Cortex®-M4

Revision: r0p1

Integration and Implementation Manual

Confidential

ARM®

ARM DII 0239B (ID062910)

Cortex-M4
Integration and Implementation Manual

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
22 December 2009	A	Confidential	First release for r0p0
29 June 2010	B	Confidential	Updated with information 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-M4 Integration and Implementation Manual

List of Tables
Cortex-M4 Integration and Implementation Manual

List of Figures
Cortex-M4 Integration and Implementation Manual

Preface

This preface introduces the *Cortex-M4 Integration and Implementation Manual* (IIM). It contains the following sections:

About this book

This book is for the Cortex-M4 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 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 an overview of the process of integrating and implementing the Cortex-M4 processor.

Chapter 2 *Configuration Guidelines*

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

Chapter 3 *Key Integration Points*

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

Chapter 4 *Functional Integration Guidelines*

Read this for a description of the guidelines for functional integration of the processor in your SoC design.

Chapter 5 *Integration Kit*

Read this for a description of the integration kit.

Chapter 6 *Key Implementation Points*

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

Chapter 7 *Floorplan Guidelines*

Read this for a description of the floorplan used as a starting point for your design.

Chapter 8 *Design For Test*

Read this for a description of the design for test features of the processor.

Chapter 9 *Netlist Dynamic Verification*

Read this for a description of how to use test vectors to verify the functionality of your implementation of the processor.

Chapter 10 *Sign-off*

Read this for a description of the sign-off criteria for your implementation.

Appendix A *GPIO Programmers Model*

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

Appendix B *CM4IKMCU GPIO Integration*

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

Appendix C *Debug Driver*

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

Appendix D *SysTick Examples*

Read this for examples of setting up SysTick timing.

Appendix E *Revisions*

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

Glossary Read this for definitions of terms used in this book.

Conventions

Conventions that this book can use are described in:

- *Typographical*
- *Signals* on page xiii.

Typographical

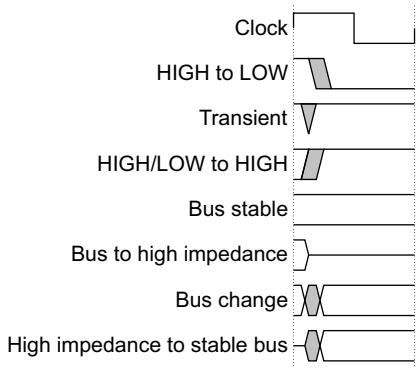
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal 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.

Timing diagrams

The figure named *Key to timing diagram conventions* on page xiii 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 onARM, `http://onarm.com`, for embedded software development resources including the *Cortex Microcontroller Software Interface Standard* (CMSIS).

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:
- *Cortex-M4 Technical Reference Manual* (ARM DDI 0439)
- *ETM-M4 Technical Reference Manual* (ARM DDI 0440)
- *ARM Debug Interface v5 Architecture Specification* (ARM 0031)
- *Embedded Trace Macrocell Architecture Specification* (ARM IHI 0014).
- *ARM AMBA® 3 AHB-Lite Protocol (v1.0)* (ARM IHI 0033)
- *ARM AMBA™ 3 APB Protocol Specification* (ARM IHI 0024)
- *CoreSight® Architecture Specification (v1.0)* (ARM IHI 0029)

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 DII 0239B
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter gives an overview of the process of integrating and implementing the Cortex-M4 processor. It contains the following sections:

1.1 About the processor

The Cortex-M4 processor is a very low gate count, highly energy efficient processor that is intended for microcontroller and deeply embedded applications that require an efficient mix of control capability and signal processing instructions.

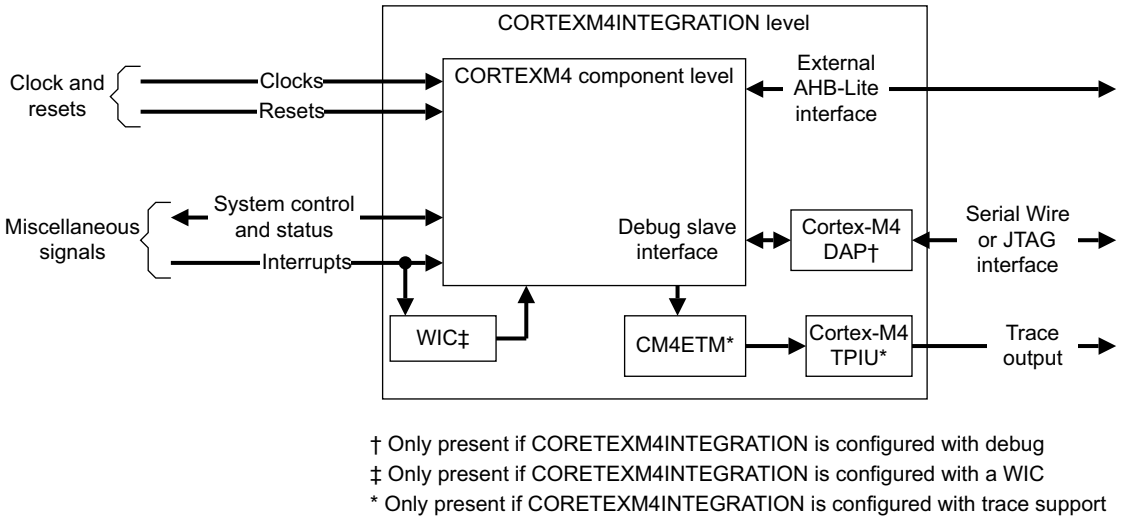
The Cortex-M4 processor supports two levels of hierarchy for integration or implementation:

- processor component level, `CORTEXM4`
- integration level, `CORTEXM4INTEGRATION`.

Figure 1-1 shows the integration level, the processor component level, and the main interfaces of the processor.

Note

You can modify the `CORTEXM4INTEGRATION` level for your own requirements.



† Only present if CORETEXM4INTEGRATION is configured with debug
‡ Only present if CORETEXM4INTEGRATION is configured with a WIC
* Only present if CORETEXM4INTEGRATION is configured with trace support

Figure 1-1 Module hierarchy and the main interfaces

CORTEXM4 component level

Contains the basic processor. This level includes optional components, configurable to your requirements. In particular, the floating point component is included at this level, if it is present. See *Floating point support* on page 2-5. You must not modify the CORTEXM4 component level RTL code.

CORTEXM4INTEGRATION level

Instantiates the Cortex-M4 processor component level and provides example integration with the Cortex-M4 *Debug Access Port* (DAP), *Wakeup Interrupt Controller* (WIC), *Embedded Trace Macrocell* (ETM) and *Trace Port Interface Unit* (TPIU). This level is provided as a working example of a single-processor sub-system. You can modify it to suit your requirements.

The Cortex-M4 processor is highly configurable. Although the top-level signals at both the integration and processor component levels are independent of configuration, the functionality of these interfaces is dependent on configuration. For example, the debug interfaces at either level are nonfunctional if your configuration does not include debug. For more details about configuring the `CORTEXM4INTEGRATION` integration or `CORTEXM4` component levels, see Chapter 2 *Configuration Guidelines*.

1.2 About integration and implementation

The flows that you can use to integrate a device in to your SoC can depend on the processing power of your design system and the EDA tools you use.

Figure 1-2 shows the integration and implementation flow.

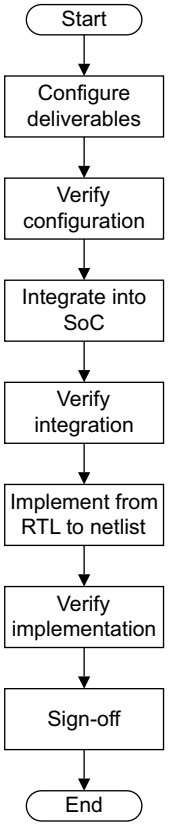


Figure 1-2 Integration and implementation flow

Figure 1-3 on page 1-4 shows the implementation and integration flow.

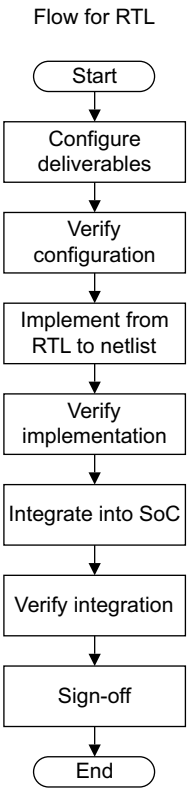


Figure 1-3 Implementation and integration flow

1.2.1 Integration

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

- connecting the required clocks and resets to the processor
- connecting the processor to all the required peripherals and buses
- implementing the chosen test strategies for the processor
- verifying the processor within the SoC design.

Possible considerations for your design depend on the elements that you include. You must consider:

- interfaces, especially the treatment of unused signals
- verification.

1.2.2 Implementation

Figure 1-4 on page 1-5 shows the top-level inputs, resources, outputs, and controls and constraints for implementation.

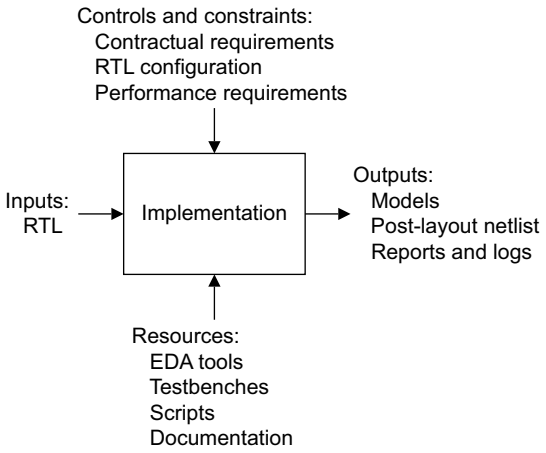


Figure 1-4 Implementation process

For an overview of the implementation process, see:

- *Implementation resources*
- *Implementation controls and constraints*
- *Implementation inputs* on page 1-6
- *Implementation flow* on page 1-6
- *Implementation outputs* on page 1-6.

For information on the deliverables see:

- *Reference data* on page 1-9.

1.2.3 Implementation resources

This book assumes that you have suitable EDA tools and compute resources for implementation. See the release note for a list of deliverables and any specific tool revisions required for implementation.

1.2.4 Implementation controls and constraints

Figure 1-4 shows the general controls and constraints that apply to the implementation. You must implement the device in accordance with your contract, see *Implementation obligations*.

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 minimize risks adequate design and operating safeguards should be provided for by you. Publication by ARM of any information in this document of information regarding any third party product or service is not an express or implied approval or endorsement of the use thereof.

1.2.5 Implementation inputs

The release note describes deliverables that are inputs to the implementation flow. These deliverables include:

- *Register Transfer Level* (RTL) code
- implementation scripts
- documentation.

1.2.6 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
  - SDF.
- Test:
  - *Automatic Test Pattern Generation* (ATPG) vectors.

1.2.7 Implementation flow

Figure 1-5 on page 1-7 shows the implementation flow.

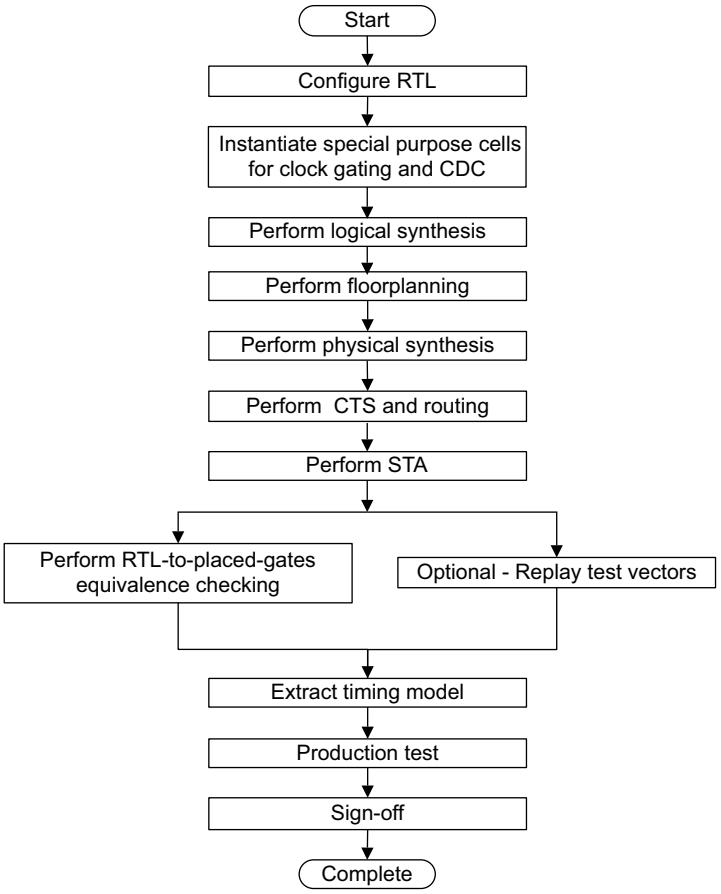


Figure 1-5 Implementation flow

Note

You have a contractual obligation to complete sign-off as part of the complete flow. For more information, see Chapter 10 *Sign-off*.

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 1-6 shows the principal directory structure when you unpack the deliverables.

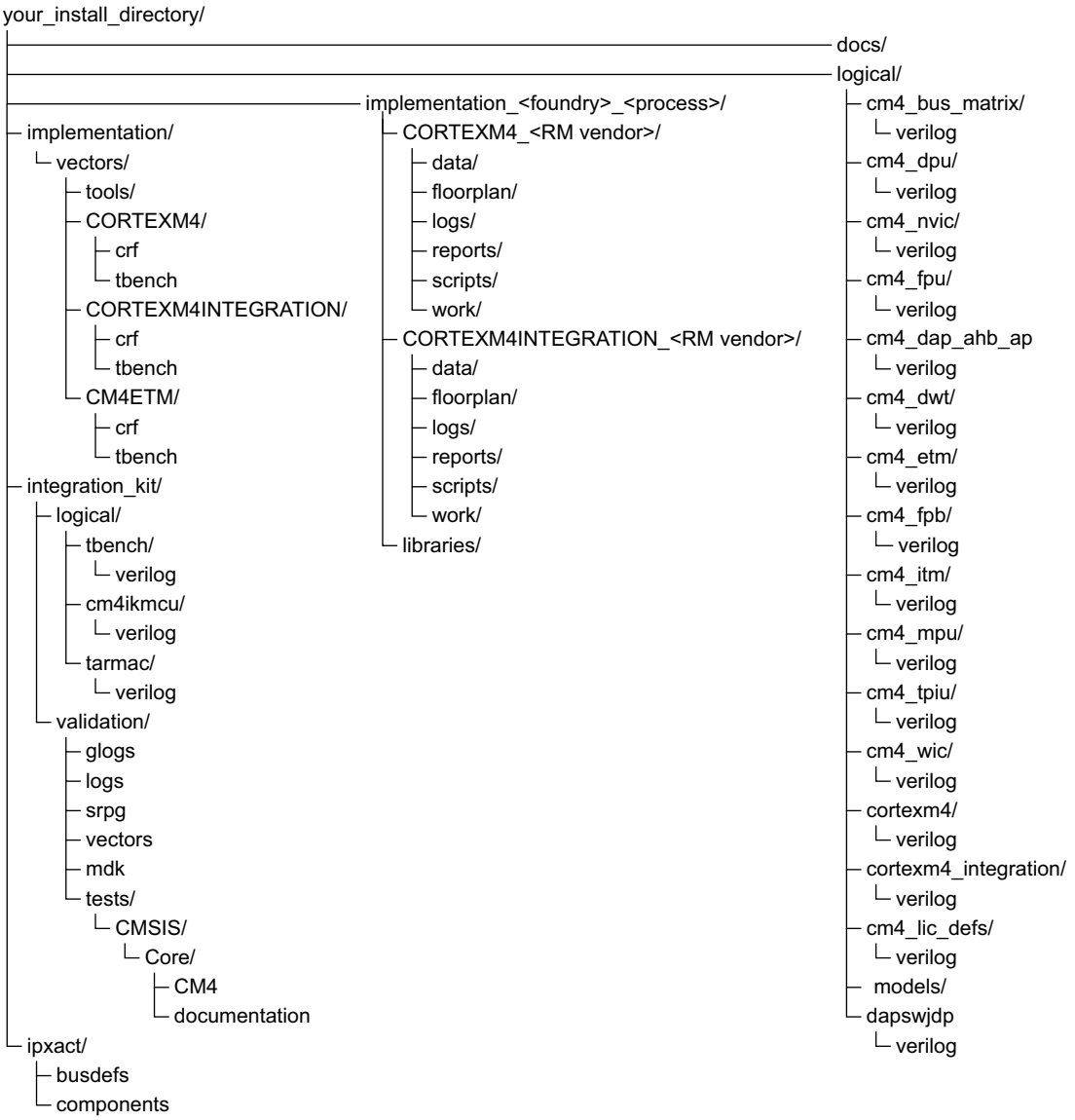


Figure 1-6 Release directory structure

Chapter 2
Configuration Guidelines

This chapter describes the guidelines for RTL configuration. You configure the RTL to tailor your implementation to the specific requirements of the target application. It contains the following sections:

2.1 About configuration guidelines

Caution

For successful configuration of the RTL you must set the configuration options to match your requirements. Failure to complete all the necessary configuration can result in malfunction.

