKConfig.h`, and recompile and rerun the test to help determine the reason for failure.

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

Run on an unmodified version of the IK, or see the golden logs in directory `glogs`, 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.

Compare executed instructions against the test code

By default, RTL simulations produces two log files, `tarmac0.log` and `tarmac1.log`, of instructions executed on the processor in CM0PIKMCU and Debug Driver respectively. These can be used to debug test failures by comparing the executed instructions with the assembly language of the test code. You can use the ARM Compiler `fromelf` utility to show the assembly language content of the compiled test code, for example:

```
fromelf -c hellow.elf
```

Note

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

8.9 Modifying the IK RTL for your SoC

This section describes the testbench structure and the IK level, CM0PIKMCU, memory map. Read this section if you want to modify the IK RTL for your own SoC requirements.

8.9.1 Testbench structure

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

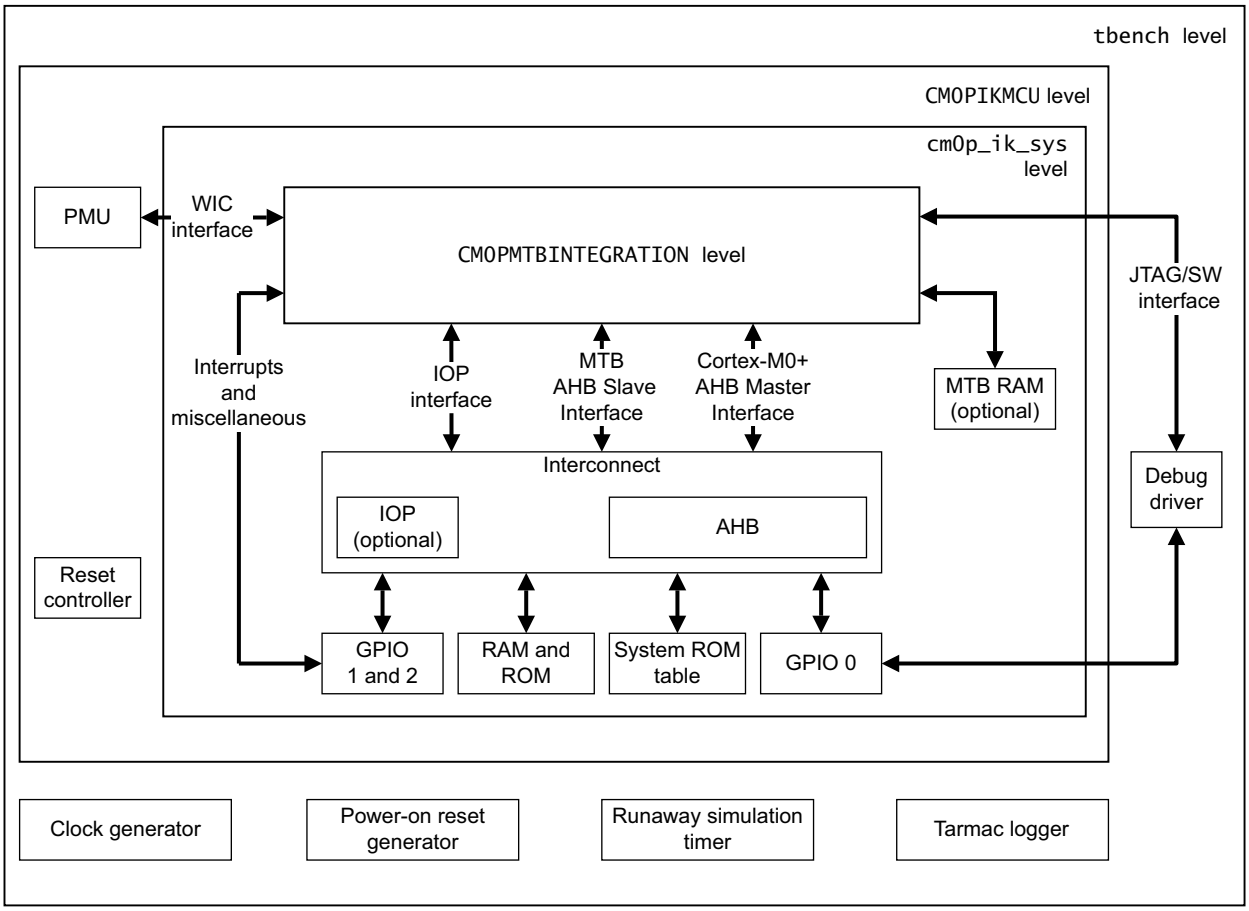


Figure 8-3 Integration kit

The IK contains these levels:

- *CM0PMTBINTEGRATION level*.
- *cm0p_ik_sys level* on page 8-24.
- *CM0PIKMCU level* on page 8-25.
- *tbench level* on page 8-25.

The `integration_kit/` path in Figure 1-7 on page 1-11 shows the supplied IK directory structure.

CM0PMTBINTEGRATION level

This level instantiates the Cortex-M0+ processor, Cortex-M0+ DAP, and the WIC.

cm0p_ik_sys level

This section describes the cm0p_ik_sys level and its components. The cm0p_ik_sys level contains:

- *GPIO* on page 8-27.
- *AHB default slave* on page 8-28.
- *ROM controller* on page 8-27.
- *ROM* on page 8-27.
- *SRAM controller* on page 8-27.
- *SRAM* on page 8-27.
- *AHB bus interconnect* on page 8-27.
- *IOP bus interconnect* on page 8-28.
- MTB SRAM.
- The system level ROM table.
- Miscellaneous logic used for integration test purposes.

If the IK is configured to include the CoreSight MTB-M0+, the MTB SRAM is instantiated in cm0p_ik_sys. See *SRAM* on page 8-27.

The system level ROM table enables a debugger to uniquely identify the Cortex-M0+ based system, and enables debug components, such as the optional CoreSight MTB-M0+, to be discovered automatically. See *CoreSight ROM tables* on page 4-34.

This example system level ROM table has two entries:

1. A pointer to the Cortex-M.0+ ROM table.
2. A pointer to the optional CoreSight MTB-M0+.

You must modify the instantiation of the module to use your own JEP106 manufacturer ID value and a part number that identifies your system. See *IK system ROM table* on page 8-14.

cm0p_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 *CM0PIKMCU GPIO Integration*.

General Purpose Input Output 1

This GPIO drives signals in the example CM0PIKMCU, for test purposes only. It drives PMU signals and other processor signals.

See Appendix B *CM0PIKMCU GPIO Integration*.

General Purpose Input Output 2

This GPIO drives signals in the example CM0PIKMCU, 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 *CM0PIKMCU GPIO Integration*.

CM0PIKMCU level

The CM0PIKMCU level is a simple example MCU and contains:

- *cm0p_ik_sys level* on page 8-24.
- *PMU* on page 8-28.
- *Reset controller* on page 8-29.

tbench level

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

- *CM0PIKMCU level*.
- *Debug driver* on page 8-29.
- Modules specific to `tbench` level:

  Clock generator

  This generates clocks for the `CM0PIKMCU` 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.

  Tarmac logger The tarmac logger creates an output log file of the instructions executed by the Cortex-M0+ processor during a test run. See Appendix H *Tarmac Trace Description* for more information.

8.9.2 CM0PIKMCU memory map

Figure 8-4 on page 8-26 shows the `CM0PIKMCU` memory map.

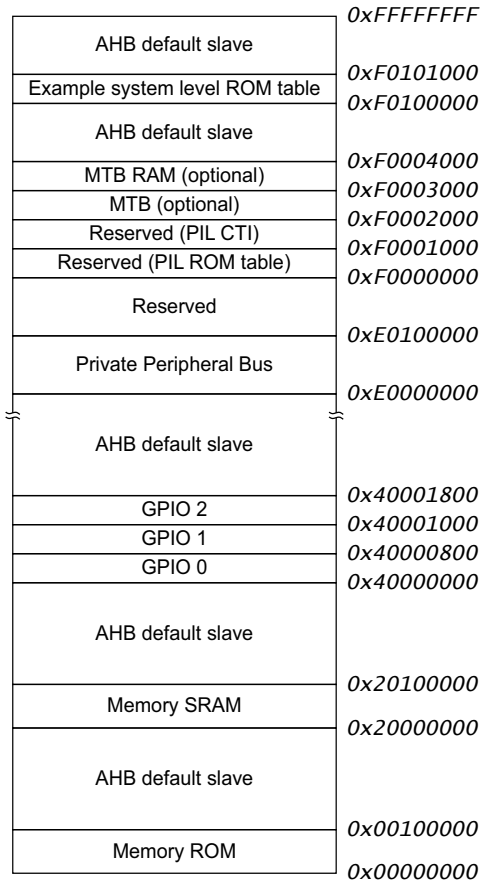


Figure 8-4 CM0PIKMCU memory map

The CM0PIKMCU memory map is characterized by the following:

- CM0PIKMCU instantiates 2MB of physical memory. It is split into two blocks of 1MB:
  - The first 1MB, typically flash in an MCU, is read-only memory that holds the code.
  - The second 1MB, typically SRAM, is where the stack and heap are located.
- There are three GPIOs, each allocated 2KB of space.

  When the IK includes IOP, the GPIO addresses map to IOP GPIO instead of AHB GPIO.
- The *Private Peripheral Bus* (PPB) space of the Cortex-M0+ processor is 1MB.
- The example system level ROM table occupies 4KB of space.
- If the CoreSight MTB-M0+ is included, it occupies 8KB of space:
  - The first 4KB is the CoreSight MTB-M0+ SFR region.
  - The second 4KB is the CoreSight MTB-M0+ SRAM.