The Cortex-M4 RTL includes two possible levels of RTL configuration:

- The `CORTEXM4` level implements the Cortex-M4 processor, the *Nested Vectored Interrupt Controller* (NVIC), breakpoint unit, watchpoint unit and associated debug control.
- The example `CORTEXM4INTEGRATION` level instantiates a `CORTEXM4` level pre-integrated with the WIC, the ETM, the TPIU and the Cortex-M4 Serial Wire or JTAG DAP.

The configurable options that are available depend on the level of hierarchy you are implementing. You control these options by defining parameter values at the integration level and the processor component level when you instantiate the level of the hierarchy you want to implement.

2.2 Configuration options

Parameters control all of the configuration options. You can either:

- alter these parameters
- override these parameters using instantiation.

Using parameters instead of defines makes it easier to integrate the processor into a multi-core system.

Table 2-1 shows the configuration options summary.

Table 2-1 Cortex-M4 configuration options

Parameter	Default value	Supported values	Description
NUM_IRQ	16	1-240	Specifies the number of user interrupts.
LVL_WIDTH	3	3-8	Specifies the bits of interrupt priority. At 3, there are 8 levels of priority, at 8 there are 256 levels.
MPU_PRESENT	0	0, 1	Specifies whether an MPU is present. Setting this parameter to 1 includes the MPU.
FPU_PRESENT	0	0, 1	Specifies whether a *Floating Point Unit* (FPU) is present. Setting this parameter to 1 includes the FPU.
BB_PRESENT	1	0, 1	Specifies whether bit-banding is supported. Setting this parameter to 0 removes the bit-banding logic.
AHB_CONST_CTRL	0	0, 1	Specifies whether the external AHB-Lite buses maintain control information during wait stated transfers. See *External AHB-Lite bus interfaces* on page 4-8 for more information.

Table 2-1 Cortex-M4 configuration options (continued)

Parameter	Default value	Supported values	Description
DEBUG_LVL	3	0, 1, 2, 3	Specifies the level of debug support: **0** No debug. Setting DEBUG_LVL to 0 removes the debug port, AHB-AP, FPB, DWT, and halting debug capability. This means the processor does not support breakpoints, watchpoints, stepping, vector catching, debug register transfers, or debugger access to memory. FPB removal also means that the processor does not support flash-patching. Debug monitor is still present and supported. **1** Minimal debug. All debug units are present but the debug capability is reduced to two breakpoints and one watchpoint. The DWT does not support data matching for watchpoint generation because it only has one comparator. This configuration does not support flash patching. All other debug functionality is present. **Note** If you require more than one trigger for trace then you must select a higher level of debug support. See *Trace support* on page 2-5. **2** Full debug without data matching. All debug functionality is present except for data matching for watchpoint generation. DATAVADDR0, DATAVADDR1, DATAVSIZE, and DATAVMATCH in the DWT Function Register 1 are not present. **3** Full debug with data matching. All debug functionality is present including data matching for watchpoint generation.
TRACE_LVL	1	0, 1, 2, 3	Specifies the level of trace support: **0** No trace. With no trace support the ETM, ITM, TPIU, HTM port, and the DWT triggers and counters are not present. **1** Standard trace. ITM, TPIU, and DWT triggers and counters are present. ETM and HTM port are not present. **2** Full trace. ITM, TPIU, ETM, and DWT triggers and counters are present. HTM port is not present. **3** Full trace plus HTM port. ETM, ITM, TPIU, and DWT triggers and counters are present. For more information about trace support parameters, see *Trace support* on page 2-5.
RESET_ALL_REGS	0	0, 1	Specifies whether all synchronous state or only architecturally required state is reset. **0** only resets architecturally required state **1** resets all synchronous state. **Note** Setting this parameter increases the size of the registers that are not reset by default, and therefore also increases the overall area of the implementation.

Table 2-1 Cortex-M4 configuration options (continued)

Parameter	Default value	Supported values	Description
JTAG_PRESENT	1	0, 1	Enables or disables the JTAG portion of the debug port. This switches between an SW-DP and an SWJ-DP. The default setting at 1, enables the JTAG portion, creating an SWJ-DP.
CLKGATE_PRESENT	0	0, 1	Specifies whether architectural clock gates are included to minimize dynamic power dissipation. This covers all levels of hierarchy, including the ETM. **Note** If you use architectural clock gating, you must replace the provided model with a direct instantiation of a clock gating cell from your target cell library.
WIC_PRESENT	0	0, 1	Specifies whether a WIC is present. Setting this parameter to 1 includes the WIC.
WIC_LINES	3	3-(3+NUM_IRQ)	Makes the WIC sensitive to a configurable number of interrupts or a configurable number of events. The WIC requires a minimum of two WIC_LINES, the Cortex-M4 core requires a minimum of three. If you want to assign two sensitivity lines to the WIC, the core and the WIC must have independent WIC_LINES parameters. The three default supported events are **NMI**, **EDBGRQ** and **RXEV**.

2.2.1 Trace support

A number of restrictions apply when you use trace support. These are:

- A nonzero TRACE_LVL is only supported if DEBUG_LVL is also nonzero. This means that there is no supported trace capability if DEBUG_LVL is 0.
- DEBUG_LVL overrides TRACE_LVL for the number of DWT comparators. This means that a full trace solution with minimal debug support only has one comparator for use as a trigger.
- You can only include the ETM if it is licensed, and if the appropriate define in logical/cm4_lic_defs/verilog/cm4_lic_defs.v is not commented out:

  ```
  `define ARM_CM4_ETM_LICENSE
  ```

2.2.2 Floating point support

You can only include the FPU if it is licensed and if the appropriate define in logical/cm4_lic_defs/verilog/cm4_lic_defs.v is not commented out:

```
`define ARM_CM4_FPU_LICENSE
```

2.3 Implementation constraints

This section describes constraints or considerations concerning implementation.

2.3.1 Architectural clock gating

The Cortex-M4 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 may be inferred by the synthesis tools during the implementation process. The architectural clock gating cells must be provided as modules named `cm4_clk_gate.v` and `cm4_etm_clk_gate.v`.

Architectural clock gating is optional because correct operation of the processor is not dependent on it. If architectural clock gating is not required and the `CLKGATE_PRESENT` parameter is set to `0`, the `cm4_clk_gate.v` and `cm4_etm_clk_gate.v` modules are bypassed.

If architectural clock gating is required and the `CLKGATE_PRESENT` parameter is set to `1`, the `cm4_clk_gate.v` and `cm4_etm_clk_gate.v` modules must directly instantiate a positive-edge clock gating cell from your target library. Correctly connect the clock input and outputs, clock enable and scan-enable signals.

For reference, simulation and logical-equivalence checking purposes, `logical/models/cells/cm4_clk_gate.v` and `logical/models/cells/cm4_etm_clk_gate.v` provide configurable versions of the `cm4_clk_gate` and `cm4_etm_clk_gate` modules. Do not use these versions for synthesis.

2.3.2 Special purpose cells

The Cortex-M4 processor has a number of interfaces that require specific consideration other than the logical behavior described in the RTL. The Cortex-M4 processor has the following *clock-domain crossing* (CDC) boundaries:

- in the DAP in `CORTEXM4INTEGRATION`, between **SWCLKTCK** and **DCLK**
- on the **CDBGPWRUPREQ** or **CDBGPWRUPACK** handshake, between **SWCLKTCK** in the DAP and an external *Power Management Unit* (PMU) using another clock
- in the TPIU, between **DCLK** and **TRACECLKIN**.

You must implement the following modules using cells with appropriate properties selected from your cell library for correct CDC operation.

- `logical/cm4_dap_ahb_ap/verilog/cm4_dap_ahb_ap_sync.v`
- `logical/dapswjdp/verilog/DAPDpSync.v`
- `logical/models/cells/cm4_sync.v`

The resets applied to `CORTEXM4INTEGRATION` must be synchronized appropriately to the corresponding clock.

You must ensure that the module cell `logical/models/cells/cm4_sync.v` is implemented in a glitch safe manner to synchronize **PORESETn** to **SWCLKTCK**.

Each of these files implements the required logical behavior of the respective cell. You can use these models for simulation.

You must use these files as part of the validated processor RTL during logical equivalence checking as part of the sign-off procedure.

You must not use these files for synthesis.

You must implement an equivalent module that instantiates cells with required properties from your cell library for synthesis. You must also ensure that the cells you instantiate are maintained throughout the implementation flow and are not re-synthesized to alternative cells.

Example modules containing cell instantiations required for implementation are provided with the Reference Methodology. See the Reference Methodology release note for the location of these files.

Table 2-2 shows the module logical behavior and required DAP properties for correct CDC and glitch safe operation.

Table 2-2 DAP properties

Module	Parameter	Description
cm4_clk_gate, cm4_etm_clk_gate	CLKGATE_PRESENT	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 **SE** to the bypass input of the architectural clock gate to ensure that implementation inserted scan chains can be clocked when in shift mode.
cm4_sync	TRACE_LVL > 0	Logically, this module is a two-flop synchronizer used to sample signals asynchronous to the reference clock. This module has an asynchronous reset. You must choose flops designed to minimize the time required to resolve metastable outputs when the input setup or hold constraints are violated.
logical/cm4_dap_ahb_ap/verilog/cm4_dap_ahb_ap_sync.v logical/dapswjdp/verilog/DAPDpSync.v	DEBUG_LVL > 0	Logically, these modules are two-flop synchronizers used to sample signals asynchronous to the reference clock These modules have an asynchronous reset. You must choose flops designed to minimize the time required to resolve metastable outputs when the input setup or hold constraints are violated.

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

Note

If you are not using the ETM, by setting the TRACE_LVL parameter to 0 or 1, then the *Integrated Clock Gate* (ICG) cell must not be instantiated in cm4_clk_gate.v. Instead, the RTL equivalent must be left in place. Otherwise synthesis tools might not be able to remove the instantiated cell, and it might be left floating in the netlist.

The same applies for cm4_clk_gate if architectural clock gating is implemented using the CLKGATE_PRESENT parameter but the cm4_clk_gate.v file instantiates the ICG cell. If you do not intend to use the ICG, it must not be present in the clock gate module.

2.3.3 ROM table edits

The information in this section only applies if you have configured the Cortex-M4 processor to include debug. If DEBUG_LVL is set to 0, the ROM table module is not included and the ROM table addresses read as zero.

This ROM table is fixed in the processor memory map at 0xE00FF000. You must set up this ROM table as the single master ROM table in your system, or reference it by another valid CoreSight ROM table in your system.

Table 2-3 shows the ROM table format. See the *ARM Debug Interface v5 Architecture Specification* for more information about the ROM table format.

Table 2-3 ROM table format

Address	Value	Name	Description
0xE00FF000	0xFFF0F003	NVIC	Points to the NVIC at 0xE000E000.
0xE00FF004	0xFFF02002 or 0xFFF02003 if present	DWT	Points to the Data Watchpoint and Trace block at 0xE0001000. Value has bit 0 set to 1 if DWT is fitted.
0xE00FF008	0xFFF03002 or 0xFFF03003 if present	FPB	Points to the Flash Patch and Breakpoint block at 0xE0002000. Value has bit 0 set to 1 if FPB is fitted.
0xE00FF00C	0xFFF01002 or 0xFFF01003 if present	ITM	Points to the Instrumentation Trace block at 0xE0000000.
0xE00FF010	0xFFF41002 or 0xFFF41003 if present	TPIU	Points to the TPIU. Value has bit 0 set to 1 if TPIU is fitted. Value has bit 0 set to 1 if ITM is fitted. TPIU is at 0xE0040000.
0xE00FF014	0xFFF42002 or 0xFFF42003 if present	ETM	Points to the ETM. Value has bit 0 set to 1 if ETM is fitted. ETM is at 0xE0041000.
0xE00FF018	0	end	Marks the end of the ROM table. If CoreSight components are added, this appears at a higher address, after the CoreSight component addresses.
0xE00FFFCC	0x1	MEMTYPE	-
0xE00FFFD0	0x4	PID4[a]	-
0xE00FFFD4	0	PID5	-
0xE00FFFD8	0	PID6	-
0xE00FFFDC	0	PID7	-
0xE00FFFE0	0xC4	PID0[a]	-
0xE00FFFE4	0xB4	PID1[a]	-
0xE00FFFE8	0x0B	PID2[a]	-
0xE00FFFEC	0x00	PID3[a]	-

Table 2-3 ROM table format (continued)

Address	Value	Name	Description
0xE00FFFF0	0x0D	CID0[b]	-
0xE00FFFF4	0x10	CID1[b]	-
0xE00FFFF8	0x05	CID2[b]	-
0xE00FFFFC	0xB1	CID3[b]	-

a. PID0-4 have default values identifying this as a generic ROM table for the Cortex-M4 processor. ARM advises you to modify these registers to identify the manufacturer and part number of the device. The modifications must be in the format indicated in the *Coresight Architecture Specification*, These modifications are not required if an additional system ROM table already performs this function.
b. CID values identify the component as a CoreSight ROM table, and must not be modified.

To make any required edits to the ROM table:

- Altering the TRACE_LVL and DEBUG_LVL parameters automatically updates the ROM table and no more edits are required for these purposes.
- If any additional CoreSight components are added into your system, the offset values for these components must be added starting at ROM Table offset 0x018. The next entry after the additional components must be set to 0x0 to indicate the end of the table.

The ROM Table is at:

logical/cortexm4_integration/verilog/cm4_rom_table.v

Chapter 3
Key Integration Points

This chapter describes the key integration points you must consider when you integrate the processor with 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 processor with 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 xiii.