  If the CoreSight MTB-M0+ is not included, these regions are mapped to the AHB default slave.
- 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 IK components

This section describes the components supplied with the IK and instantiated by the `CORTEXM0PLUSINTEGRATIONCS` or `cm0p_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, `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_ik_ahb_gpio.v` and `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_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, `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_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, `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_ik_rom.v`, is an example behavioral Verilog model which supports image preloading from a file. The IK test code image is loaded here.

Note

- `cm0p_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, `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_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, `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_ik_sram.v`, is a synchronous SRAM.

8.10.6 AHB bus interconnect

The AHB bus interconnect module, `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_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 CM0PMTBINTEGRATION level.

The seven master ports are connected to the ROM, the RAM, GPIO 0, GPIO 1, GPIO 2, the system ROM table, and the AHB default slave respectively.

8.10.7 IOP bus interconnect

The IOP bus interconnect module, integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_ik_iop_interconnect.v, contains logic for driving **IOMATCH** and **IORDATA** to the processor I/O port. This block exists only if the IK is configured to include the I/O port.

8.10.8 AHB default slave

The AHB default slave, integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_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.

8.10.9 WIC

The Wake up Interrupt Controller module, cm0p_integration/verilog/cm0p_wic.v, is an example module that is external to the Cortex-M0+ 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 CM0PINTEGRATION.v.

The WIC includes two parameters IRQDIS and WICLINES which correspond to the Cortex-M0+ 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 4-26 if you plan to build a WIC that is not clocked by **FCLK**.

8.10.10 PMU

The PMU, integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_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 power-down 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 power-down.

8.10.11 CoreSight ROM table

The system level ROM table, `cm0p_integration/verilog/cm0p_ahb_cs_rom_table.v`, is an example AHB-Lite four-entry CoreSight ROM table which uses parameters to define its content. If you use this component as a system level ROM table you must modify the instantiation of the module to use your own JEP106 manufacturer ID value and a part number that identifies your system.

8.10.12 Debug driver

The Debug driver module, `integration_kit/logical/tbench/verilog/cm0p_ik_debug_driver.v`, is provided to run tests on a Cortex-M0+ based subsystem that is used in the IK to provide Serial Wire or JTAG debug stimulus to `CM0PIKMCU`. See Appendix D *Debug Driver*.

8.10.13 Reset controller

The Reset controller, `integration_kit/logical/cm0p_ik_mcu/verilog/cm0p_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.

8.10.14 Downsizer

The Downsizer module, `integration_kit/logical/tbench/verilog/cm0p_32to16_dnsize.v`, is an example 32bit to 16bit AHB-Lite downsizer.

You can use the downsizer example in applications where the Cortex-M0+ 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-M0+ 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 IK tests

The supplied test programs rely on the IK 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 IK. 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 IK 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 IK includes a minimal subset of the CMSIS for the Cortex-M0+ processor that is sufficient to support the supplied tests. See `http://arm.com/cmsis` for the full version of the CMSIS and `http://onarm.com` 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-M0+ processor based device.

Table 8-13 shows the files in the `integration_kit/validation/tests` and `integration_kit/validation/tests/CMSIS` directories that constitute the CMSIS.

Table 8-13 CMSIS files

Filename	Location	Supplied by	Description
`boot_cm0pikmcu.c`	`tests/Device/ARM/cm0pikmcu/Source/ARM`	ARM	This file provides the stack and heap initialization, vector table and default exception handlers. `boot_cm0pikmcu.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_cm0pikmcu.c`.
`core_cm0plus.h`	`tests/CMSIS/Include`	ARM	Defines the core peripherals for the Cortex-M0+ processor.
`core_cmFunc.h`	`tests/CMSIS/Include`	ARM	Defines the Cortex-M Core Register access functions.
`core_cmInstr.h`	`tests/CMSIS/Include`	ARM	Defines the Cortex-M Core instructions. That is, it defines functions that provide access to instructions that are not part of the C language.
`cm0pikmcu.h`	`tests/Device/ARM/cm0pikmcu/Include`	Device vendor	Device specific file that defines the peripherals for the `CM0PIKMCU` example microcontroller device.
`system_cm0pikmcu.h`	`tests/Device/ARM/cm0pikmcu/Include`	Device vendor	Header file that provides device specific configuration for the `CM0PIKMCU` example microcontroller device.
`system_cm0pikmcu.c`	`tests/Device/ARM/cm0pikmcu/Include`	Device vendor	C file that provides device specific configuration for the `CM0PIKMCU` example microcontroller device.

Table 8-14 shows the test support files in the integration_kit/validation/tests directory.

Table 8-14 Test support files

Filename	Description
IKtests.h	This header file describes and defines the allocation of **GPIO** pins used by the CM0PIKMCU in the IK. It also declares the function prototypes the IK tests use to communicate with the debug driver.
IKtests.c	This file provides the functions the IK 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 IK.
Makefile	This file enables you to build the IK tests using the ARM Compiler toolchain.
retarget_cm0pikmcu.h	This file implements the functions necessary to retarget the C-library printf() function output to the CM0PIKMCU **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 sections:

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 integration kit to perform dynamic verification, by simulating your netlist. See Chapter 8 *Integration Kit* for more information.

Chapter 10
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:

10.1 About sign-off

Figure 10-1 shows the top level inputs, resources, outputs, and controls and constraints for sign-off.

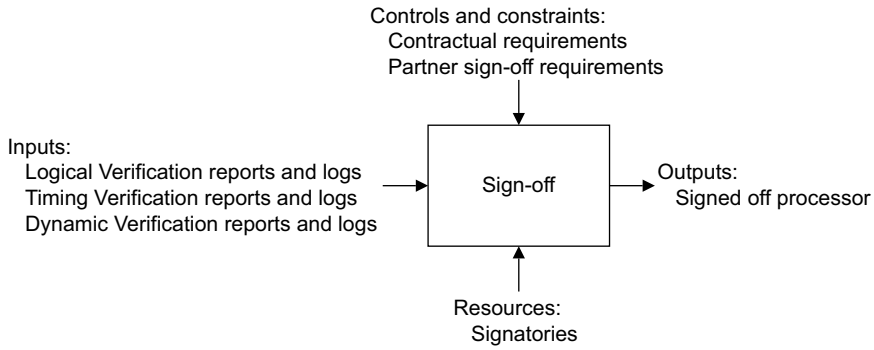


Figure 10-1 Sign-off process

10.2 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 and constraints* on page 1-7 for more information.

10.3 Requirements for sign-off

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

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

10.3.1 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 the following implementation stages successfully for sign-off:

- Logical Equivalence Check.
  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 10-6.

Note

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

10.3.2 Recommended for sign-off

Table 10-1 shows the recommended stages for sign-off.

Table 10-1 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 `CORTEXM0PLUS` or `CM0PINTEGRATION` levels.

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

10.4.1 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 `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.

10.4.2 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.

10.4.3 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 of the supplied validation tests on a netlist with back-annotated timing as a final check.

10.5 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 Cortex-M0+ *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.

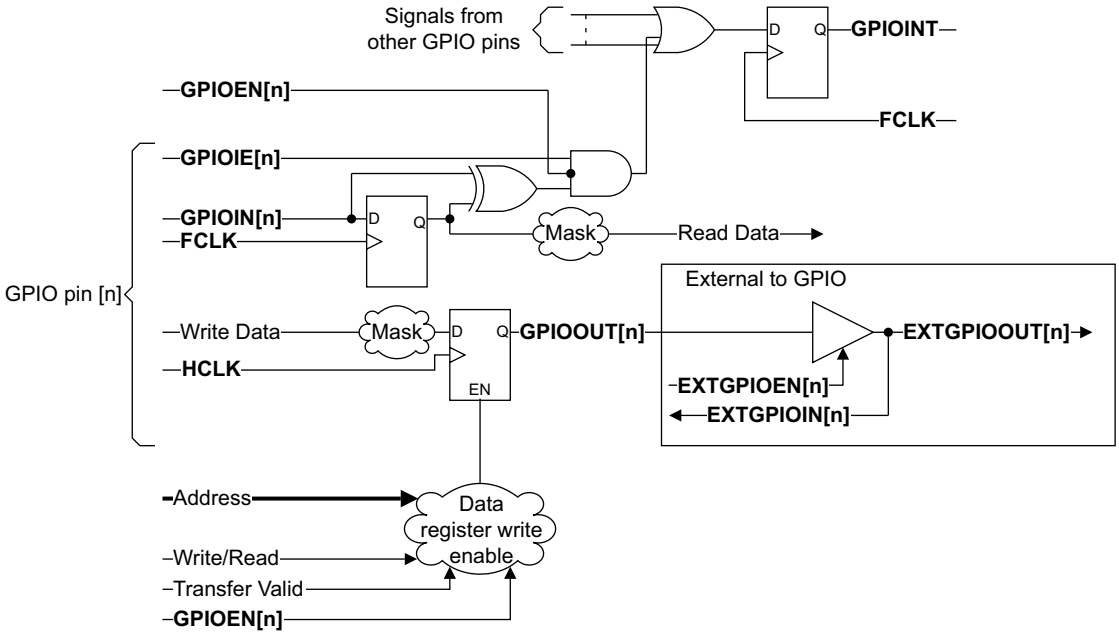


Figure A-1 GPIO Data Register

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
CM0PIKMCU GPIO Integration

This appendix describes the integration of the GPIO within the CM0PIKMCU 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 integration kit. 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-M0+ integration kit 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.

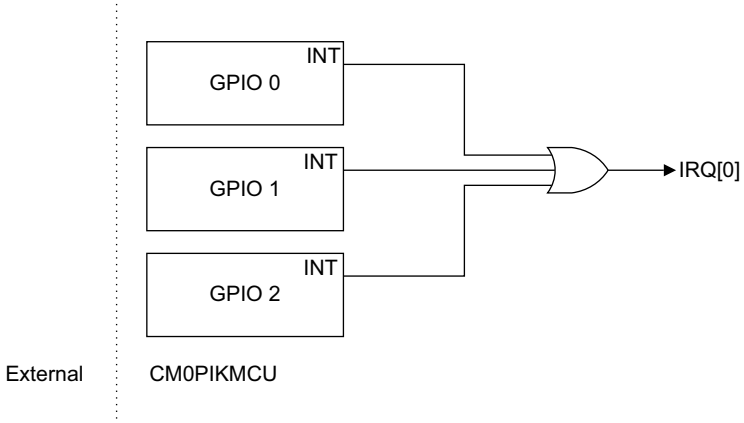


Figure B-1 GPIO interrupt line connections

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 integration kit interrupt tests to work even when the processor is configured to have only one external interrupt.

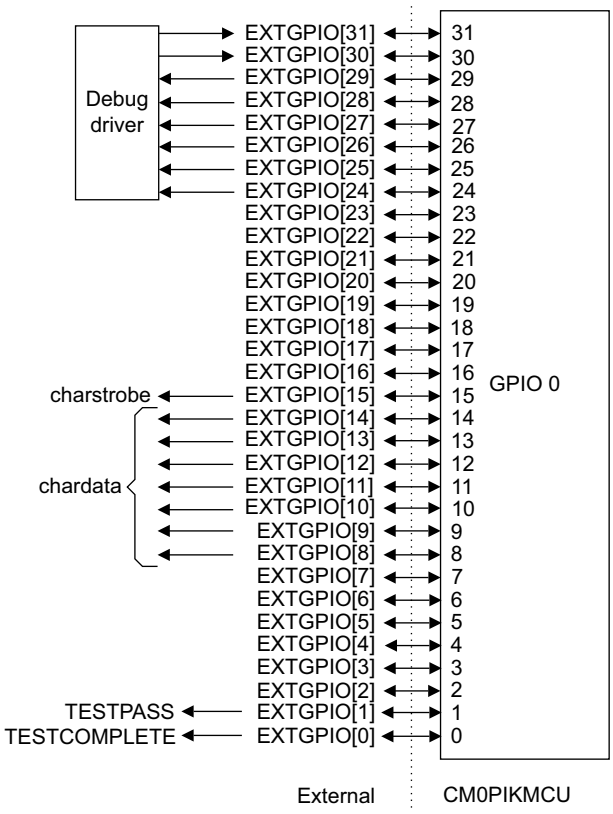


Figure B-2 GPIO 0 connections

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

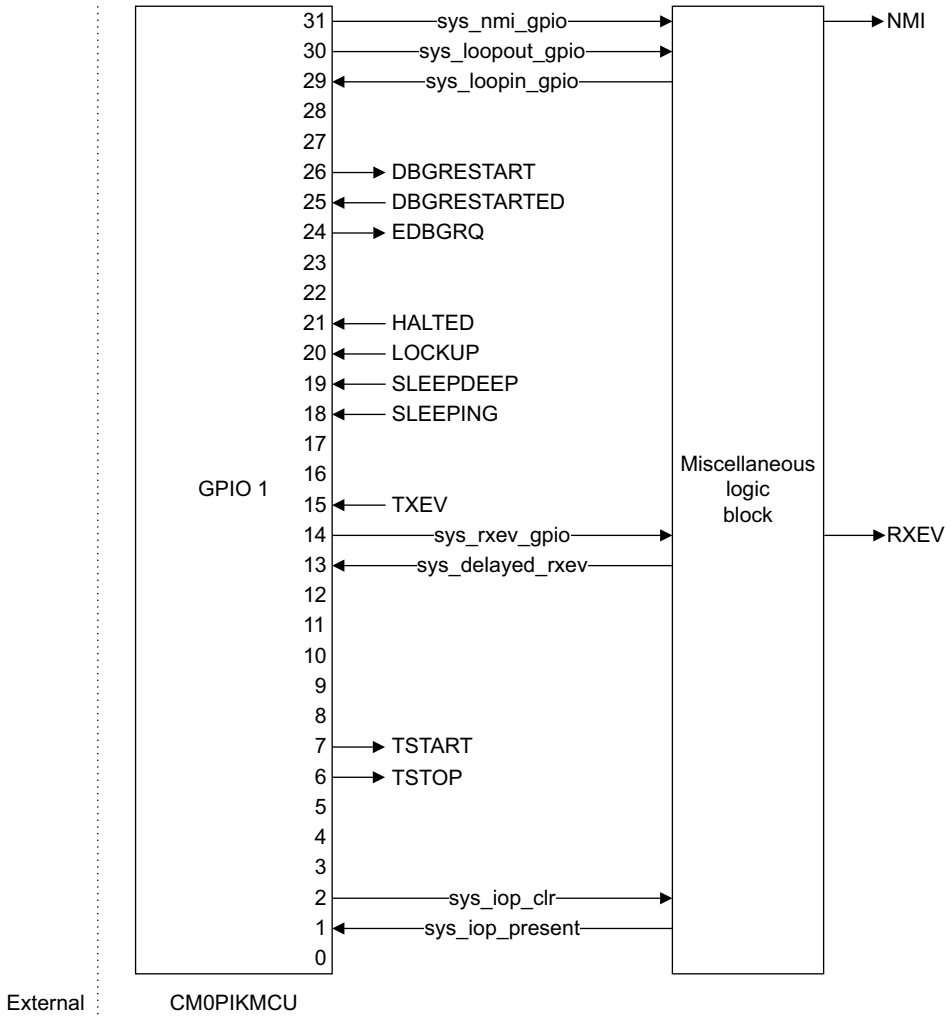


Figure B-3 GPIO 1 connections

Figure B-4 shows the GPIO 2 connections.

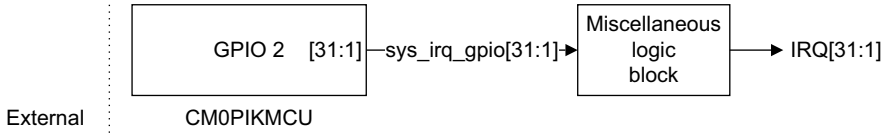


Figure B-4 GPIO 2 connections

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 CM0PIKMCU, with its signals routed to the top level of the CM0PIKMCU.

The **EXTGPIO** signals are used in the integration kit 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-M0+ integration kit.

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 CM0PIKMCU 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-M0+ 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-M0+ processor
[25]	**DBGRESTARTED**	-	Connects to the **DBGRESTARTED** pin of the Cortex-M0+ processor.
[24]	-	**EDBGRQ**	Drives the **EDBGRQ** pin of the Cortex-M0+ processor.
[23:22]	-	-	Not used.
[21]	**HALTED**	-	Connects to the **HALTED** pin of the Cortex-M0+ processor.
[20]	**LOCKUP**	-	Connects to the **LOCKUP** pin of the Cortex-M0+ processor.
[19]	**SLEEPDEEP**	-	Connects to the **SLEEPDEEP** pin of the Cortex-M0+ processor.
[18]	**SLEEPING**	-	Connects to the **SLEEPING** pin of the Cortex-M0+ processor.
[17:16]	-	-	Not used.

Table B-2 GPIO 1 bit assignments (continued)

Bit	Receive input from	Send output to	Connection information
[15]	**TXEV**	-	Connects to the **TXEV** pin of the Cortex-M0+ processor.
[14]	-	**RXEV**	Drives the **RXEV** pin of the Cortex-M0+ processor.
[13]	Miscellaneous logic block	-	Delayed version of the **RXEV** pin of the Cortex-M0+ processor.
[12:8]	-	-	Not used.
[7]	-	**TSTART**	Connects to **TSTART** pin of CoreSight MTB-M0+.
[6]	-	**TSTOP**	Connects to **TSTOP** pin of CoreSight MTB-M0+.
[5:3]	-	-	Not used
[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-M0+ processor.
[0]	-	-	Not used.

B.1.3 GPIO 2 bit assignments

GPIO 2 is used internally to the example CM0PIKMCU to drive **IRQ[31:1]** 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
[31:1]	-	Miscellaneous logic block	Drives the corresponding **IRQ** pin of the Cortex-M0+ processor, after a delay. For example, bit[31] drives **IRQ[31]**.
[0]	-	-	Not used.

Appendix C
SysTick Examples

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

C.1 SysTick examples

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

Example C-1 25MHz SCLK, no reference provided

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

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

Example C-2 14.31818MHz SCLK, no reference provided

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

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

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

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

Example C-4 shows an 32.768kHz asynchronous example.

Example C-4 32.768kHz asynchronous example

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

Appendix D
Debug Driver

This appendix describes the debug driver supplied with the integration kit. It contains the following section:

- *Debug driver* on page D-2.

D.1 Debug driver

The debug driver block is a testbench component that uses a GPIO and software to control the Cortex-M0+ DAP by driving the SW or JTAG pins of CM0PIKMCU. The debug driver contains an instantiation of the CM0PMTBINTEGRATION level, memories, and a GPIO. Tests running on CM0PIKMCU 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 integration kit to test CM0PIKMCU even when its processor is sleeping.