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

3.2 Key CORTEXM4 integration tasks

Table 3-1 lists key tasks for CORTEXM4 integration.

Table 3-1 Key CORTEXM4 integration tasks

Key task		Description
1.	Connect the **HCLK**, **FCLK**, and **DAPCLK** clocks correctly.	See *Clocks* on page 4-3
2.	Connect **PORESETn**, **SYSRESETn,** and **DAPRESETn** correctly.	See *Resets* on page 4-5
3.	Tie off or connect the following interface inputs appropriately: • external AHB-Lite interfaces • AHB-Lite interface extensions • interrupt interface • Serial Wire or JTAG interface • Debug slave interface • miscellaneous signals. • SysTick signals • WIC interface.	See Chapter 4 *Functional Integration Guidelines*
4.	Set the master defines for FPU present or not. The file `cm4_lic_defs.v` contains the following line: `define `ARM_CM4_FPU_LICENSE` If the FPU is not licensed, comment out the appropriate line.	
5.	Use the integration kit to verify your design.	See Chapter 5 *Integration Kit*.

3.3 Key CORTEXM4INTEGRATION integration tasks

Table 3-2 lists key tasks for CORTEXM4INTEGRATION integration.

Table 3-2 Key CORTEXM4INTEGRATION integration tasks

Key task		Description
1.	Connect the following clocks: **HCLK** **FCLK** **DAPCLK** **TRACECLKIN** **SWCLKTCK**.	See *Clocks* on page 4-3
2.	Connect **PORESETn**, **SYSRESETn**, and **nTRST** correctly.	See *Resets* on page 4-5
3.	Tie off or connect the following interface inputs appropriately: • external AHB-Lite interfaces • AHB interface extensions • interrupt interface • Serial Wire or JTAG interface • Debug slave interface • miscellaneous signals. • SysTick signals • WIC interface.	See Chapter 4 *Functional Integration Guidelines*
4.	Set the master defines for ETM and FPU present or not. The file cm4_lic_defs.v contains the following lines: `define `ARM_CM4_ETM_LICENSE` `define `ARM_CM4_FPU_LICENSE` If either is not licensed, comment out the appropriate line or lines.	-
5.	Set the parameters of CORTEXM4INTEGRATION to the required value or override them when instantiating the module.	See *Configuration options* on page 2-3 for more information.
6.	Use the integration kit to verify your design.	See Chapter 5 *Integration Kit*.

Chapter 4
Functional Integration Guidelines

This chapter describes the guidelines for functional integration of the processor in 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 processor clocks in your SoC design in:

- *CORTEXM4 level clocks*
- *CORTEXM4INTEGRATION level clocks* on page 4-4.

Figure 4-1 shows the Cortex-M4 processor clock domains.

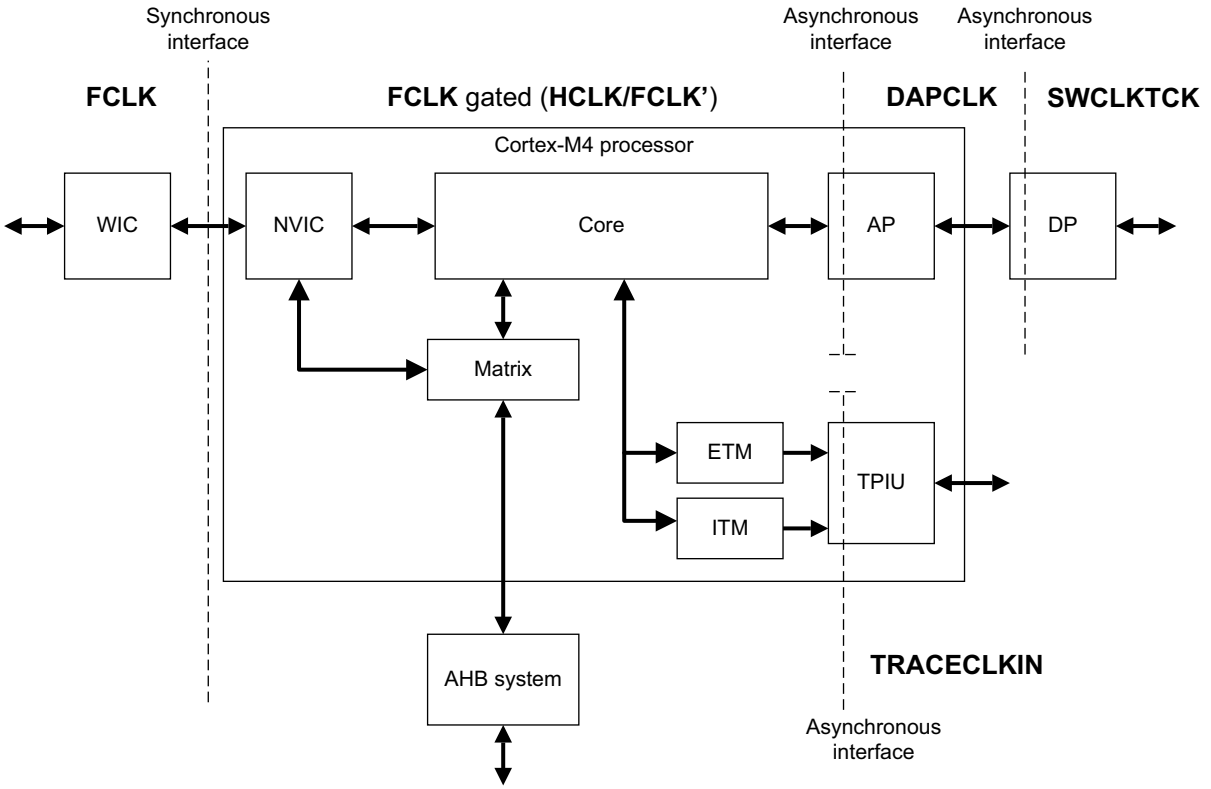


Figure 4-1 Cortex-M4 clock domains

4.2.1 CORTEXM4 level clocks

Table 4-1 shows the clocks at the CORTEXM4 level of hierarchy.

Table 4-1 Clocks at the CORTEXM4 level

Name	Direction	Description	Connection information
HCLK	Input	Main Cortex-M4 clock.	**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 **SLEEPING** to determine when the processor is sleeping.
FCLK	Input	Free running Cortex-M4 clock.	**FCLK** 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 **FCLK** at reduced frequency
DAPCLK	Input	AHB-AP clock.	Can be asynchronous to **FCLK** and **HCLK**.

4.2.2 CORTEXM4INTEGRATION level clocks

Table 4-2 shows the clocks at the CORTEXM4INTEGRATION level of hierarchy.

Table 4-2 Clocks at the CORTEXM4INTEGRATION level

Name	Direction	Description	Connection information
HCLK	Input	Main Cortex-M4 clock.	**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**. If **HCLK** is gated, the clock to the ETM and TPIU must also be gated.
FCLK	Input	Free running Cortex-M4 clock.	**FCLK** must always be running.
SWCLKTCK	Input	Serial wire clock and JTAG Test clock.	Can be asynchronous to **FCLK** and **HCLK**.
TRACECLKIN	Input	Trace port output clock.	Can be asynchronous to **FCLK** and **HCLK**.

4.3 Resets

This section describes considerations for connecting and using the Cortex-M4 processor resets.

4.3.1 CORTEXM4 level resets

Table 4-3 shows the resets at the CORTEXM4 level of hierarchy.

Table 4-3 CORTEXM4 level resets

Name	Direction	Description	Connection information
PORESETn	Input	Power-on-reset. Resets entire Cortex-M4 system.	Deassert the reset synchronously to **FCLK**. Assert the reset on power-on. Assert the reset for at least three **FCLK** cycles.
SYSRESETn	Input	The **SYSRESETn** signal resets the processor excluding debug logic in the NVIC, FPB, DWT, and ITM.	
DAPRESETn	Input	AHB-AP reset.	Deassert **DAPRESETn** synchronously to **DAPCLK**. Assert **DAPRESETn** on power-on. Assert **DAPRESETn** for at least three **DAPCLK** cycles.

4.3.2 CORTEXM4INTEGRATION level resets

Table 4-4 shows the resets at the CORTEXM4INTEGRATION level of hierarchy.

Table 4-4 CORTEXM4INTEGRATION level resets

Name	Direction	Description	Connection information
PORESETn	Input	Power-on-reset. Resets entire Cortex-M4 system.	Deassert the reset synchronously to **FCLK**. Assert the reset on power-on. Assert the reset for at least three **FCLK** cycles.
SYSRESETn	Input	The **SYSRESETn** signal resets the processor excluding debug logic in the NVIC, FPB, DWT, and ITM.	
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 be synchronized to **SWCLKTCK**. Tie **nTRST** LOW if JTAG is not configured.

4.3.3 Reset modes

The reset signals present in the processor design enable you to reset different parts of the design independently. Table 4-5 shows the reset signals, and the combinations and possible applications that you can use them in.

Table 4-5 Reset modes

Reset mode	SYSRESETn	nTRST	DAPRESETn	PORESETn	Application
Power-on reset	0	0	0	0	Reset at power on, full system reset. Cold reset.
Processor reset	0	x	x	1	Reset of processor core only, excluding debug logic. Warm reset.
SWJ-DP reset	1	0	1	0	Reset of SWJ-DP and processor including debug logic
SW-DP reset	1	x	1	0	Reset of SW-DP and processor including debug logic
AHB-AP reset	1	1	0	1	Reset of AHB-AP logic

Note

PORESETn resets a superset of the **SYSRESETn** logic.

Power-on reset

ARM recommends that you assert the reset signals for at least three **FCLK** cycles to ensure correct reset behavior.

Processor reset

A processor or *warm* reset initializes the majority of the processor, excluding NVIC debug logic, FPB, DWT, and ITM. This reset is typically used for resetting a processor that has been operating for some time, for example, watchdog reset. A processor reset is achieved using **SYSRESETn**.

SWJ-DP reset

nTRST reset initializes the state of the SWJ-DP TAP controller. This reset is typically used by the RealView® ICE module for hot-plug connection of a debugger to a system. It initializes the SWJ-DP controller without affecting the normal operation of the processor.

Because the **nTRST** signal is not synchronized within the processor, it must only be deasserted synchronously to **SWCLKTCK**. This is typically handled by the RealView ICE module.

SW-DP reset

PORESETn reset initializes the SW-DP and the AHB-AP logic.

Normal operation

During normal operation the resets, **PORESETn**, **SYSRESETn** and **DAPRESETn** are deasserted. If the SWJ-DP or SW-DP ports are not in use, **nTRST** must be tied to 0 or 1.

Scan operation

During scan testing, the reset synchronizers and **SYSRESETn** are bypassed by the assertion of the **RSTBYPASS** input. **CGBYPASS** is also used to bypass clock gating cells during scan testing.

4.4 Interfaces

This section describes the interfaces of the Cortex-M4 processor in:

- *External AHB-Lite bus interfaces*
- *APB interface* on page 4-23
- *FPU signals* on page 4-25
- *Miscellaneous signals* on page 4-26
- *WIC interface* on page 4-27
- *Power Management Unit interface* on page 4-29
- *Debug access* on page 4-30
- *Embedded Trace Macrocell* on page 4-32
- *Trace interface* on page 4-35
- *HTM interface* on page 4-36
- *SWJ-DP interface* on page 4-36
- *SW-DP Interface* on page 4-37
- *Trace port interface* on page 4-38.

4.4.1 External AHB-Lite bus interfaces

The processor drives the AHB-Lite bus interface from the processor core. The processor has the following AHB-Lite bus ports:

- ICode port
- DCode port
- System port.

Ensure you understand the AHB-Lite bus interface signals, described in the *AMBA AHB-Lite Protocol Specification*, before connecting any of the signals described here. Table 4-6 on page 4-9 to Table 4-8 on page 4-12 show the AHB-Lite buses present on the processor, and describe how to connect the signals in your SoC design.

The processor matches the AMBA 3 specification except for maintaining control information during waited transfers. The AMBA 3 AHB-Lite Protocol states that when the slave is requesting wait states the master must not change the transfer type, except for the following cases:

- On an IDLE transfer, the master can change the transfer type from IDLE to NONSEQ.
- On a BUSY transfer with a fixed length burst, the master can change the transfer type from BUSY to SEQ.
- On a BUSY transfer with an undefined length burst, the master can change the transfer type from BUSY to any other transfer type.

The processor does not match this definition because it might change the access type from SEQ or NONSEQ to IDLE during a waited transfer. The processor might also change the address or other control information and therefore request an access to a new location. The original address that was retracted might not be requested again.This cancels the outstanding transfer that has not occurred because the previous access is wait-stated and awaiting completion. This is done so that the processor can have a lower interrupt latency and higher performance in wait-stated systems by retracting accesses that are no longer required.

To achieve complete compliance with the AMBA 3 specification you can set the AHB_CONST_CTRL parameter to 1. This ensures that once transfers are issued during a wait-stated response they are never retracted or modified and the original transfer is honoured.

The consequences of setting this parameter are that there might be a slight area increase in the implemented design and the performance of the processor might decrease for wait-stated systems because of increases in interrupt and branch latency.

The Cortex-M4 processor does not register the bus interfaces. Excessive loads on the bus outputs, or slow timings on the bus inputs, directly impact the performance of the processor. You must therefore ensure that the loads and timings on these interfaces meet those specified by your Cortex-M4 implementer.

Table 4-6 shows the signals for the ICode Interface.

Table 4-6 ICode interface signals

Name	Direction	Description	Connection information
HADDRI[31:0]	Output	32-bit instruction address bus.	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HTRANSI[1:0]	Output	Indicates the type of the current transfer, which can be: `0b00` = IDLE `0b10` = NONSEQUENTIAL.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HSIZEI[2:0]	Output	Indicates the size of the instruction fetch. All instruction fetches are 32 bits on Cortex-M4 processor. **HSIZEI** is always `0b010`.	Connect to the slaves through the bus infrastructure.
HBURSTI[2:0]	Output	Indicates if the transfer is part of a burst. All instruction fetches and literal loads are performed as SINGLE. **HBURSTI** is always `0b000`.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HPROTI[3:0]	Output	Provides information on the access. Always indicates cacheable and bufferable on this bus. **HPROTI[0]**: 0 = Instruction fetch 1 = Vector fetch. **HPROTI[1]**: 0 = Unprivileged 1 = Privileged. **HPROTI[2]**: Always 1 = Bufferable. **HPROTI[3]**: Always 1 = Cacheable.	Connect to the slaves through the bus infrastructure.
MEMATTRI[1:0]	Output	Memory attributes. Always `0b00` for this bus. They are nonallocate, nonshareable.	

Table 4-6 ICode interface signals (continued)

Name	Direction	Description	Connection information
BRCHSTAT[3:0]	Output	0b0000 = No hint 0b0001 = Conditional branch backwards in decode 0b0010 = Conditional branch in decode 0b0011 = Conditional branch in execute 0b0100 = Unconditional branch in decode 0b0101 = Unconditional branch in execute 0b0110 = Reserved 0b0111 = Reserved 0b1000 = Conditional branch in decode taken (cycle after **IHTRANS**) 0b1001...0b1111 - Reserved.	Can be used to optimize operation of memory controllers or prefetchers.
HRDATAI[31:0]	Input	Instruction read bus.	Connect from the slaves through AHB infrastructure.
HREADYI	Input	When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.	Connect to the slaves through the bus infrastructure.
HRESPI[1:0]	Input	The transfer response status. 0b00 = OKAY 0b01 = ERROR.	

Table 4-7 shows the signals for the DCode interface.

Table 4-7 DCode interface signals

Name	Direction	Description	Connection information
HADDRD[31:0]	Output	32-bit instruction address bus.	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HTRANSD[1:0]	Output	Indicates the type of the current transfer, which can be: 0b00 = IDLE 0b10 = NONSEQUENTIAL 0b11 = SEQUENTIAL.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HWRITED	Output	Write not Read.	Connect to the slaves through the bus infrastructure.
HSIZES[2:0]	Output	Indicates the size of the access. Accesses can be: 0b000=byte 0b001=halfword 0b010=word.	
HBURSTD[2:0]	Output	Indicates if the transfer is part of a burst.	Connect to the AHB arbiter and slaves through the bus infrastructure.

Table 4-7 DCode interface signals (continued)

Name	Direction	Description	Connection information
HMASTERD[1:0]	Output	Indicates the current DCode bus master: • 0 = Core data side accesses • 1 = DAP accesses • 2 = Core instruction side accesses. These include vector fetches that are marked as data by **HPROTD[0]**. This value cannot appear on **HMASTERD** • 3 = Reserved. This value cannot appear on **HMASTERD**.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HPROTD[3:0]	Output	Provides information on the access. **HPROTD[0]**: Always 1 = Data access **HPROTD[1]**: 0 = Unprivileged 1 = Privileged. **HPROTD[2]**: 0 = Non-bufferable 1 = Bufferable. **HPROTD[3]**: 0 = Non-cacheable 1 = Cacheable.	Connect to the slaves through the bus infrastructure.
MEMATTRD[1:0]	Output	Memory attributes.: Bit 0 = Allocate Bit 1 = Shareable.	
HWDATAD[31:0]	Output	Data write bus.	
HREADYD	Input	When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.	
HRESPD[1:0]	Input	The transfer response status. `0b00` = OKAY `0b01` = ERROR.	

Table 4-7 DCode interface signals (continued)

Name	Direction	Description	Connection information
HRDATAD[31:0]	Input	Read Data.	Connect from the slaves through AHB infrastructure.
EXREQD	Output	Exclusive Request. **EXREQD** is an address phase control signal that indicates if the access is because of a `LDREX` or `STREX`: 0 = No request 1 = Exclusive request. You can use **EXREQD** and **EXRESPD** to synchronize primitives and semaphores.	Connect to Exclusives Monitor.
EXRESPD	Input	Exclusive Response. **EXRESPD** is a data phase response like **HRESPD**, but is only valid for exclusive accesses and indicates the success or failure of an exclusive operation: 0 = Exclusive request accepted 1 = Exclusive request failed. You can use **EXREQD** and **EXRESPD** to synchronize primitives and semaphores.	Connect to Exclusives Monitor or tie LOW if not required.

Table 4-8 shows the signals for the system bus interface.

Table 4-8 System bus Interface signals

Name	Direction	Description	Connection information
HADDRS[31:0]	Output	32-bit address.	Connect to address decoders, arbiter, and slaves through the bus infrastructure.
HTRANSS[1:0]	Output	Indicates the type of the current transfer, which can be: `0b00` = IDLE `0b10` = NONSEQUENTIAL `0b11` = SEQUENTIAL.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HSIZES[2:0]	Output	Indicates the size of the access. Accesses can be: `0b000`=byte `0b001`=halfword `0b010`=word.	Connect to the slaves through the bus infrastructure.
HBURSTS[2:0]	Output	Indicates if the transfer is part of a burst.	Connect to the AHB arbiter and slaves through the bus infrastructure.

Table 4-8 System bus Interface signals (continued)

Name	Direction	Description	Connection information
HPROTS[3:0]	Output	Provides information on the access. **HPROTS[0]**: 0 = Instruction 1 = Data access. **HPROTS[1]**: 0 = Unprivileged 1 = Privileged. **HPROTS[2]**: 0 = Non-bufferable 1 = Bufferable. **HPROTS[3]**: 0 = Non-cacheable 1 = Cacheable.	Connect to the slaves through the bus infrastructure.
HMASTERS[1:0]	Output	Indicates the current system bus master: • 0 = Core data side accesses • 1 = DAP accesses. • 2 = Core instruction side accesses. These include vector fetches that are marked as data by **HPROTS[0]**. • 3 = Reserved. This value cannot appear on **HMASTERS**.	Connect to the AHB arbiter and slaves through the bus infrastructure.
HWDATAS[31:0]	Output	32-bit write data bus.	Connect to the slaves through the bus infrastructure.
HWRITES	Output	Write not Read.	Connect to the slaves through the bus infrastructure.
HMASTLOCKS	Output	Indicates a transaction that must be atomic on the bus. This is only used for bit-band writes, performed as read-modify-write.	Connect to arbiter.
MEMATTRS[1:0]	Output	Memory Attributes. Bit 0 = Allocate Bit 1 = Shareable.	Connect to the slaves through the bus infrastructure.
HRDATAS[31:0]	Input	Read data bus.	Connect from the slaves through AHB infrastructure.

Table 4-8 System bus Interface signals (continued)

Name	Direction	Description	Connection information
HREADYS	Input	When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.	Connect to the slaves through the bus infrastructure.
HRESPS[1:0]	Input	The transfer response status: `0b00` = OKAY `0b01` = ERROR.	
EXREQS	Output	Exclusive Request. **EXREQS** is an address phase control signal that indicates if the access is because of a `LDREX` or `STREX`: 0 = No request 1 = Exclusive request You can use **EXREQS** and **EXRESPS** to synchronize primitives and semaphores.	Connect to Exclusives Monitor.
EXRESPS	Input	Exclusive Response. **EXRESPS** is a data phase response like **HRESPS**, but is only valid for exclusive accesses and indicates the success or failure of an exclusive operation: 0 = Exclusive request accepted 1 = Exclusive request failed. You can use **EXREQS** and **EXRESPS** to synchronize primitives and semaphores.	Connect to Exclusives Monitor or tie LOW if not required.

AHB-Lite byte lane definition

The Cortex-M4 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 4-9 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 4-9 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]**
`0b00`	`0b00`	-	-	-	-	Rd[7:0]
`0b00`	`0b01`	-	-	-	Rd[7:0]	-
`0b00`	`0b10`	-	-	Rd[7:0]	-	-
`0b00`	`0b11`	-	Rd[7:0]	-	-	-
`0b01`	`0b00`	Little-endian	-	-	Rd[15:8]	Rd[7:0]
`0b01`	`0b00`	Big-endian	-	-	Rd[7:0]	Rd[15:8]

Table 4-9 AHB-Lite byte lane assignments (continued)

Address phase			Corresponding data phase			
HSIZE [1:0]	**HADDR [1:0]**	**Processor endianness**	**HxDATA [31:24]**	**HxDATA [23:16]**	**HxDATA [15:8]**	**HxDATA [7:0]**
0b01	0b10	Little-endian	Rd[15:8]	Rd[7:0]	-	-
0b01	0b10	Big-endian	Rd[7:0]	Rd[15:8]	-	-
0b10	0b00	Little-endian	Rd[31:24]	Rd[23:16]	Rd[15:8]	Rd[7:0]
0b10	0b00	Big-endian	Rd[7:0]	Rd[15:8]	Rd[23:16]	Rd[31:24]

Note

- Instruction fetches are always performed as 32-bit aligned word accesses.

AHB-Lite interface arbitration

ARM recommends that any external arbitration between the ICode and DCode AHB bus interfaces ensures that DCode has a higher priority than ICode.

For some systems you might want to combine the ICode and DCode buses in the processor core into a single, unified Code bus. To support this for high-speed operation, the processor has the **DNOTITRANS** input that suppresses the **HTRANSI** line when **HTRANSD** becomes active. With **DNOTITRANS** asserted, if **HTRANSI** and HTRANSD are to be active simultaneously in corresponding single-cycle address phases, then only **HTRANSD** is asserted. The ICode transaction is waited internal to the processor. In other words, the external ICode bus is forced into an idle state. The two **HTRANS** signals are therefore guaranteed never to be simultaneously active, which permits the bus multiplexer to be a very simple device.

An example multiplexor called `cm4_code_mux.v` which can be used to combine the DCode and ICode AHB-Lite interfaces can be found in `logical/cortexm4_integration/verilog`.

Note

DNOTITRANS is a static input that must be tied high to enforce this behavior. **DNOTITRANS** can be asserted in the integration level by defining **ARM_CODEMUX**.

The external ICode/DCode bus multiplexer can be integrated into a Cortex-M4 system as Figure 4-2 on page 4-16 shows.

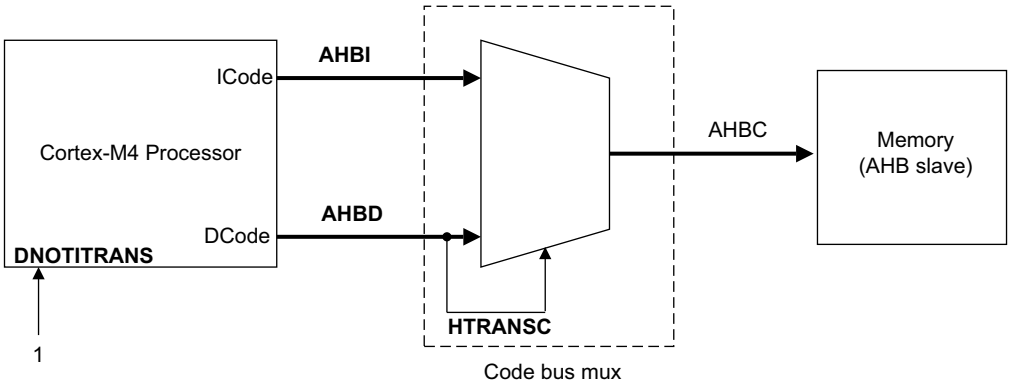


Figure 4-2 ICode/DCode multiplexer

AHB timing characteristics

The processor does not contain memories within the macrocell. To achieve high system performance, and to give the implementer complete flexibility in their memory architecture, memory requests from the processor are presented directly to the AHB interfaces unregistered.

Because of this, the Cortex-M4 AHB outputs are valid approximately 50% into the cycle, and the AHB inputs have a setup requirement of approximately 50% of the clock period. Excessive loads on the bus outputs, or slow timings on the bus inputs, have a direct impact on the performance of the processor.

Table 4-10 describes the timing characteristics of each of the interfaces.

Table 4-10 Interface timing characteristics

interface	Timing characteristics
ICODE	Instruction address and control signals are generated from the ALU, and as a result are valid approximately 50% into the clock cycle. Read data (**HRDATAI**) and read response (**HRESPI**) are presented directly to the processor and have approximately 50% of clock period setup.
DCODE	Core data and debug requests are presented over this bus. Both data and debug requests are presented relatively early in the cycle, and they are generated from registers with a small amount of combinatorial logic after the register. Requests on this bus have more slack than those presented on the ICODE bus. Write data (**HWDATAD**) is presented directly from the ALU and is valid approximately 50% into the clock cycle. Read data (**HRDATAD**) and read response (**HRESPD**) are presented directly to the processor and have approximately 50% of clock period setup.
SYSTEM	Instruction fetches from this bus are pipelined, and data and debug requests to this bus are presented early in the cycle, so requests on this bus have more slack than those presented on the ICODE bus. Write data (**HWDATAS**) is presented directly from the ALU and is valid approximately 50% into the clock cycle. Read data (**HRDATAS**) and read response (**HRESPS**) are presented directly to the processor and have approximately 50% of clock period setup.
PPB	Data and debug requests to this bus are presented early in the cycle, so requests on this bus have more slack than those presented on the ICODE bus. Write data (**PWDATA**) is presented directly from the ALU and is valid approximately 50% into the clock cycle. Read data (**PRDATA**) is presented directly to the processor and has approximately 50% of clock period setup

4.4.2 Branch target forwarding

The processor forwards certain branch types, by which the memory transaction of the branch is presented at least a cycle earlier than when the opcode reaches execute. Branch forwarding increases the performance of the core, because branches are a significant part of embedded controller applications. Branches affected are PC relative with immediate offset, or use LR as the target register. For conditional branches, by opcode definition or within IT block, that are forwarded, the address must be presented speculatively because the condition evaluation is an internal critical path.

Branch forwarding loses a fetch opportunity if speculated on a conditional opcode, but is mitigated by a three-entry fetch queue and a mix of 16/32-bit opcodes and single cycle ALU. The additional penalty is a cycle of pipeline stalling. The worst case is three 32-bit load/store single opcodes, the instructions word-unaligned, with no data waitstates. The **BRCHSTAT** interface provides information on forwarded branches to conditional execution, the direction if conditional, and a trailing registered evaluation of success of the preceding conditional opcode. For more information on **BRCHSTAT** see *Branch status interface*.

The performance of the core with ICODE registered with prefetch is effectively the same as the core without the branch forwarding interface, around 10% slower. Branch forwarding can be thought of as the internal address generation logic pre-registration to the address interface, increasing flexibility to the memory controller if you have the timing budget to make use of the information a cycle sooner. For example lower MHz power sensitive targets, in 0.13u down to 65nm. Otherwise, you have the flexibility of having access to this early address in your memory controller for lookups before registration to the system.

Branch speculation is more costly against a wait-stated memory because of mispredictions. To avoid this overhead, a rule in the controller that conditional branches are not speculated but instead registered gives subroutine calls and returns the benefits of branch forwarding without the mispredictions penalty. A refinement is to only predict backward conditional branches to accelerate loops. Alternatively, with ARM compilers favouring loops with unconditional branch backwards at the bottom and then conditional branch forward tests on the loop limit, the core fetch queue being ahead at the start of the loop yields good behavior.

The **BRCHSTAT** also includes other information about the next opcode to reach execute. Unlike the forwarded branches where **BRCHSTAT** is incident with the transaction, **BRCHSTAT** with respect to execute opcodes is a hint unrelated to any transaction and can be asserted for multiple cycles. The controller can use this information to suppress additional prefetching because it knows a branch is taken shortly. This helps to avoid any trailing waitstates of the controller prefetch from impacting the branch target when it is generated in execute.

4.4.3 Branch status interface

The processor speculatively fetches some branch targets at decode time rather than waiting for the branch to be executed. Because the processor has an internal pre-fetch buffer, incorrect speculative fetches might not incur a penalty depending on the instructions being executed and the number of wait-states on the memory. This means that branches for low wait state memories can be executed in one cycle. Certain implementations might find it beneficial to employ additional pre-fetching logic external to the processor to improve instruction throughput. This logic might be able to use **BRCHSTAT** to improve efficiency. The branch status signal, **BRCHSTAT**, gives information about the opcode currently in decode. **BRCHSTAT** indicates whether the current instruction in decode causes a non-sequential fetch to occur at decode time

or execute time, whether the branch is conditional and whether the branch is forward or backwards. Execute time branches might have multicycle **BRCHSTAT**, which is dependent on the stall of the preceding opcode in execute. Table 4-11 describes the signal function.