The debug driver always executes the binary file debugdriver.bin, regardless of the test run on CM0PIKMCU. The debugdriver.bin binary must be built from the supplied source code. See *Test program compilation using the ARM Compiler* on page 8-14 and *Test program compilation using Keil MDK-ARM* on page 8-15 for information on how to compile and build the integration kit tests.

Table D-1 lists the software source files used by the debug driver.

Table D-1 Debug driver source files

File	Location	Description
core_cmFunc.h	tests/CMSIS/Include	CMSIS file that defines the Cortex-M Core Register access functions.
core_cmInstr.h	tests/CMSIS/Include	CMSIS file that defines the Cortex-M Core instructions.
core_cm0plus.h	tests/CMSIS/Include	CMSIS file that defines the core peripherals for the Cortex-M0+ processor.
cm0pikdebugdriver.h	tests/Device/ARM/cm0pikdebugdriver/Include	CMSIS device specific file that defines the peripherals for the cm0pikdebugdriver block.
system_cm0pikdebugdriver.h	tests/Device/ARM/cm0pikdebugdriver/Include	CMSIS device vendor Header file that provides device specific configuration for the cm0pikdebugdriver block.
system_cm0pikdebugdriver.c	tests/Device/ARM/cm0pikdebugdriver/Source	CMSIS device vendor C file that provides device specific configuration for the cm0pikdebugdriver block.
boot__cm0pikdebugdriver.c	tests/Device/ARM/cm0pikdebugdriver/Source/ARM	This file provides the stack and heap initialization, vector table, default handlers and _sys_exit function used by cm0pikdebugdriver.
retarget_cm0pikdebugdriver.c	tests	This file implements the functions necessary to retarget the C-library printf() function output to the cm0pikdebugdriver GPIO pins.
debugdriver.h	tests	This header file contains various defines used by the debug driver block, including the GPIO pin allocations.
debugdriver.c	tests	This is the main source code file for the debug driver block. It includes the routines for communicating with CM0PIKMCU through the GPIO and the routines to drive the CM0PIKMCU Serial Wire or JTAG interface.
debugdriver_functions.h	tests	This header file contains a C enum representing the functions that the debug driver makes available to CM0PIKMCU.

Figure D-1 on page D-3 shows the debug driver.

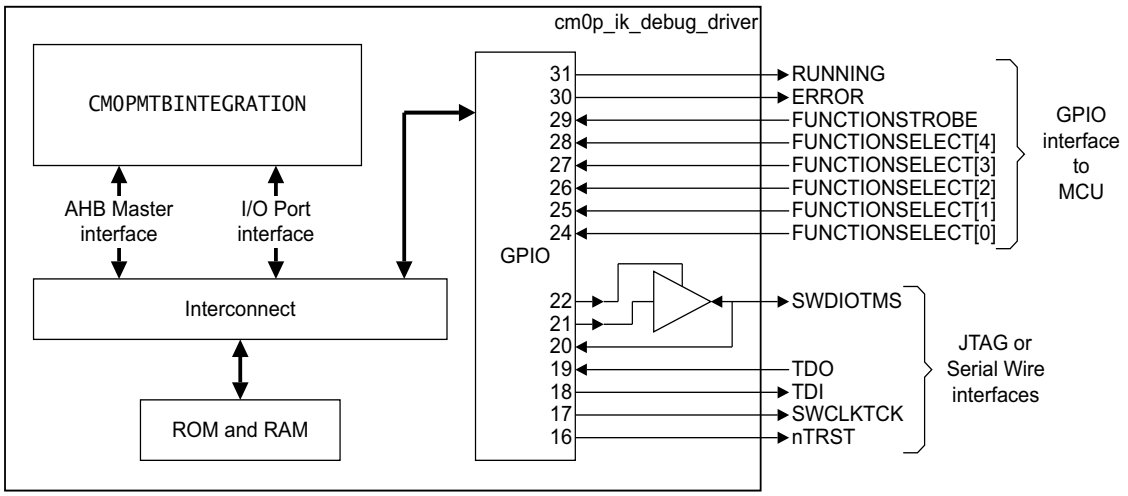


Figure D-1 Debug driver

Note

- During netlist simulations, the netlist replaces the RTL in both CM0PIKMCU 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 E
Power Intent

This appendix describes the optional State Retention and Power Gating features of the Cortex-M0+ processor. It contains the following section:

.

E.1 Cortex-M0+ 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:

E.1.1 Power intent specification

The *Unified Power Format* (UPF) and *Common Power Format* (CPF) files specify the power intent for the Cortex-M0+ processor. The UPF files are located in the `logical/models/upf` directory and the CPF files are located in the `logical/models/cpf` directory. Both file formats contain equivalent information. 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 or CPF 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. The Integration Kit supports power-aware simulation. See Chapter 8 *Integration Kit*.

You must only use one power intent file in your system. If your implementation includes both the `CORTEXM0PLUSIMP` and `CM0PINTEGRATION` module levels, then you must only use the `CM0PINTEGRATION` power intent file. If you include `CORTEXM0PLUSIMP` module level alone, then you must only use the `CORTEXM0PLUSIMP` power intent file. This appendix describes the power intent at the `CM0PINTEGRATION` level 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/models/power_intent_<tech>`.

Note

The power intent files are suitable for RTL configurations that have the DBG configuration option value set to 1. See *Configuration options* on page 2-4 for details. If you use a DBG configuration option value of 0, then you must remove the statements in the power intent file that use the PD_DBG domain.

E.1.2 Power domains

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

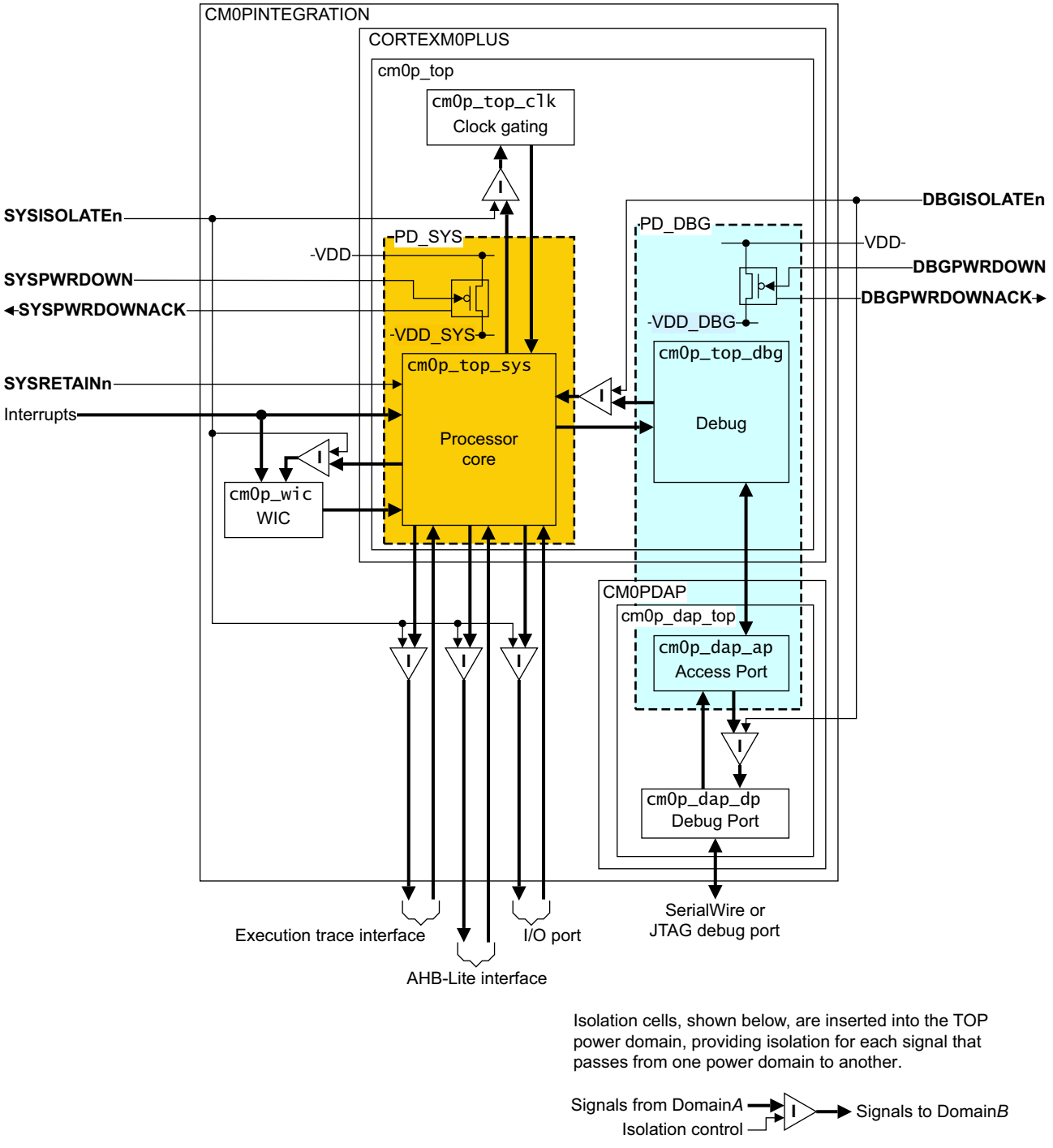


Figure E-1 Cortex-M0+ power domains

In Figure E-1 blocks that are not colored are in the TOP power domain. The colored blocks show the PD_SYS and PD_DBG power domains. Additionally, 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 E-1 on page E-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 E-1 describes which blocks are supplied by each domain.