Table 4-11 Branch status signal function

Name	Direction	Description
BRCHSTAT	Output	`0b0000` = No branch currently in decode `0b0001` = Decode time conditional branch backwards currently in decode `0b0010` = Decode time conditional branch currently in decode `0b0011` = Execute time conditional branch currently in decode `0b0100` = Decode time unconditional branch currently in decode `0b0101` = Execute time unconditional branch in decode `0b0110` = Reserved `0b0111` = Reserved `0b1000` = Conditional branch in decode taken, cycle after `0b0001` or `0b0010`

Table 4-12 shows the branches that the processor can execute. For each type of branch the stage in which the branch is evaluated is shown. For example, all branches with immediates are evaluated during decode. This means that when ever a branch immediate enters the decode stage the branch target address is issued on the AHB.

The ALU register based branches and LSU PC modifying instructions are recognized as conditional branches, `0b0011`, if they are present in IT blocks. Otherwise they are recognized as unconditional branches, `0b0101`.

Table 4-12 Branches and stages evaluated by the processor

Branch Instruction	Instruction size	Stage branch target is issued	Notes
`B <imm>`	16 bits	Decode	-
`B <imm>`	32 bits	Decode	-
`BL`	32 bits	Decode	If LR is not being written during decode.
`BLX LR`	16 bits	Decode	If LR is not being written during decode.
`BX LR`	16 bits	Decode	If LR is not being written during decode.
`MOV PC, LR`	16 bits	Decode	If LR is not being written during decode.
`ADD PC`	32 bits	Execute	-
`BLX`	16 bits	Execute	If LR is not the source register or if LR is being written during decode.
`BX`	16 bits	Execute	If LR is not the source register or if LR is being written during decode.
`CBZ, CBNZ`	16 bits	Execute	-
`ISB`	16 bits	Execute	-
`LDR PC`	32 bits	Execute	-
`LDM to PC`	32 bits	Execute	-

Table 4-12 Branches and stages evaluated by the processor (continued)

Branch Instruction	Instruction size	Stage branch target is issued	Notes
`MOV PC`	32 bits	Execute	If LR is not the source register or if LR is being written during decode and LR is the source register.
`POP {PC}`	16 bits	Execute	-
`POP {PC}`	32 bits	Execute	-
`TBB/TBH`	32 bits	Execute	-

Note

- The encoding `0b1000` is only asserted in the cycle after conditional decode branches if the branch is taken. This is a registered output, so could be used to drive a multiplexor of addresses in the memory controller.
- Multicycle LSU in the `0b0101` encoding suppresses fetches during execute because it is known that the unconditional branch is executed so sequential fetches are prevented.
- Encodings are present for the multicycle duration of the decode, not only when decode enable is asserted.

Speculative fetches might be canceled during wait states. This means that the fetch address might change to a new address while **HREADY** is LOW. See *External AHB-Lite bus interfaces* on page 4-8.

Figure 4-3 and Figure 4-4 on page 4-20 show a conditional branch backwards not taken and taken. The branch occurs speculatively in the decode phase of the opcode. The branch target is a halfword unaligned 16-bit opcode.

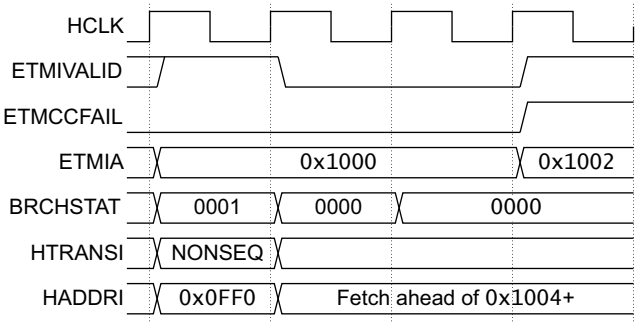


Figure 4-3 Conditional branch backwards not taken

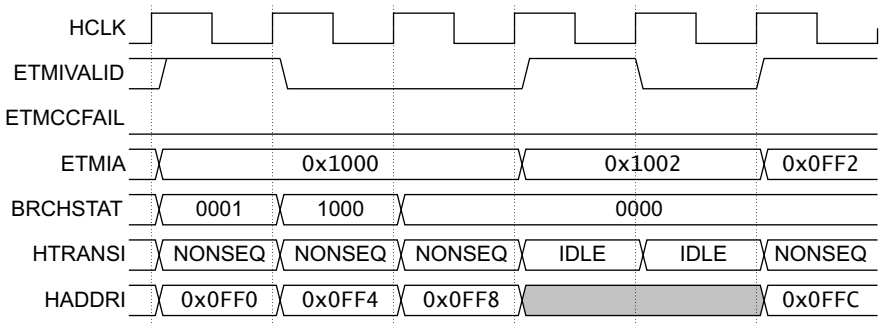


Figure 4-4 Conditional branch backwards taken

Figure 4-5 and Figure 4-6 show a conditional branch forwards not taken and taken. The branch occurs speculatively in the decode phase of the opcode. The branch target is a halfword aligned 16-bit opcode.

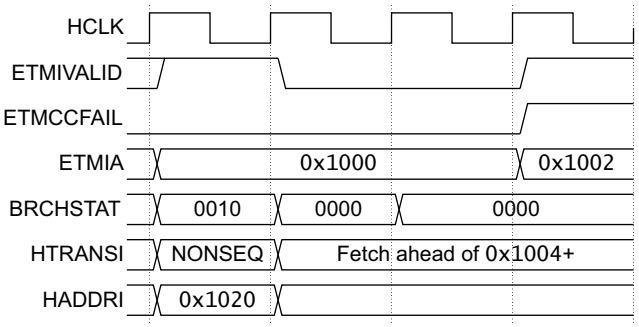


Figure 4-5 Conditional branch forwards not taken

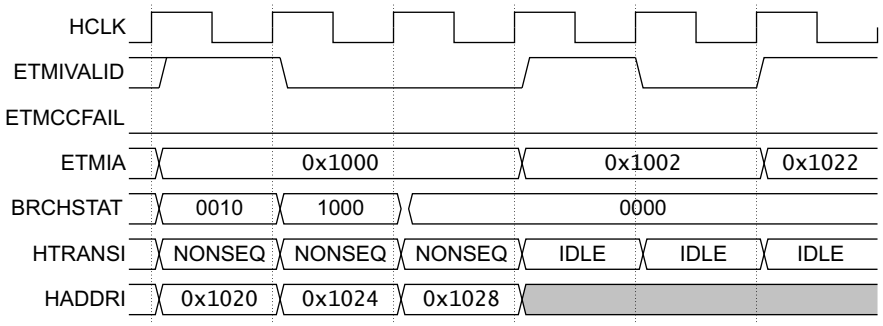


Figure 4-6 Conditional branch forwards taken

Figure 4-7 on page 4-21 and Figure 4-8 on page 4-21 show an unconditional branch in this cycle, during the execute phase of the preceding opcode without and with pipeline stalls. The branch occurs in the decode phase of the opcode. The branch target is an aligned 32-bit opcode.

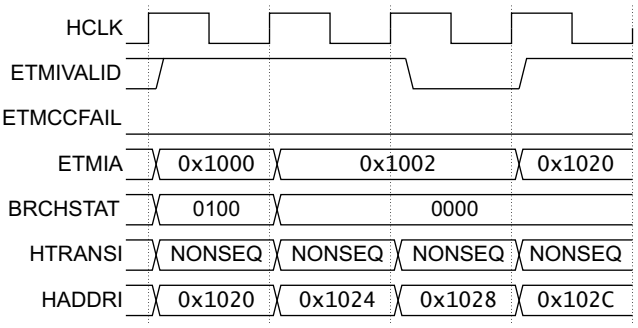


Figure 4-7 Unconditional branch without pipeline stalls

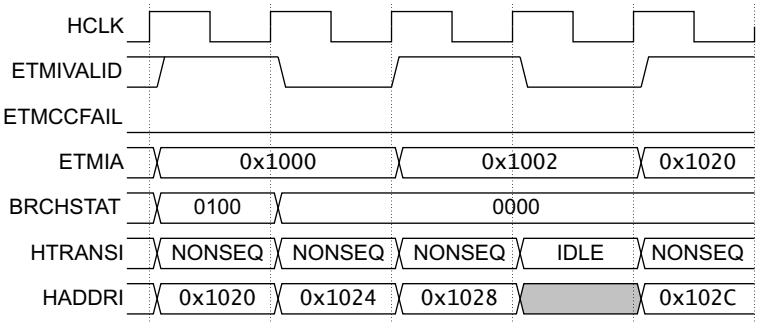


Figure 4-8 Unconditional branch with pipeline stalls

Figure 4-9 and Figure 4-10 show an unconditional branch in the next opcode. The branch occurs in the execute phase of the opcode. The branch target is an aligned and unaligned 32-bit ALU opcode.

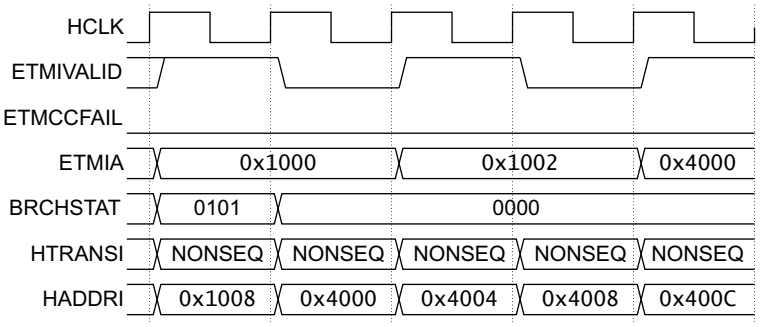


Figure 4-9 Unconditional branch in execute aligned

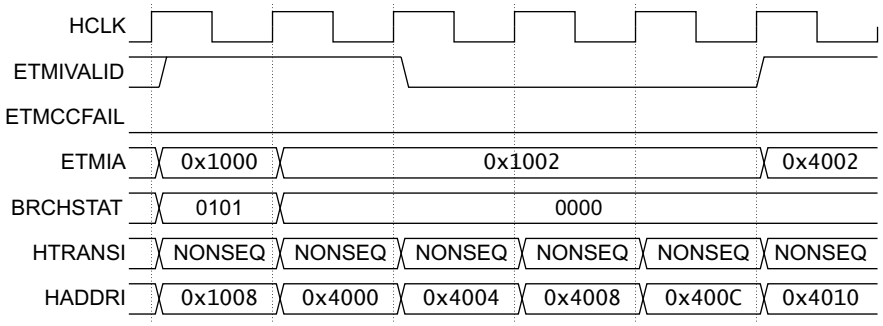


Figure 4-10 Unconditional branch in execute unaligned

Table 4-13 shows an example of an opcode sequence.

Table 4-13 Example of an opcode sequence

Execute cycle	Fetch address	Opcode
1	0x1020	ADD r1,#1
2	0x1022	LDR r3,[r4]
3	0x1024	ADD r2,#3
4	0x1026	CMP r3,r2
5	0x1028	BEQ = Target1
6	0x1040	CMP r1,r2
7	0x1042	ITE EQ // folded
8	0x1044	LDR EQ r3,[r4,r1] // skipped
9	0x1046	LDR NE r3,[r4,r2]
10	0x1048	ADD r6,r3
11	0x104A	NOP
12	0x104C	BX r14
13	0x0FC4	CMP
14	0x0FC6	BEQ = Target2 // not taken
15	0x0FC8	BX r5

Figure 4-11 on page 4-23 shows the timing sequence for the example opcode sequence in Table 4-13.

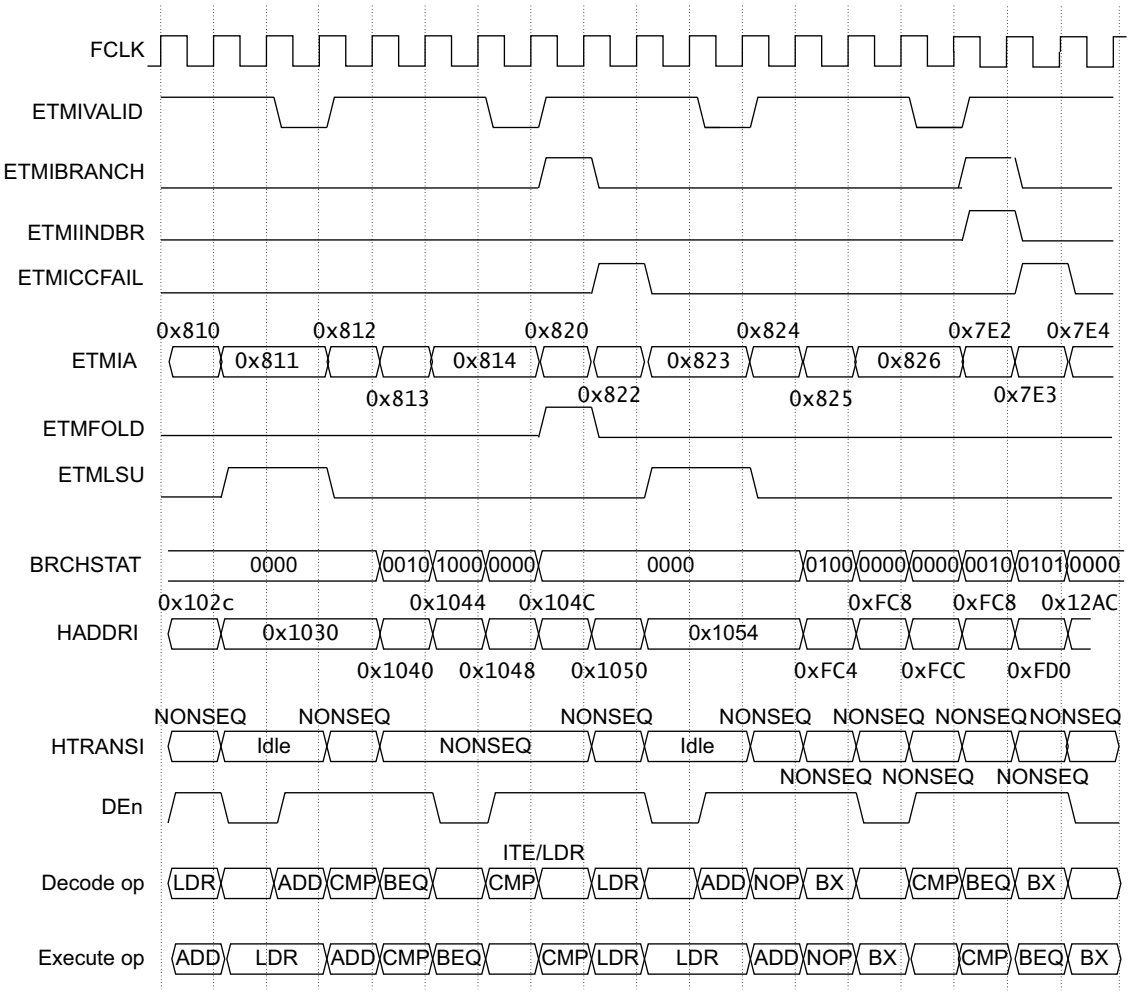


Figure 4-11 Example of an opcode sequence

4.4.4 APB interface

Table 4-14 shows the signals for the APB interface. This interface is to the External Private Peripheral Bus, and conforms to the AMBA v3.0 protocol. Only the address bits necessary to decode the External PPB space are supported on this interface.

Table 4-14 APB interface signals

Signal name	Direction	Description	Connection information
PSEL	Output	APB device select	Indicates that a data transfer is requested.
PENABLE	Output	APB control signal	Strobe to time all accesses. Used to indicate the second cycle of an APB transfer.
PADDR[19:2]	Output	APB address bus	18-bit address. Only the bits that are relevant to the External Private Peripheral Bus are driven.
PADDR[31]	Output	APB master	This signal is driven HIGH when the AHB-AP is the requesting master. It is driven LOW when the processor is the requesting master.
PWRITE	Output	APB transfer direction	Write not read.