Table E-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 and WIC logic.
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 and the access port logic in the DAP. 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.

E.1.3 Low power modes

The Cortex-M0+ power intent supports three low power modes Run, Sleep and Debug. See the Low power modes section in the Cortex-M0+ Technical Reference Manual for more information. Table E-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 E-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.

E.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 these ports and the **WAKEUP**, **WICSENSE**, **WICENREQ**, **WICENACK**, **CDBGPWRUPREQ** and **CDBGPWRUPACK** signals. See *Power Management Unit interface* on page 4-27.

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

Table E-3 Power Control Signals

Signal	Direction	Active sense	Description
SYSPWRDOWN	Input	HIGH	PD_SYS domain power down control.
SYSPWRDOWNACK	Output	HIGH	PD_SYS domain power down acknowledge.
SYSISOLATEn	Input	LOW	PD_SYS domain isolate control.
SYSRETAINn	Input	LOW	PD_SYS domain retention control.
DBGPWRDOWN	Input	HIGH	PD_DBG domain power down control.
DBGPWRDOWNACK	Output	HIGH	PD_DBG domain power down acknowledge.
DBGISOLATEn	Input	LOW	PD_DBG domain isolate control.

E.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 E-1 on page E-4 shows the location of the isolation cells. Where isolation cells are inferred, Figure E-1 on page E-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 E-4.

Table E-4 Clamp HIGH output signals

Domain	Module name	Output signal
PD_SYS	`cm0p_top_sys`	**sleeping_o**
		sleep_deep_o
PD_DBG	`cm0p_top_dbg`	**dbg_restarted_o**

E.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 E-5 shows the state retention information.

Table E-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**	HIGH	**SYSRETAINn**	LOW

E.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 F
Processor Integration Layer

This appendix describes the Processor Integration Layer. It contains the following sections:

F.1 CoreSight processor integration layer

The CORTEXM0PLUSINTEGRATIONCS level, also referred to as the Processor Integration Layer or PIL, supports integration of the Cortex-M0+ processor, Cortex-M0+ CTI, and optional CoreSight MTB-M0+ into an SoC design alongside other processors and debug components to form a CoreSight compliant debug system. The PIL consists of a Verilog RTL module, IP-XACT description, and some additional files to support integration using the ARM *CoreSight SoC* (CSSoC) tool.

For descriptions of the CORTEXM0PLUSINTEGRATIONCS PIL content, configuration, and interfaces see:

- Figure 1-2 on page 1-4.
- *CORTEXM0PLUSINTEGRATIONCS configuration options* on page 2-10.
- Chapter 3 *Key Integration Points*.
- Chapter 4 *Functional Integration Guidelines*.

The PIL includes the following AHB-Lite ports:

- AHB-Lite External Debug Slave port for connection to a debugger using CSSoC.
- AHB-Lite master port for connection to your memory system and peripheral components.
- AHB-Lite Debug Component Slave port that provides access to the PIL ROM table, CTI, and optional MTB.

You must ensure that your system interconnect routes accesses in the PIL DCS address range back into the DCS port.

The PIL ROM table identifies the PIL and its contents to an external debugger. The ROM table ID reports ARM as the manufacturer, and the part number as 0x4C1 meaning the Cortex-M0+ PIL. You must not change these values.

F.1.1 CTI connectivity

The CTI in the PIL communicates events with other parts of your debug system. You can use the CSSoC tool to connect the CTI channel interface **CTICHIN** and **CTICHOUT** to a *Cross Trigger Matrix* (CTM) in your system.

The PIL exports two interrupt outputs, **CTIIRQ[1:0]**, from the CTI. For debug purposes, you should connect these signals to two processor interrupt inputs. You can share these interrupts with existing interrupt sources if required. You can only use **CTIIRQ** when debugging. You should select a **CTIIRQ** to **IRQ** assignment that is the least invasive to your system when you use this debugging feature. The PIL instantiates the Cortex-M0+ CTI. See Appendix G *Cortex-M0+ CTI* for details of the CTI trigger connectivity.

PIL memory map

The debug components in the PIL share the same memory space as the processor system. You must build your system level interconnect so that the PIL Debug Component Slave (DCS) AHB-Lite port is accessed for the address ranges of the PIL components.

Table F-1 PIL memory map

Address Range	Component
0xF0000000 - 0xF0000FFF	PIL primary ROM table.

Table F-1 PIL memory map (continued)

Address Range	Component
0xF0001000 - 0xF0001FFF	CTI.
0xF0002000 - 0xF0003FFF	MTB.
MTBSRAMBASE - MTBSRAMBASE + (2^AWIDTH)	MTB SRAM.

PIL security considerations

Any privileged code running on the processor can access the PIL debug components, which allows it to inject debug events to the CTI channels. This behavior enables the software running on the processor to halt or restart other processors in a multi-core system.

If you need to disable this behavior, you must perform the following:

1. You must modify the address decoding inside the AHB interconnect so that the CTI can only be accessed when a debugger accesses it.
2. You must add custom event gating logic to the debug event interface. You can program this gating logic so that the debugger can decide whether the processor is permitted to inject debug events into a specified channel.

F.2 Using the Cortex-M0+ PIL with CoreSight SoC

The CSSoC tool allows you to build CoreSight debug systems for complex SoC designs. The tool:

- Provides a set of configurable debug and trace components.
- Provides an environment for graphically configuring the components.
- Provides design rule checks to test for CoreSight compliance.
- Creates a test bench infrastructure to run system-level tests to confirm debug and trace operation.

The CSSoC flow relies on IP-XACT views of all of the components which are to be integrated and tested. IP-XACT views of the configurable debug components are supplied with the CSSoC tool. The IP-XACT view of the Cortex-M0+ PIL is supplied with the Cortex-M0+ processor.

Note

The CSSoC tool is not provided with this processor. CSSoC may be licensed separately from ARM.

F.2.1 CoreSight SoC PIL support files

Figure F-1shows the main directories used by the CSSoC tool when you have installed the deliverables and rendered the processor.

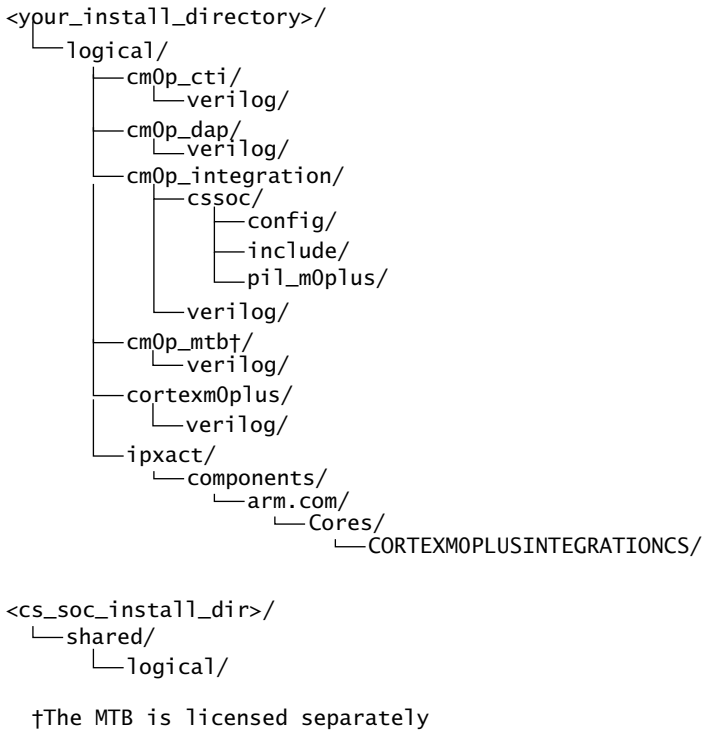


Figure F-1 CoreSight integration flow directories

F.2.2 Testbench structure

This section describes the testbench and its contents, including the memory map. It contains the following sub-sections:

- *Testbench memory map*.
- *Debug APB memory map*.

Testbench memory map

The testbench consists of the CSSYS, PIL, and testbench components.

Figure F-2 shows the testbench memory map.

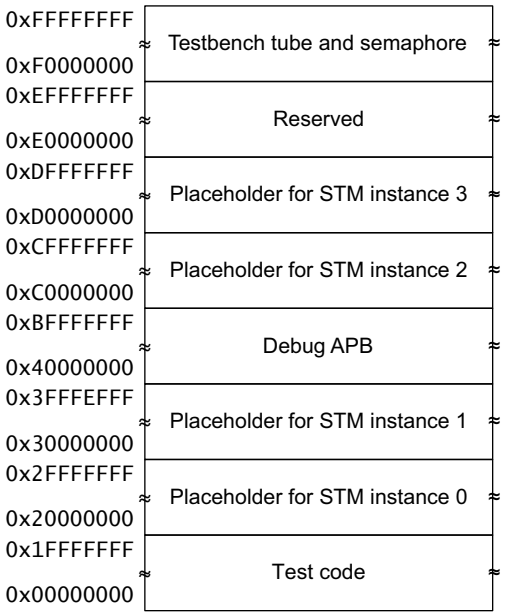


Figure F-2 Testbench memory map

Debug APB memory map

The CoreSight components in the PIL are accessed through the debug AHB interface. The CoreSight components in the testbench and CSSYS, such as the CSCTI, are accessed through the debug APB interface.

Figure F-3 shows an example debug APB memory map for the Cortex-M0+ processor.

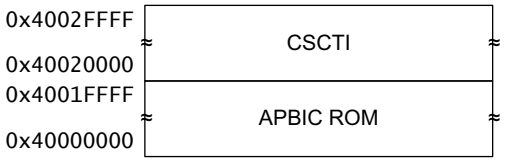


Figure F-3 Debug APB memory map for the Cortex-M0+ PIL example system

F.2.3 Example system for Cortex-M0+ processor

You can use the provided example system shown in Figure F-4 on page F-7as a reference for integrating a CortexM0+ PIL with the debug and trace components. The example system contains the following:

- *DAP subsystem*.
- *Testbench components*.

DAP subsystem

The DAP subsystem consists of the following:

- SWJ-DP.
- DAPBUSIC.
- AHB-AP.
- APB-AP.

AHB-AP is connected to the processor through the AHB-Lite interface.

APB-AP is connected to system-level APB interconnect through APB_Slave0.

Testbench components

The following testbench components are instantiated at the top-level:

- Dedicated *Network InterConnect* (NIC).
- NIC TOP.
- tbench_mem.
- tbench_tube_mem.

A dedicated NIC is connected to the PIL and has peripherals or memory in the address range `0x0000_0000-0x0003_FFFF`, and routes accesses in the address range `0xE000_0000-0xF000_FFFF` into the PIL DCS port.

Accesses outside of these address ranges are forwarded to the NIC TOP. The test code is loaded into the tbench_mem. The tbench_tube_mem serves as a semaphore location for synchronization in the testcases.

Table F-2 shows the address ranges.

Table F-2 Address ranges

Address range	Description
`0x00000000 - 0x0003FFFF`	Goes to the code memory.
`0x00040000 - 0xDFFFFFFF`	Goes to the NIC TOP.
`0xE0000000 - 0xF000FFFF`	Goes to the DCS port of the Cortex-M0+ PIL.
`0xE0000000 - 0xF000FFFF`	Goes to the NIC TOP.

Cortex-M0+ PIL example system

Figure F-4 on page F-7 shows the example system for Cortex-M0+ PIL

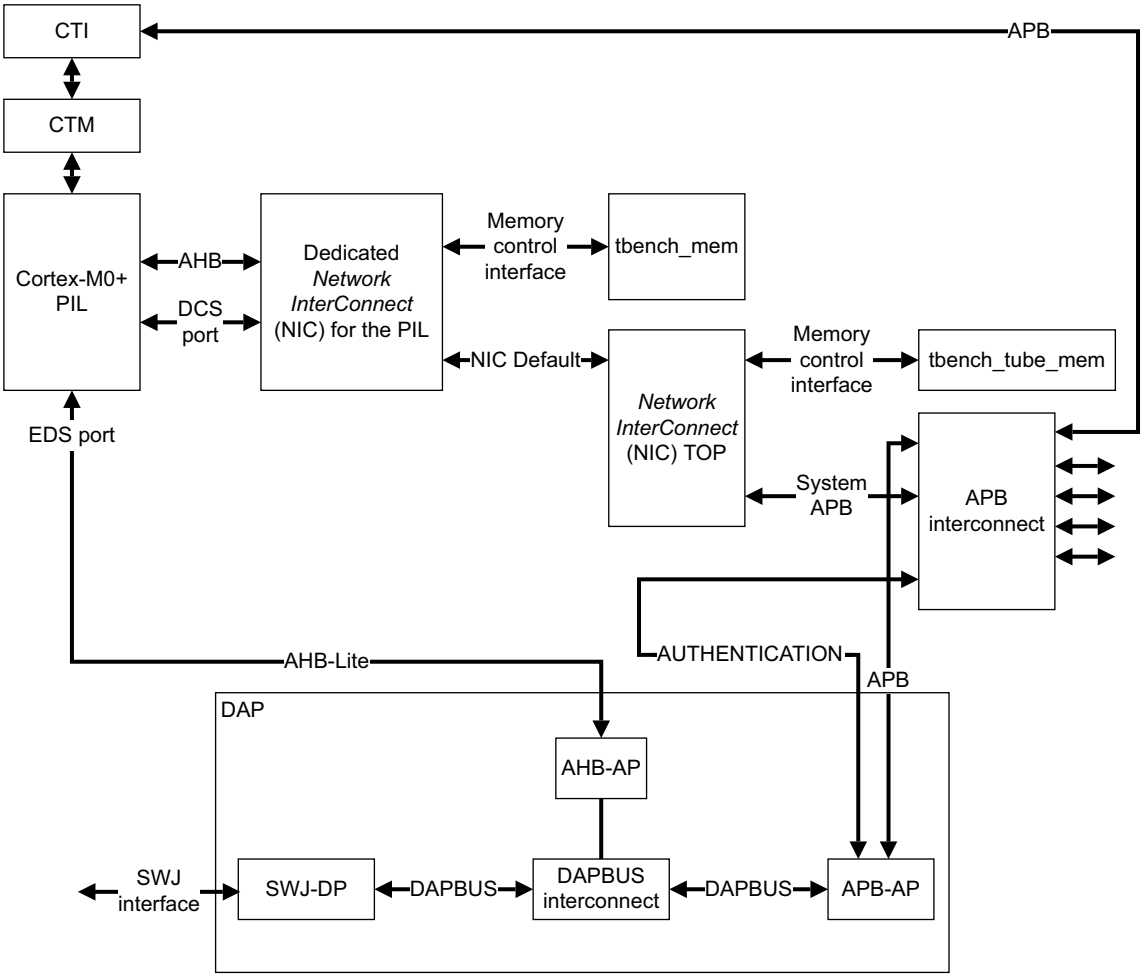


Figure F-4 Example system for Cortex-M0+ PIL

F.2.4 Test overview

The CSSoC infrastructure enables generation of testcases automatically based on the system topology. For more information on how to generate testcases and testcase descriptions, see the *CoreSight SoC User Guide*.

F.2.5 Installing and running the Cortex-M0+ PIL example system

This section describes the procedure for:

- Configuring the Cortex-M0+ PIL IP-XACT description and adding it to CSSoC.
- Rendering the Cortex-M0+ PIL example system.
- Running CSSoC tests.

See the *CoreSight SoC User Guide* for further details of the CSSoC product, including:

- Details of the CSSoC tests.
- How you can build your own system or customise the example system.

Cortex-M0+ PIL

To install and run the Cortex-M0+ PIL example system:

1. Set up your CSSoC tools.

   See the *CSSoC User Guide* for details of how to install and setup the tools so that they are ready to use.

2. Add bus definitions files to AMBA Designer.

   See Appendix J *IP-XACT* for details of the bus definitions used by the Cortex-M0+ PIL. See the AMBA Designer User Guide for details of how to add bus definitions to your workspace.

3. Create a configured version of the Cortex-M0+ PIL IP-XACT description.

   Note

   See Chapter 2 *Configuration Guidelines* for information on configuration options.

   a. Go to the directory where the build scripts reside by typing:

   ```
   cd <M0+-install-dir>/ipxact/tools
   ```

   b. Type:

   ```
   ./Build_Component
   <M0+-install-dir>/ipxact/components/arm.com/Cores/CORTEXM0PLUSINTEGRATIONCS/
   r0p1_0/CORTEXM0PLUSINTEGRATIONCS.xml
    default_cfg -config
   <M0+-install-dir>/logical/cm0p_integration/cssoc/config/default_cfg.conf
   ```

   Note

   - These examples use the `r0p1_0` directory. You must use the directory appropriate to the revision of the processor and IP-XACT file.
   - While running the `Build_Component` script, the warning in Example F-1 might appear. You can safely ignore this.

Example F-1 Configuration warning message

```
Use of uninitialized value in string at ./Build_Component line 64.
Empty compile time value given to use lib at ./Build_Component line 64.
```

4. Add the configured PIL to the AMBA Designer library by typing:

   ```
   adcanvas --addcomp
   <M0+-install-dir>/ipxact/components/arm.com/Cores/CORTEXM0PLUSINTEGRATIONCS/r0p1_0
   /CORTEXM0PLUSINTEGRATIONCS_default_cfg.xml
   ```

5. Render the example system with the PIL configuration by typing:

   ```
   ae render
   --cfg=<M0+-install-dir>/logical/cm0p_integration/cssoc/pil_m0plus/pil_cortexm0plus
   _default_cfg/pil_cortexm0plus_default_cfg_design.xml --gui
   --targetdir=<M0+-install-dir>/logical/cm0p_integration/cssoc/logical
   ```

   Note

   This command creates the example system `Verilog RTL` beneath the given `targetdir`.

6. Change directory to the target directory by typing:

```
cd
<M0+-install-dir>/logical/cm0p_integration/cssoc/logical/pil_cortexm0plus_default_
cfg_RTL
```

7. Generate the testbench and the testcases by typing:

   ```
   ae gen_tb --cfg=pil_cortexm0plus_default_cfg --gui
   ```

8. Compile the testbench by typing:

   ```
   ae compile --cfg=pil_cortexm0plus_default_cfg --gui
   ```

9. Build and simulate all the generated testcases by typing:

   ```
   ae simulate --cfg=pil_cortexm0plus_default_cfg
   ```

   Note

   To obtain a list of available testcases and simulate options, type:

   ```
   ae simulate --cfg=pil_cortexm0_default_cfg --help
   ```

   For information on using the AMBA Designer GUI to generate, compile, and simulate the testcases, see the *CoreSight SoC User Guide*.