Table 4-14 APB interface signals (continued)

Signal name	Direction	Description	Connection information
PWDATA[31:0]	Output	APB write data bus	32-bit write data bus.
PRDATA[31:0]	Input	APB read data bus	32-bit read data bus.
PREADY	Input	APB slave ready	This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
PSLVERR	Input	APB slave error	This signal is driven HIGH if the currently accessed APB device cannot handle the requested transfer.

4.4.5 Sleep interface

Table 4-15 shows the sleep interface signals present on the processor, and describes how you must set the signals in your SoC design.

Table 4-15 Sleep interface signals

Name	Direction	Description	Connection information
TXEV	Output	Event transmitted as a result of `SEV` instruction. This is a single-cycle pulse. You can use it to implement a more power efficient spin-lock in a multi-processor system.	Connected to other processors in a multi processor system. In a multi processor- system, **TXEV** from each processor can be broadcast to the **RXEV** input of the other processors.
RXEV	Input	Causes a wake-up from a `WFE` instruction[a].	Connect to **TXEVs** from other processors in a multi processor system. Tie LOW if not required.
SLEEPING	Output	Indicates that processor is in sleep mode.	
SLEEPDEEP	Output	Indicates that processor is in deep sleep mode.	
GATEHCLK	Output	**HCLK** can be gated because the processor is asleep without the debugger being active.	
SLEEPHOLDREQn	Input	Request to extend sleep. Can only be asserted when **SLEEPING** is HIGH.	
SLEEPHOLDACKn	Output	Acknowledge signal for **SLEEPHOLDREQn**.	

a. Input from OR'ing **TXEV** signals from other processors in the system. If different processors run at different frequencies then synchronizers must be used to guarantee that **TXEV** is synchronous to this processor. **TXEV** must also be a single-cycle pulse.

4.4.6 Interrupt interface

Table 4-16 shows the interrupt interface signals present on the processor, and describes how you must set the signals in your SoC design.

Table 4-16 Interrupt interface signals

Name	Direction	Description	Connection information
INTISR[239:0]	Input	External interrupt signals.	Connect to interrupt logic. The number of functional interrupt signals depends on your implementation.
INTNMI	Input	Non-maskable interrupt.	Connect to your interrupt logic.
CURRPRI[7:0]	Output	Indicates what priority interrupt, or base boost, is being used now. **CURRPRI** represents the pre-emption priority, and does not indicate secondary priority.	-
AUXFAULT[31:0]	Input	Auxiliary fault status information from the system.	Connect to fault status generating logic if required. Result appears in the Auxiliary Fault Status Register at address `0xE000ED3C`. A one-cycle pulse of information results in the information being stored in the corresponding bit until a write-clear occurs.
STKALIGNINIT	Input	Initial stack alignment.	Forms reset value of stack alignment. 0 ensures 4 byte alignment, and 1 ensures 8 byte alignment. You can alter Stack alignment after reset using bit [9] of the Configuration Control Register at address `0xE000ED14`.

The Cortex-M4 processor does not implement synchronizers for the **INTISR** and **INTNMI** inputs. If you want to use asynchronous interrupts, you must implement external synchronizers to reduce the possibility of metastability issues.

4.4.7 FPU signals

Table 4-17 shows the FPU signals.

Table 4-17 FPU signals

Name	Direction	Description	Connection information
FPIXC	Output	Inexact.	Cumulative exception flags from the *Floating Point Status and Control Register* (FPSCR). These signals indicate when a floating-point exception has occurred.
FPIDC	Output	Input denormal.	
FPOFC	Output	Overflow.	
FPUFC	Output	Underflow.	
FPDZC	Output	Divide-by-zero.	
FPIOC	Output	Invalid operation.	

4.4.8 Miscellaneous signals

Table 4-18 shows the miscellaneous signals present on the processor, and describes how you must set the signals in your SoC design. The configuration input signals are sampled at reset.

Table 4-18 Miscellaneous Interface signals

Name	Direction	Description	Connection information
SYSRESETREQ	Output	System reset request	Connect to reset controller.
LOCKUP	Output	Indicates that the core is locked up.	You can connect this signal to a Watchdog, for example, that resets the processor using **SYSRESETn**. It can also be connected to the ETM using one of the **EXTIN** ports.
BIGEND	Input	Static endianness setting. This signal is sampled at reset, and cannot be changed when reset is inactive. 1 = big-endian 0 = little-endian.	Hardwire to either 0 or 1 depending on the endianness of your system.
TRCENA	Output	Trace Enable. This signal reflects the setting of bit [24] of the Debug Exception and Monitor Control Register. This signal can be used to gate the clock to the TPIU and ETM blocks to reduce power consumption when trace is disabled.	Connect to clock gating logic for TPIU and ETM.
PPBLOCK[5:0]	Input	Reserved.	Must be tied to `0b000000`.
VECTADDR[9:0]	Input	Reserved.	Must be tied to `0b0000000000`.
VECTADDREN	Input	Reserved.	Must be tied LOW.
DNOTITRANS	Input	Suppression of parallel ICode and DCode transactions.	Static tie-off that forces the processor to not enable ICode and DCode AHB transactions to occur at the same time. This prevents instruction fetches occurring on the ICODE AHB at the same time as a data transaction is occurring on the DCODE AHB. See *AHB-Lite interface arbitration* on page 4-15
IFLUSH	Input	Instruction flush.	Must be tied LOW.
TSVALUEB	Input	Global timestamp value.	Connect to a natural binary count value if global timestamping is required. You must tie this signal to `0x000000000000` if not used. The count value can be derived from a different clock to **HCLK**, but must be cleanly synchronised to **HCLK**.

Table 4-18 Miscellaneous Interface signals (continued)

Name	Direction	Description	Connection information
TSCLKCHANGE	Input	Timestamp clock ratio change.	Pulse HIGH for one clock cycle if the ratio between **HCLK** and the clock being used to generate timestamps is altered. Tie LOW if **TSVALUEB** is not used, or if **TSVALUEB** is generated from HCLK.
MPUDISABLE	Input	MPU disable.	If asserted the MPU is invisible and unusable. Tie HIGH to disable the MPU. Tie LOW to enable the MPU, if present.
FPUDISABLE	Input	FPU disable.	If asserted the FPU is invisible and unusable. Tie HIGH to disable the FPU. Tie LOW to enable the FPU, if present.

4.4.9 WIC interface

The WIC interface is only functional if a WIC is configured. The interface signals only exist at the CORTEXM4 level.

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

Figure 4-12 shows the WIC mode enable sequence. It includes:

- the **WICENREQ** and **WICENACK** PMU interface signals that are present at the CORTEXM4INTEGRATION level only, see *Power Management Unit interface* on page 4-29 for more information.
- the **WICDSREQn** and **WICDSACKn** WIC signals that are present at the CORTEXM4 level only, see *WIC interface* for more information.

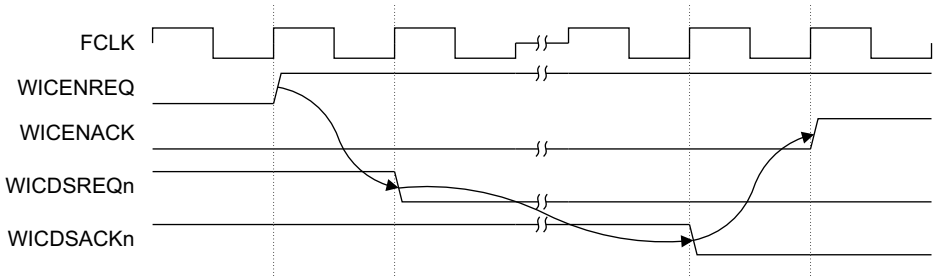


Figure 4-12 WIC mode enable sequence

Figure 4-13 on page 4-28 shows the power down timing sequence. It includes:

- the **WICLOAD**, **WICMASK[]**, **WICCLEAR**, **WICINT[]**, **WICPEND**, and **WAKEUP** WIC interface signals that are present at the CORTEXM4 level only, see *WIC interface* for more information.
- the **SLEEPING** and **SLEEPDEEP** miscellaneous interface signals that are present at the CORTEXM4 and CORTEXM4INTEGRATION level, see *Sleep interface* on page 4-24 for more information.
- the **ISOLATEn** and **RETAINn** signals that are present at the CORTEXM4INTEGRATION level are intended for use with state retention cells. They should be connected during synthesis.

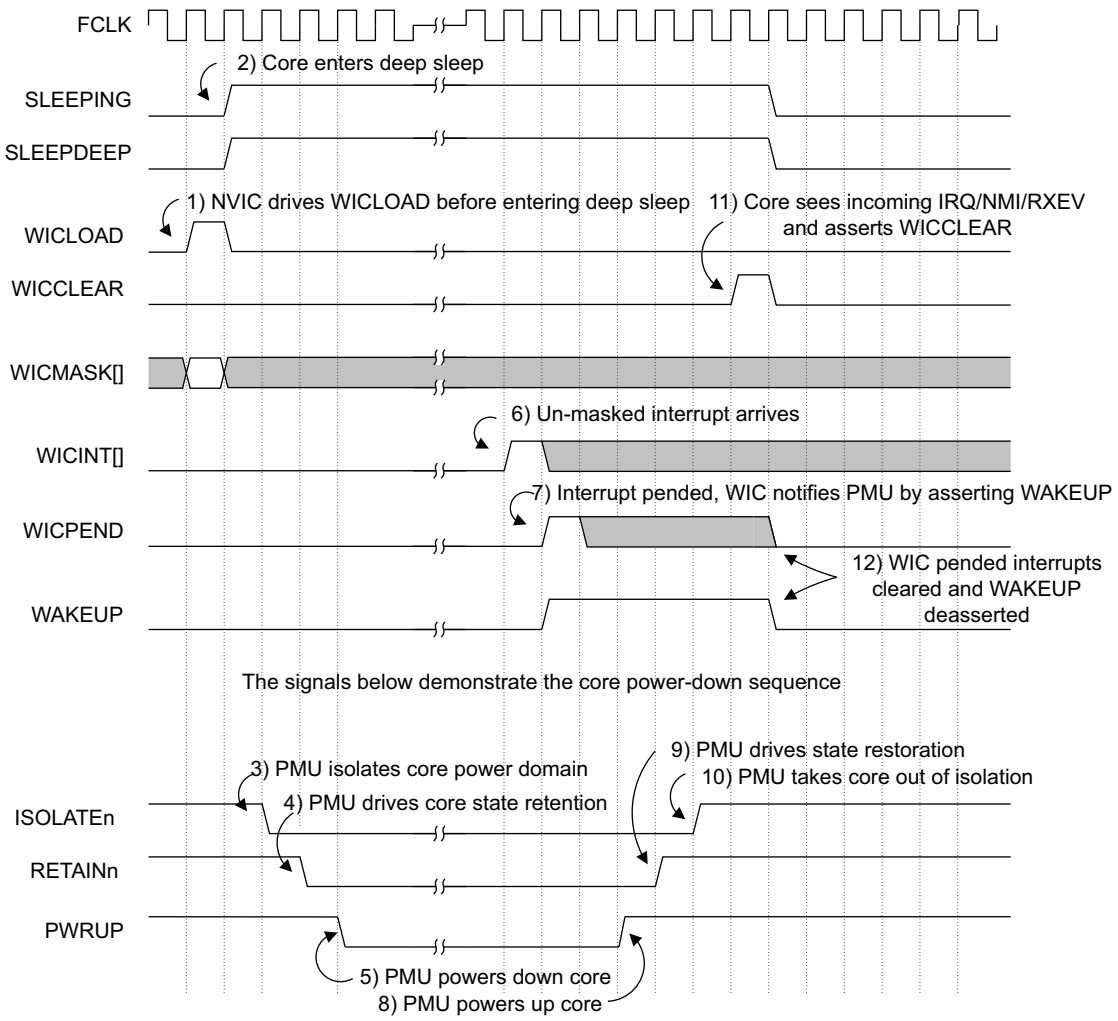


Figure 4-13 Power down timing sequence

Table 4-19 shows the WIC interface signals.

Table 4-19 WIC interface signals

Name	Direction	Description
ISOLATEn	Input	Used for power-down state retention implementations. Isolates the core power domain when 0. Tie HIGH if you are not using a state retention implementation.
RETAINn	Input	Used for power-down state retention implementations. Retains core state during power down when 0. Tie HIGH if you are not using a state retention implementation.
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	Output	Interrupt sensitivity mask used for WIC wake-up detection.
WICMASKNMI	Output	**NMI** sensitivity mask used for WIC wake-up detection.

Table 4-19 WIC interface signals (continued)

Name	Direction	Description
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.

4.4.10 Power Management Unit interface

The PMU interface only exists at the CORTEXM4 level.

Table 4-20 shows the PMU interface signals.

Table 4-20 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[a b]	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**. If no PMU is required then tie this signal high to enable the WIC if the WIC is implemented.
CDBGPWRUPREQ	Output	Active HIGH signal that indicates an external debugger request to the PMU to power-up the debug domain, if added in implementation, in readiness for a debugging session. This signal must be synchronized before use.
CDBGPWRUPACK	Input	Active HIGH signal that indicates the debug domain, if added in implementation, is powered up in response to **CDBGPWRUPREQ** being HIGH. This signal must be driven synchronously to **FCLK**.

a. Only functional if a WIC is configured.
b. Tie LOW if WIC interface is not configured.

CDBGPWRUPREQ and **CDBGPWRUPACK** form a four-phase handshake as defined in the *ARM Debug Interface v5 Architecture Specification*. 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** These signals must be connected regardless of their use in controlling a power domain.

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

Figure 4-12 on page 4-27 and Figure 4-13 on page 4-28 show how the **WAKEUP**, **WICSENSE**, **WICENREQ**, and **WICENACK** signals are sequenced.

4.4.11 Debug

Table 4-21 shows the debug signals.

Table 4-21 Debug signals

Name	Direction	Description	Connection information
HALTED	Output	In halting mode debug. **HALTED** remains asserted while the core is in debug.	Connect to ETM if present.
DBGRESTART	Input	External restart request. Multiprocessor debug support to connect to CoreSight Embedded Cross Trigger. If multiprocessor debug support is not required, **DBGRESTART** must be tied LOW.	Multiprocessing debug support. Connect to a CTI.
DBGRESTARTED	Output	Handshake for **DBGRESTART**.	
EDBGRQ	Input	External debug request. Combined debug request from ETM and multiprocessor debug support to connect to CoreSight Embedded Cross Trigger. If multiprocessor debug support is not required, **EDBGRQ** can be connected to **ETMDBGRQ** on ETM.	-
DBGEN	Input	External debug enable. If **DBGEN** is de-asserted then all debug features cannot be used. If **DBGEN** is asserted then debug features may be used but other enables such as **C_DEBUGEN** must still be set to allow debug events such as halt to occur.	Either tie HIGH or connect to a debug access controller if required.

If a CTI is present in the system, for example when multiple processors are present in the design, please contact ARM for further details of the correct interconnection and handshaking logic that is required. This applies both to systems containing multiple Cortex-M4 processors, and to systems implementing processors of different types.

4.4.12 Debug access

Table 4-22 shows the signals for the AHB-AP interface.