10. Check for correct completion of the testcases by looking at the log files in:

    ```
    <M0+-install-dir>/logical/cm0p_integration/cssoc/logical/pil_cortexm0plus_default_
    cfg_RTL/logical/pil_cortexm0plus_default_cfg/validation/logs/
    ```

Appendix G
Cortex-M0+ CTI

This appendix describes the Cortex-M0+ *Cross Trigger Interface* (CTI)

G.1 Cortex-M0+ CTI

The Cortex-M0+ CTI is an implementation of the CoreSight Cross Trigger Interface architecture version CTI v2 which has been optimized for use with the Cortex-M0+ processor and optional CoreSight MTB-M0+ Micro Trace Buffer as follows:

- Fully synchronous clocking only (synchronous to the processor clock, **HCLK**).
- 4 channel inputs and output.
- 1 trigger input, 6 trigger outputs.
- Direct signal interfacing to the processor and MTB.
- AHB-Lite programming interface.

The Cortex-M0+ CTI is supplied in `logical/cm0p_cti/verilog/CM0PCTI.v`

If you require the CTI channel interface to be clocked asynchronously to the processor, you may add an appropriate bridge, or replace the Cortex-M0+ 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 Cortex-M0+ CTI top level.

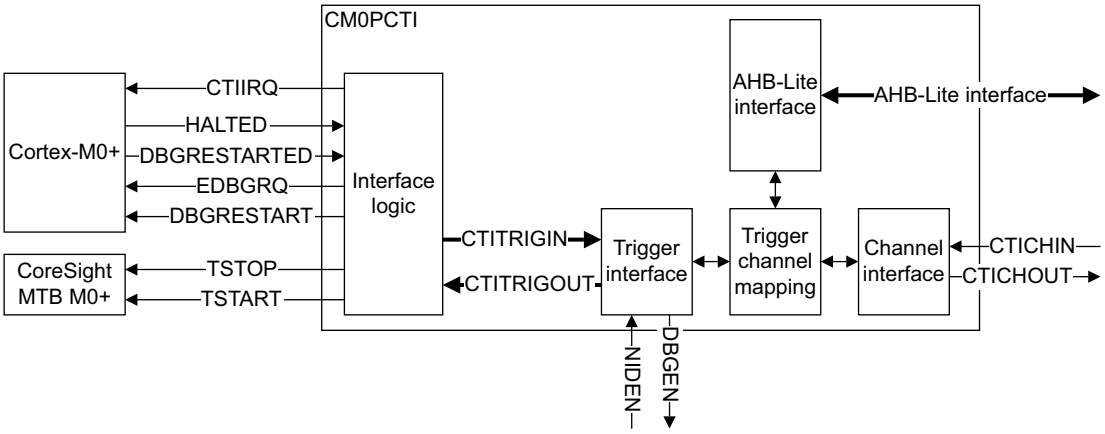


Figure G-1 Cortex-M0+ CTI

G.1.1 Connecting Cortex-M0+ to the CTI

The Cortex-M0+ CTI is an implementation of the CoreSight CTI architecture that is optimized for use with the Cortex-M0+ processor and CoreSight MTB-M0+ Micro Trace Buffer.

Table G-1 shows how the CTI architected trigger inputs are connected to the processor.

Table G-1 Trigger signals to the CTI

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[7]	-	-	-
CTITRIGIN[6]	-	-	-
CTITRIGIN[5]	-	-	-
CTITRIGIN[4]	-	-	-

Table G-1 Trigger signals to the CTI (continued)

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[3]	-	-	-
CTITRIGIN[2]	-	-	-
CTITRIGIN[1]	-	-	-
CTITRIGIN[0]	**HALTED**	Processor to CTI.	pulsed.

Table G-2 shows how the CTI architected trigger outputs are connected to the processor and MTB.

Table G-2 Trigger signals from the CTI

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	**DBGRESTART**	CTI to processor.	Generated from **DBGRESTARTED**, **HALTED**.
CTITRIGOUT[6]	-	-	-
CTITRIGOUT[5]	**TSTOP**	CTI to MTB	pulsed
CTITRIGOUT[4]	**TSTART**	CTI to MTB	pulsed
CTITRIGOUT[3]	**CTIIRQ[1]**	CTI to system.	Acknowledged by writing to the CTIINTACK register in ISR.
CTITRIGOUT[2]	**CTIIRQ[0]**	CTI to system.	Acknowledged by writing to the CTIINTACK register in ISR.
CTITRIGOUT[1]	-	-	-
CTITRIGOUT[0]	**EDBGRQ**	CTI to processor.	Acknowledged by the debugger writing to the CTIINTACK register.

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
Tarmac Trace Description

This section defines the Tarmac trace format for the Cortex-M0+ processor in:

H.1 About the tarmac trace file

The tarmac trace file is generated by the Verilog module `cm0p_tarmac.v` during a processor simulation. It is a normal text file that you can view in a text editor. Each line in the file shows a specific trace type output from the module as described in *Types of trace*. Each trace line is time stamped with the simulation time or, optionally, the cycle count. Because there can be more that one trace type output for a particular timestamp they are written to the file as they are generated by the module, interleaved in time order.

Table H-1 shows the different trace types.

Table H-1 Types of trace

Trace type	Description
Instruction trace	Provides information about retired instructions.
Register trace	Keeps track of changes to the currently visible register bank.
Event trace	Describes the interrupts and exceptions that occur during program execution.
Memory trace	Provides information about data side accesses to the memory system.

H.2 Trace format definition and examples

Table H-2 shows the values used in the trace format.

Table H-2 Trace format values

Value	Description
<time>	Simulation time or cycle when the instruction is retired.
<timescale>	The timescale can be either the time unit as determined in the simulation or hclk when a cycle count is used.
<i_addr>	Address of the instruction.
<opcode>	16-bit or 32-bit hexadecimal opcode of the instruction.
<disasm>	Disassembly for the opcode.
<register>	Register name.
<value>	The hexadecimal value written to the register, up to 64 bits.
<event_name>	See the tarmac module cm0p_tarmac.v.
<event_data>	See the tarmac module cm0p_tarmac.v.
<sz>	Size of the data transfer in bytes, 1, 2, or 4.
<m_addr>	Address of the memory location accessed.
<data>	Hexadecimal value of data transferred, little-endian, up to 64 bits.
<branch_addr>	Target of branch instruction.

H.2.1 Instruction trace

At instruction retirement, the following formatted trace is generated:

```
<time> <timescale> [IT|IS] <i_addr> <opcode> <disasm> ; <branch_addr>
```

Where IT and IS indicate:

IT Instruction Taken (executed).

IS Instruction Skipped (not executed).

For example:

```
23870 ns IT 0000034c 18a2 ADDS r2,r4,r2
```

H.2.2 Register trace

Updated registers generate the following formatted trace:

```
<time> <timescale> R <register> <value>
```

For example:

```
23930 ns R r5 00000003
```

H.2.3 Event trace

Interrupts or exceptions generate the following formatted trace:

```
<time> <timescale> E <event_name> <event_data>
```

For example:

```
24170 ns E TUBE 40400000 45 ; "12345678 0x9ABCD" ... "E"
```

H.2.4 Memory trace

Memory operations generate the following formatted trace:

```
<time> <timescale> M[R|W]<sz>[_|E][I|D|P|S] <m_addr> <data>
```

Where `R` and `W` indicate:

`R` Read access.

`W` Write access.

Where `_` and `E` indicate:

`_` Access completed normally.

`E` Access caused an Exception.

Where `I, D` and `P` indicate:

`I` AHB Instruction access (Instruction fetch).

`D` AHB Data access.

`P` IOP data access.

`S` Access initiated from the debug slave port.

For example:

```
23950 ns MR4_I 00000354 200abc30
```

Appendix I
Signal Timing Constraints

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

I.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 I-1 shows the AHB-Lite master interface timing constraints.

Table I-1 AHB-Lite master interface

Signal	Timing constraint	Direction
HTRANS[1:0]	50	Output
HSIZE[2:0]	60	Output
HMASTLOCK	60	Output
HWRITE	60	Output
HPROT[3:0]	60	Output
HBURST[2:0]	60	Output
HADDR[31:0]	60	Output
HWDATA[31:0]	60	Output
HRDATA[31:0]	60	Input
HREADY	60	Input
HRESP	60	Input

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

Table I-2 AHB interface extensions

Signal	Timing constraint	Direction
SPECHTRANS	80	Output
HMASTER	60	Output
SHAREABLE	60	Output
CODENSEQ	80	Output
CODEHINT [3:0]	20	Output
DATAHINT[1:0]	60	Output

Table I-3 shows the IO port interface output timing constraints.

Table I-3 IO port output timing constraints

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

Table I-4 shows the combinatorial paths from processor outputs to inputs.

Table I-4 IO port combinatorial paths

Start	End	Timing constraint
IOCHECK	**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

Table I-5 shows the debug slave interface timing constraints.

Table I-5 Debug slave interface

Signal	Timing constraint	Direction
SLVRDATA[31:0]	20	Output
SLVREADY	20	Output
SLVRESP	20	Output
SLVTRANS[1:0]	20	Input
SLVSIZE[1:0]	20	Input
SLVPROT[3:0]	20	Input
SLVWRITE	20	Input
SLVADDR[31:0]	20	Input
SLVWDATA[31:0]	20	Input
SLVSTALL	60	Input

Table I-6 shows the WIC interface timing constraints.

Table I-6 WIC interface

Signal	Timing constraint	Direction
WICDSACKn	60	Output
WICLOAD	60	Output
WICCLEAR	60	Output
WICMASKISR[31:0]	60	Output
WICMASKNMI	60	Output
WICMASKRXEV	60	Output
WICDSREQn	60	Input

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

Table I-7 Sleep-control interface

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

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

Table I-8 Interrupts and events

Signal	Timing constraint	Direction
TXEV	60	Output
RXEV	60	Input
IRQ[31:0]	60	Input
NMI	60	Input

Table I-9 shows the status timing constraints.

Table I-9 Status

Signal	Timing constraint	Direction
LOCKUP	60	Output
HALTED	60	Output

Table I-10 shows the debug timing constraints.

Table I-10 Debug

Signal	Timing constraint	Direction
DBGRESTARTED	60	Output
DBGRESTART	60	Input
EDBGRQ	60	Input
NIDEN	10	Input
DBGEN	10	Input

Table I-11 shows the configuration timing constraints.

Table I-11 Configuration

Signal	Timing constraint	Direction
STCALIB[25:0]	60	Input
ECOREVNUM[19:0]	60	Input

Table I-12 shows the miscellaneous signals timing constraints.

Table I-12 Miscellaneous signals

Signal	Timing constraint	Direction
SYSRESETREQ	90	Output
STCLKEN	80	Input
CPUWAIT	80	Input
IRQLATENCY[7:0]	80	Input
DFTSE[a]	-	Input
DFTRSTDISABLE	80	Input

a. This signal has high fanout and must be treated as multicycle.

Table I-13 shows the PC monitoring and branch signals timing constraints.

Table I-13 PC monitoring and branch trace

Signal	Timing constraint	Direction
ATOMIC	80	Output
IAEX[30:0]	80	Output
IAEXEN	40	Output
IAEXSEQ	40	Output

Table I-14 shows the JTAG and serial wire interface timing constraints.

Table I-14 JTAG and Serial Wire interface

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

Table I-15 shows the WIC-PMU interface timing constraints.

Table I-15 WIC-PMU interface

Signal	Timing constraint	Direction
WICENACK	60	Output
WAKEUP	60	Output
WICSENSE[33:0]	60	Output
WICENREQ	60	Input

Table I-16 shows the PMU interface timing constraints.

Table I-16 PMU interface

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

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

b. **CDBGPWRUPACK** is asynchronous

Table I-17 shows the configuration signals timing constraints.

Table I-17 Configuration

Signal	Timing constraint	Direction
ECOREVNUM[27:0]	60	Input
INSTANCEID[3:0]	60	Input

Appendix J
IP-XACT

This appendix describes the location and configuration of the IP-XACT files in the following sections:

J.1 Location of the IP-XACT files

Table J-1 shows the IP-XACT version 1.4 description files that are included for the following levels of hierarchy

Table J-1 IP-XACT files

Verilog module	IP-XACT description	Description
CORTEXM0PLUS.v	CORTEXM0PLUS/r0p1_0/CORTEXM0PLUS.xml	The processor top level.
CM0PINTEGRATION.v	CM0PINTEGRATION/r0p1_0/CM0PINTEGRATION.xml	The processor and DAP integration level.
CORTEXM0PLUSINTEGRATIONCS.v	CORTEXM0PLUSINTEGRATIONCS/r0p1_0/CORTEXM0PLUSINTEGRATIONCS.xml	The processor, CTI and MTB integration level or PIL.

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/Cores/` directory. The revision number, rXpY_Z, is formed from the revision number of the processor, rXpY, and the revision number of the source IP-XACT file, Z.

Note

There is no IP-XACT description for the `CM0PMTBINTEGRATION` level of hierarchy.

J.2 Generating the IP-XACT description

This section describes how you can generate an IP-XACT description based on your chosen configuration of the IP.

To generate an IP-XACT description, you must run the `Build_Component` script on an unconfigured, source IP-XACT file, and supply the Verilog configuration parameters described in *Configuration options* on page 2-4.

Note

- The `Build_Component` script uses the `IPXACT_lib.pm` module. The script requires Perl utilities and Perl XML libraries.
- For more information about the script, type `Build_Component -h`.

The filename of the generated IP-XACT file is the source IP-XACT filename with a uniquifier appended. You can supply a specific string to use as the uniquifier. If you do not supply a string, `Build_Component` generates a string based on your configuration.

You can run the script with or without a configuration file:

- With a configuration file:

  Run `Build_Component`, supplying the name of the IP-XACT file and the configuration file.

  Example:

  ```
  ./Build_Component ../components/arm.com/Cores/CORTEXM0PLUS/r0p1/CORTEXM0PLUS.xml
  [uniquifier] -config <config_file>
  ```

  The configuration file should be in the format:

  ```
  parameter_name=parameter_value
  ```

- Without a configuration file:

  Run `Build_Component`, supplying the name of the IP-XACT file. The script prompts for configuration parameters to be typed in.

  Example:

  ```
  ./Build_Component ../components/arm.com/Cores/CORTEXM0PLUS/r0p1/CORTEXM0PLUS.xml
  ```

  You can save the parameters you typed in when the script prompts you.

The `Build_Component` script generates the configured IP-XACT file in the same directory as the unconfigured source IP-XACT file.

J.3 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 `ipxact/busdefs` directory.

Additionally, the IP-XACT descriptions reference the spiritconsortium interrupt bus definition files. You can download this from `http://www.accellera.org`.

Figure J-1shows the structure of the IP-XACT busdefs directory.

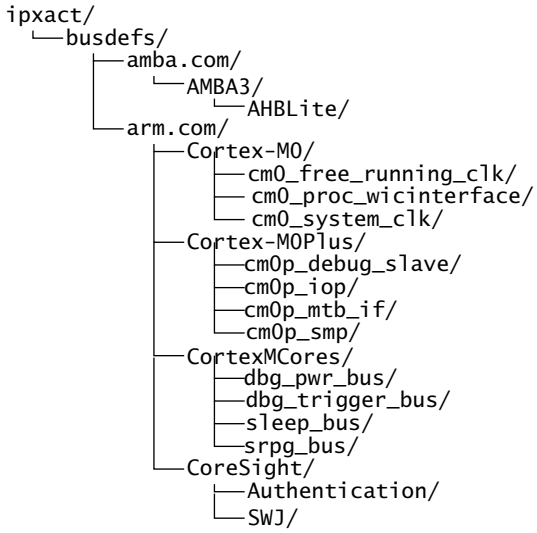


Figure J-1 IP-XACT busdefs directory structure

Appendix K
Revisions

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

Table K-1 Issue A

Change	Location	Affects
This is the first release of this guide.	-	-

Table K-2 Differences between issue A and issue B

Change	Location	Affects
Clarified description of `IRQDIS` and `NUMIRQ`.	Table 2-1 on page 2-4	All revisions
Updated `CM0PINTEGRATION` configuration options description.	*CM0PINTEGRATION configuration options* on page 2-6	All revisions
Updated description of `HALTEV`.	Table 2-2 on page 2-6	All revisions
Updated description of `MPU`.	Table 2-2 on page 2-6	All revisions
Updated description of `TARGETID`.	Table 2-2 on page 2-6	All revisions
Updated description of `USER`.	Table 2-2 on page 2-6	All revisions
Updated description of `CM0PMTBINTEGRATION` configuration options.	*CM0PMTBINTEGRATION configuration options* on page 2-9	All revisions
Added `USER` parameter.	Table 2-3 on page 2-9	All revisions
Updated description of **DBGRESTART** and **DBGRESTARTED**.	Table 4-19 on page 4-21	All revisions

Table K-2 Differences between issue A and issue B (continued)

Change	Location	Affects
Updated description of **SLEEPHOLDACKn** and **SLEEPHOLDREQn**.	Table 4-19 on page 4-21	All revisions
Updated description of **SYSRESETREQ**.	Table 4-19 on page 4-21	All revisions
Added new section on security considerations.	*Security Considerations* on page 4-33	All revisions
Added new section on CoreSight components.	*CoreSight* on page 4-34	All revisions
Added new section on Integration Kit components.	*IK components* on page 8-27	All revisions
Updated module name for CoreSight ROM table base address.	Table 8-3 on page 8-9	All revisions
Updated module name for CoreSight ROM table base address.	Table 8-10 on page 8-14	All revisions
Updated description of defines.	*Running RunIK with the -upf or -cpf options* on page 8-19	All revisions
Added description of ROM and ROM controller.	*ROM controller* on page 8-27	All revisions
Updated module name for System level ROM table.	*CM0PMTBINTEGRATION level* on page 8-23	All revisions
Updated filenames and locations.	Table 8-13 on page 8-30	All revisions
Added description for boot_cm0pikmcu.c.	Table 8-13 on page 8-30	All revisions
Updated description for IKtests.c.	Table 8-14 on page 8-31	All revisions
Updated description for Debug driver.	Appendix D *Debug Driver*	All revisions
Updated filenames and locations.	Table D-1 on page D-2	All revisions
Updated diagram to show Debug driver connections.	Figure D-1 on page D-3	All revisions
Removed **hprot_o[1]** clamp signal.	*Signal isolation* on page E-6	All revisions
Added new appendices.	Appendix F *Processor Integration Layer*	r0p1
	Appendix G *Cortex-M0+ CTI*	r0p1
	Appendix H *Tarmac Trace Description*	All revisions
	Appendix I *Signal Timing Constraints*	All revisions
	Appendix J *IP-XACT*	r0p1