Table 4-22 AHB-AP interface signals

Name	Direction	Description	Connection information
DAPRDATA[31:0]	Output	The read bus is driven by the selected AHB-AP during read cycles, when **DAPWRITE** is LOW.	Connect to **DAPRDATA** port of SW-DP or SWJ-DP using a slave multiplexor.
DAPREADY	Output	The AHB-AP uses this signal to extend a DAP transfer.	Connect to **DAPREADY** port of SW-DP or SWJ-DP using a slave multiplexor.
DAPSLVERR	Output	The error response is because: • Master port produced an error response, or transfer not initiated because of **DBGEN** preventing a transfer. • Access to AP register not accepted after a **DAPABORT** operation.	Connect to **DAPSLVERR** port of SW-DP or SWJ-DP using a slave multiplexor.

Table 4-22 AHB-AP interface signals (continued)

Name	Direction	Description	Connection information
DAPCLKEN	Input	DAP clock enable (clock ratio control or power saving).	Connect to your clock controller. **CDBGPWRUPREQ** can be used to control this signal.
DAPEN	Input	AHB-AP enable.	Connect high when enabling debug accesses.
DAPADDR[31:0]	Input	DAP address bus.	Connect to **DAPADDR[31:0]** port of SW-DP or SWJ-DP.
DAPSEL	Input	Select signal generated from the DAP decoder to each AP. This signal indicates that the slave device is selected, and a data transfer is required. There is a **DAPSEL** signal for each slave. The decoder monitors the address bus and asserts the relevant **DAPSEL.**	Connect to DAP address decoder.
DAPENABLE	Input	This signal is used to indicate the second and subsequent cycles of a DAP transfer from DP to AHB-AP.	Connect to **DAPENABLE** port of SW-DP or SWJ-DP.
DAPWRITE	Input	When HIGH indicates a DAP write access from DP to AHB-AP. When LOW indicates a read access.	Connect to **DAPWRITE** port of SW-DP or SWJ-DP
DAPWDATA[31:0]	Input	The write bus is driven by the DP block during write cycles, when **DAPWRITE** is HIGH.	Connect to **DAPWDATA[31:0]** port of SW-DP or SWJ-DP
DAPABORT	Input	Aborts the current transfer. The AHB-AP returns **DAPREADY** HIGH without affecting the state of the transfer in progress in the AHB Master Port.	Connect to **DAPABORT** port of SW-DP or SWJ-DP
FIXHMASTERTYPE	Input	The AHB-AP can issue AHB transactions with a HMASTER value of either 1 (DAP) or 0 (core data side) depending on how the AHB-AP is configured using the MasterType bit in the AHB-AP Control and Status Word Register. **FIXHMASTERTYPE** can be used to prevent this if required. If it is tied to `0b1` then the HMASTER issued by the AHB-AP is always 1 (DAP) and it cannot pretend to be the core. If it is tied to `0b0` then HMASTER can be issued as either 0 or 1.	Tie off as required

4.4.13 SysTick signals

These signals are present at both the `CORTEXM4` and `CORTEXM4INTEGRATION` levels.

The ARMv7-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 **STCLK** and **STCALIB** signals are nonfunctional.

Software can configure the SysTick timer to select **FCLK** as its clock source, or an alternative clock source.

In the Cortex-M4 processor, the alternative clock source is based on **FCLK** gated by **STCLK**. If you want to use an asynchronous clock to generate **STCLK**, you must use a synchronizer circuit. Figure 4-14 shows an example asynchronous clock generating **STCLK**.

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

Table 4-23 shows the **STCALIB[25:0]** values required for the software developer using the SysTick calibration register in the Cortex-M4 NVIC memory map.

Table 4-23 SysTick signals

Name	Mapping	Description
STCALIB[25]	NOREF	Indicates that no alternative reference clock source has been integrated. Tie HIGH if **STCLK** has been tied off.
STCALIB[24]	SKEW	Tie this LOW if the system timer clock, the external reference clock, or **FCLK** as indicated by **STCALIB[25]**, can guarantee an exact multiple of 10ms. Otherwise, tie this signal HIGH.
STCALIB[23:0]	TENMS	Provides an integer value to compute a 10ms (100Hz) delay from either the reference clock, or **FCLK** if the reference clock is not implemented. For example, apply the value `0x07A11F` if no reference is implemented, and **FCLK** is 50MHz.

Figure 4-14 shows an example asynchronous clock generating **STCLK**.

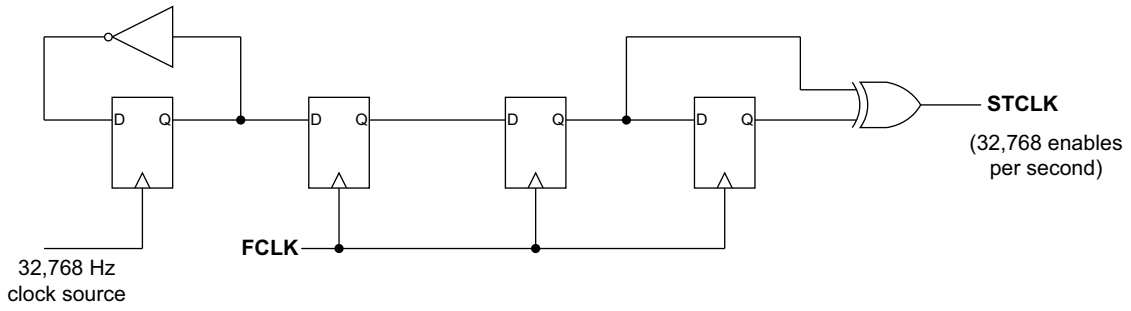


Figure 4-14 Asynchronous SysTick clock example

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

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

4.4.14 Embedded Trace Macrocell

If your SoC design does not include an ETM, you can leave the processor ETM interface outputs floating.

Note

This interface only exists at the `CORTEXM4` level and not the `CORTEXM4INTEGRATION` level.

Table 4-24 shows the signals as they appear on the ETM.

Table 4-24 ETM interface signals

Name	Direction	Description	Connection information
DWTMATCH[3:0]	Input	Trigger from DWT. One bit for each of the four DWT comparators.	Connect to **ETMTRIGGER[3:0]** of the processor.
DWTINOTD[3:0]	Input	Indicates if the ETM is triggered on an instruction or data match.	Connect to **ETMTRIGINOTD[3:0]** of the processor.
ETMIVALID	Input	Instruction valid.	Connect to **ETMIVALID** of the processor.
ETMIA[31:1]	Input	PC of the instruction being executed.	Connect to **ETMIA[31:1]** of the processor.
ETMICCFAIL	Input	Condition Code fail. Indicates if the current instruction has failed or passed its conditional execution check.	Connect to **ETMICCFAIL** of the processor.
ETMIBRANCH	Input	Opcode is a branch target.	Connect to **ETMIBRANCH** of the processor.
ETMIINDBR	Input	Opcode is an indirect branch target.	Connect to **ETMIBRANCH** of the processor.
ETMINTSTAT[2:0]	Input	Interrupt status. Marks interrupt status of current cycle: `0b000` - no status `0b001` - interrupt entry `0b010` - interrupt exit `0b011` - interrupt return `0b100` - vector fetch and stack push. **ETMINTSTAT** entry or return is asserted in the first cycle of the new interrupt context. Exit occurs without **ETMIVALID**.	Connect to **ETMINTSTAT[2:0]** of the processor.
ETMINTNUM[8:0]	Input	Marks the interrupt number of the current execution context.	Connect to **ETMINTNUM[8:0]** of the processor.
ETMISB	Input	Indicates execution of `ISB` instruction.	Connect to **ETMISB** of the processor.
ETMPWRUP	Output	ETM is enabled.	Connect to **ETMPWRUP** of the processor.
ETMDVALID	Input	Data valid.	Connect to **ETMDVALID** of the processor.
ETMCANCEL	Input	Instruction canceled.	Connect to **ETMCANCEL** of the processor.
ETMFINDBR	Input	Flush is indirect. Marks flush hint destination cannot be inferred from the PC.	Connect to **ETMFINDBR** of the processor.
ETMFOLD	Input	Opcode fold. An `IT` or `NOP` opcode has been folded in this cycle. PC advances past the current 16-bit opcode plus the IT/NOP instruction of 16 bits. This is reflected in the `ETMIA`.	Connect to **ETMFOLD** of the processor.
ETMISTALL	Input	Indicates that the last instruction signaled by the core has not yet entered execute.	Connect to **ETMISTALL** of the processor.

Table 4-24 ETM interface signals (continued)

Name	Direction	Description	Connection information
ETMFIFOFULL	Output	**ETMFIFOFULL** is asserted when the ETM FIFO reaches a watermark, and causes the processor to stall until the FIFO has drained, to ensure that no trace is lost. This functionality can be enabled using a register in the ETM.	Connect to **FIFOFULL** of the processor, or tie LOW at the processor if no ETM is present. To permanently disable this functionality, tie **FIFOFULL** of the processor LOW.
ETMTRIGOUT	Output	ETM trigger output.	Connect to **TRIGGER** on TPIU or to CTI if present.
ETMDBGRQ	Output	Debug request.	Connect to **EDBGRQ** of the processor, or OR with the debug request output form the CTI if present.
COREHALT	Input	Halt status.	Connect to **HALTED** of the processor, and to the CTI if present.
ETMPWRUP	Output	ETM power up.	Connect to **ETMPWRUP** of the processor.
ETMEN	Output	ETM port enable.	Use for power control of trace port, if required. See *Embedded Trace Macrocell Architecture Specification* for more information.

Table 4-25 shows the miscellaneous ETM connections.

Table 4-25 Miscellaneous ETM connections

Name	Direction	Description	Connection information
NIDEN	Input	**NIDEN** must be HIGH to enable the ETM to trace instructions.	Tie HIGH.
FIFOFULLEN	Input	Indicates support for **FIFOFULL** functionality by reading an ETM register.	Tie HIGH if **FIFOFULL** functionality is provided. Otherwise, tie LOW and also tie the **FIFOFULL** input to the processor LOW to permanently disable **FIFOFULL** functionality.
ETMEXTIN[1:0]	Input	General purpose inputs to the ETM from on-chip resources that can be used to generate triggers or to control trace regions.	Connect to **LOCKUP** from the processor, or any other on-chip resources. Tie unused bits LOW.
MAXEXTIN[1:0]	Input	Defines the number of **ETMEXTIN** signals that are connected.	Tie off to indicate the number of **ETMEXTIN** signals that are connected. Valid settings are `0b00`, `0b01`, `0b10`, indicating 0,1, or 2 **EXTIN** signals connected, respectively.

4.4.15 Trace interface

Table 4-26 and Table 4-27 show the signals for the ITM interface as they connect to the TPIU.

Table 4-26 ITM interface signals with no ETM present

Name	Direction	Description	Connection information
ATVALID1S	Input	ATB valid	Connect to **ATVALID** of the processor
AFREADY1S	Input	ATB flush	Connect to **AFREADY** of the processor
ATDATA1S[7:0]	Input	ATB data	Connect to **ATDATA[7:0]** of the processor
ATIDITM1S[6:0]	Input	ITM ID for TPIU	Connect to **ATIDITM[6:0]** of the processor
ATREADY1S	Output	ATB ready	Connect to **ATREADY** of the processor
TPIUACTV	Output	TPIU has data	Connect to **TPIUACTV** of the processor. Tie LOW at the processor if the Cortex-M4 TPIU is not present.
TPIUBAUD	Output	Unsynchronized baud indicator from TPIU	Connect to **TPIUBAUD** of the processor. Tie LOW at the processor if the Cortex-M4 TPIU is not present.
SYNCREQ	Input	Periodic synchronization packet trigger for formatted modes.	Connect to **DSYNC** of the processor
MAXPORTSIZE[1:0]	Input	Indicates the number of pins available for the TracePort mode.	Tie off as [pin count – 1] to indicate the size of the trace port available on the device
TSVALUEB	Input	Global timestamp value	Connect to a natural binary count value if global timestamping is required. You must tie this signal to `0x000000000000` if not used. The count value can be derived from a different clock to **HCLK**, but it must be cleanly synchronized to **HCLK**
TSCLKCHANGE	Input	Timestamp clock ratio change	Pulse high for one clock cycle if the ratio between CLK and the clock used to generate timestamps is altered. Tie LOW if **TSVALUEB** is not used, or if **TSVALUEB** is generated from **HCLK**.

Table 4-27 ITM interface signals with ETM present

Name	Direction	Description	Connection information
ATVALID1S	Input	ATB valid	Connect to **ATVALID** of the ETM
ATVALID2S	Input	ATB valid	Connect to **ATVALID** of the processor
AFREADY1S	Input	ATB flush	Connect to **AFREADY** of the ETM
AFREADY2S	Input	ATB flush	Connect to **AFREADY** of the processor
ATDATA1S[7:0]	Input	ATB data	Connect to **ATDATA[7:0]** of the ETM
ATDATA2S[7:0]	Input	ATB data	Connect to **ATDATA[7:0]** of the processor
ATID1S[6:0]	Input	ETM ID for TPIU	Connect to **ATIDITM[6:0]** of the ETM
ATID2S[6:0]	Input	ITM ID for TPIU	Connect to **ATIDITM[6:0]** of the processor

Table 4-27 ITM interface signals with ETM present (continued)

Name	Direction	Description	Connection information
ATREADY1S	Output	ATB ready	Connect to **ATREADY** of the ETM
ATREADY2S	Output	ATB ready	Connect to **ATREADY** of the processor
TPIUACTV	Output	TPIU has data	Connect to **TPIUACTV** of the processor. Tie to LOW at the processor if the Cortex-M4 TPIU is not present.
TPIUBAUD	Output	Unsynchronized baud indicator from TPIU	Connect to **TPIUBAUD** of the processor. Tie LOW at the processor if the Cortex-M4 TPIU is not present.
SYNCREQ	Input	Periodic synchronization packet trigger for formatted modes	Connect to **DSYNC** of the processor
MAXPORTSIZE[1:0]	Input	Indicates the number of pins available for the TracePort mode	Tie off to indicate the size of the trace port available on the device
TRIGGER	Input	Indicates a trigger packet in the trace stream	Connect to **ETMTRIGOUT** of the ETM if present, or tie to LOW if no ETM is present

4.4.16 HTM interface

Table 4-28 shows the signals for the HTM interface from the processor.

Table 4-28 HTM interface signals

Name	Direction	Description	Connection information
HTMDHADDR[31:0]	Output	32-bit address.	Connect to HTM if implemented.
HTMDHTRANS[1:0]	Output	Indicates the type of the current data transfer. Can be IDLE, NONSEQUENTIAL, or SEQUENTIAL.	Connect to HTM if implemented.
HTMDHSIZE[1:0]	Output	Indicates the size of the access. Can be 8, 16, or 32 bits.	Connect to HTM if implemented.
HTMDHBURST[2:0]	Output	Indicates if the transfer is part of a burst.	Connect to HTM if implemented.
HTMDHPROT[3:0]	Output	Provides information on the access.	Connect to HTM if implemented.
HTMDHWDATA[31:0]	Output	32-bit write data bus.	Connect to HTM if implemented.
HTMDHWRITE	Output	Write not read.	Connect to HTM if implemented.
HTMDHRDATA[31:0]	Output	Read data bus.	Connect to HTM if implemented.
HTMDHREADY	Output	Ready signal.	Connect to HTM if implemented.
HTMDHRESP[1:0]	Output	The transfer response status. Can be OKAY or ERROR.	Connect to HTM if implemented.

4.4.17 SWJ-DP interface

This section provides information on how to connect the JTAG and Serial Wire interface signals in your SoC design. This interface is only used if the processor has been implemented with SWJ-DP.

Table 4-29 shows the SWJ-DP interface signals present on the processor, and describes how you must set the signals if you intend to use JTAG debug support.

Note

See Table 4-30 for the Serial Wire interface description for the SWJ-DP.

Table 4-29 lists the debug interface signals.

Table 4-29 Debug interface signals

Name	Direction	Description	Connection information
nTRST	Input	Debug **nTRST**	See *Resets* on page 4-5.
TDI	Input	Debug **TDI**	For more information on how to connect these signals, see the *Debug Port* chapter of the *CoreSight Components Technical Reference Manual*.
SWDITMS	Input	Debug **TMS**	
TDO	Output	Debug **TDO**	
SWDO	Output	Serial Wire Data Out	
SWDOEN	Output	Serial Wire Output Enable	
SWCLKTCK	Input	Serial Wire Clock/TCK	
nTDOEN	Output	**DBGTDO** output pad control signal	This signal must be used to control the output enable on the pad for **DBGTDO**.

Multiplexing JTAG ports

There is no support in RealView-ICE for multiplexing **TCK**, **TMS**, **TDI**, **TDO**, and **RTCK**, between a number of different processors. ARM does not recommend multiplexing JTAG ports.

4.4.18 SW-DP Interface

Table 4-30 shows the SW-DP interface signals present on the processor, and describes how you must set the signals in your SoC design.

Table 4-30 SW-DP Interface signals

Name	Direction	Description	Connection information
SWDO	Output	Serial Wire Data Out	Connect to tristate pad for **SWDIO**
SWDOEN	Output	Serial Wire Output Enable	Connect to tristate pad for **SWDIO**
PORESETn	Input	Serial Wire Reset	Connect to reset controller
SWCLKTCK	Input	Serial Wire Clock	Connect to input pad for **SWCLKTCK**
SWDITMS	Input	Serial Wire Data In	Connect to tristate pad for **SWDIO**

4.4.19 Trace port interface

Table 4-31 lists the trace port interface signals.

Table 4-31 Trace port interface signals

Name	Direction	Description	Connection information
TRACECLKIN	Input	Trace Port Clock	Connect to your Trace Port Analyzer
TRACECLK	Output	Trace Clock	Connect to your Trace Port Analyzer
TRACEDATA[3:0]	Output	Output Trace Port Data Bus	Connect to your Trace Port Analyzer
TRACESWO	Output	Serial Wire Viewer data	Connect to your Trace Port Analyzer

Serial wire output connection

The Cortex-M4 TPIU provides a serial wire output mode that requires a single external pin. There are three options available to connect this pin:

- *A dedicated pin for TRACESWO*
- *SWO shared with TRACEPORT*
- *SWO Shared with JTAG-TDO* on page 4-39.

A dedicated pin for TRACESWO

This is the simplest option, but it requires an extra package pin. Figure 4-15 shows the dedicated pin option.

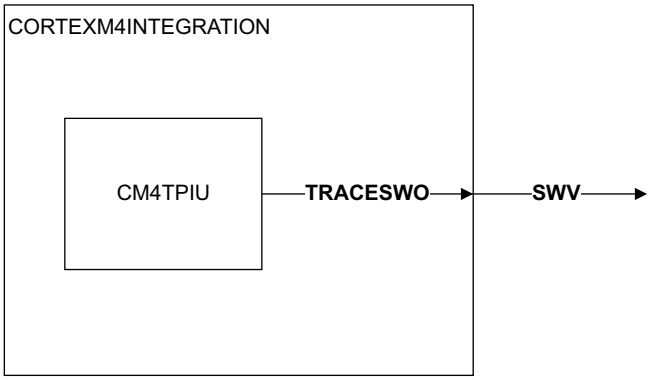


Figure 4-15 Dedicated pin used for TRACESWO

SWO shared with TRACEPORT

A pin can be shared between **TRACEDATA[0]** and **TRACESWO**. Because only one of these two pins can be in use at any one time, there is no loss of functionality using this option, and this is the preferred option when a dedicated trace port is present on the package.

To implement this option, the **SWOACTIVE** output from Cortex-M4 TPIU is used to control the multiplexor. Figure 4-16 on page 4-39 shows the SWO shared with TRACEPORT option.

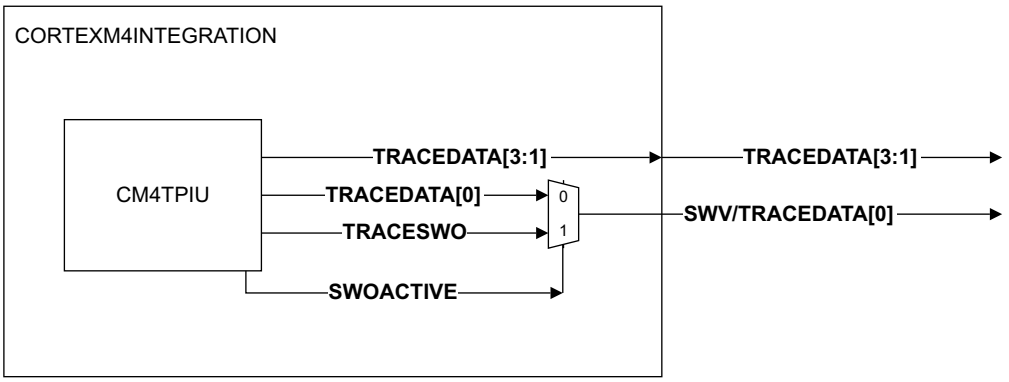


Figure 4-16 SWO shared with TRACEPORT

SWO Shared with JTAG-TDO

For minimal pin count, it is possible to overlay JTAG debug and SWO on the same package pin. This approach is only recommended where there is no provision for a conventional trace port, or for use with more complex system-level debug configuration controls.

If this option is chosen, the Instrumentation Trace is not accessible while the debug port is being used in a JTAG configuration. Serial wire debug and SWO can be used together at the same time.

To implement this option, the **JTAGNSW** output from SWJ-DP is used to control the multiplexor. Figure 4-17 shows the SWO shared with JTAG-TDO option.

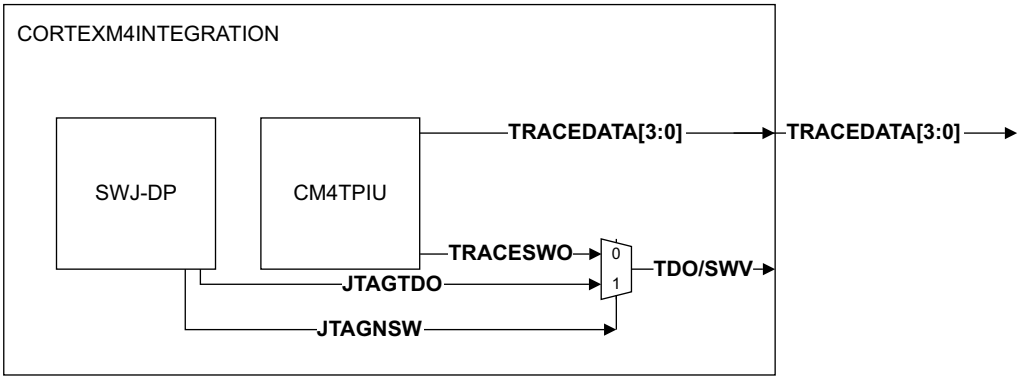


Figure 4-17 SWO shared with JTAG-TDO

4.5 Test interface

Table 4-32 shows the functions of the test interface pins.

Table 4-32 Test interface signals

Name	Direction	Description
SE	Input	**SE** is the scan-enable input. Set this signal to HIGH during vector scan-in and scan-out, and to LOW during normal operation.
RSTBYPASS	Input	Set to HIGH to bypass the internal reset synchronizers. This enables test pattern generation tools to have controllability of reset flops in the design.
CGBYPASS	Input	Set to HIGH to bypass the architectural clock-gate cell instantiations.

Table 4-33 shows the required test interface pins.

Table 4-33 Test interface pins

Level	Required pins
CORTEXM4	**SE**
CORTEXM4INTEGRATION	**SE, RSTBYPASS**.

Note

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

See Chapter 8 *Design For Test* for more information about testing and the testing interface.

Chapter 5
Integration Kit

This chapter describes the Cortex-M4 *Integration Kit* (IK). It contains the following sections:

5.1 About the integration kit

The Cortex-M4 integration kit is provided as a reference to enable easy integration of the Cortex-M4 processor into your system, and contains tests to check that you have performed this integration correctly. Support for simulation of both the Cortex-M4 RTL and Cortex-M4 netlists is provided. These simulations are performed at the `CORTEXM4INTEGRATION` level of hierarchy.

In addition, example vector capture testbenches are provided to enable you to capture vectors for silicon testing and netlist replay at the following levels:

- `CORTEXM4`
- `CORTEXM4INTEGRATION`
- `CM4ETM`, if present in your system.

The Cortex-M4 integration kit is based on a simple example microcontroller, CM4IKMCU, that instantiates

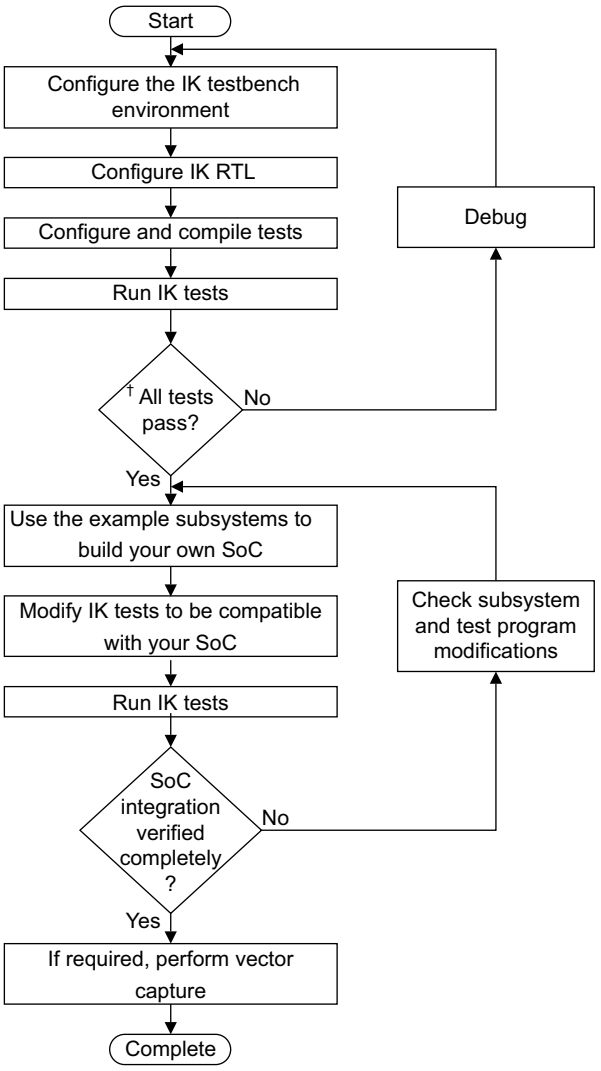
- the Cortex-M4 processor
- the integration level components (DWJ-DP, TPIU and optionally ETM)
- a ROM
- a RAM
- a PMU
- a WIC
- a reset controller
- some *General Purpose Input Output* (GPIO).

Note

The integration kit tests do not exhaustively test the Cortex-M4 processor and integration level I/O because not all I/O pins have an effect that is visible to program code, and some pins are meant to be static. This means that you must not use passing of all integration kit tests as the only sign-off criteria for successful processor integration. See Chapter 10 *Sign-off* for details of your Sign-Off obligations.

5.2 Cortex-M4 IK flow

The Cortex-M4 IK is intended as a platform from which you can develop a SoC incorporating the Cortex-M4 processor. Figure 5-1 shows the Cortex-M4 IK flow.



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

Figure 5-1 Cortex-M4 IK flow

5.3 Test overview

Table 5-1 shows the example test programs in the `integration_kit/validation/tests` directory.

Table 5-1 Example test programs

Test program	Description
`hellow`	The processor reads and checks its CPU ID and writes to the GPIO registers to print a simple message.
`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 are building tests using the `Makefile`, edit the value of ITERATIONS there. • If you are building tests using MDK-ARM, right-click **CM4IKMCU - Cortex-M4** within the dhry project to open the target options, then edit the value of ITERATIONS under the **C/C++** tab.
`max_power`	This test executes instructions that exercise the Cortex-M4 processor and maximize power consumption. Replay the captured test vectors on your netlist to get power values.
`speed_indicative`	This test executes instructions that exercise the critical paths in the Cortex-M4 processor. Replay the captured test vectors on your netlist to test timing accuracy.
`config_check`	This test verifies that the processor configuration matches the expected configuration values set in the `IKConfig.h` file.
`interrupt`	This test exercises **NMI**, **IRQ**, **TXEV**, and **RXEV**. Only the first 32 IRQ inputs can be tested with this test unless you modify the example system. The test is written to fail and to highlight this failure if more than 32 IRQ lines are present.
`interrupt_priority`	This test demonstrates the use of different interrupt priorities.
`reset`	This test checks the **SYSRESETREQ** output and that accesses to the AHB default slave cause a fault.
`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, and if the processor includes debug, the test also wakes the processor from sleep through the DAP. You can extend this test to verify state retention if that is implemented.
`debug`	This test is only useful for implementations that include a debug port. If you run this on an implementation without a debug port, the test passes without performing useful work. The test checks the pins **LOCKUP**, **EDBGRQ**, **HALTED**, **DEBUGRESTART**, and **DEBUGRESTARTED**.
`romtable`	This test checks that it is possible for a debugger to autodetect the Cortex-M4 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. This test is only useful for implementations that include a debug port. If you run this on an implementation without a debug port, the test passes without performing useful work.
`bus_matrix`	This test demonstrates execution of code from the system region. The test also demonstrates re-locating the vector table in the system region.
`fpu`	This test performs some simple floating point calculations and confirms that the FPU is present by checking the status register. This test relies on the C compiler to generate the relevant floating point instructions.

Table 5-1 Example test programs (continued)

Test program	Description
`mpu`	This test sets up a number of MPU regions and demonstrates the generation of faults from the MPU. **Note** This test is simple user code, so there is no recovery from faults, as would be required for an operating system that makes use of the MPU.
`itm_trace`	This test generates trace using the ITM and the Trace Port. Checking is limited to validating the protocol of the generated trace.
`itm_swo`	This test generates trace using the ITM and the SWV output. Checking is limited to validating the protocol of the generated trace.
`etm_trace`	This test generates trace using the ITM, ETM and the Trace Port. Checking is limited to validating the protocol of the generated trace.
`etm_swo`	This test generates trace using the ITM, ETM and the SWV output. ETM FIFO full stalling is enabled. Checking is limited to validating the protocol of the generated trace.
`flash_patch`	This test sets up a small piece of code in the system area and then uses the flash patch functionality to replace part of the main code with the code in the system area.
`ahb_compliance_check`	Sets up the conditions so that the Cortex-M4 exhibits the AHB non-compliance on the I-Code bus. For the non-compliance to be demonstrated, the testbench must have wait states enabled. See *Configuring the testbench* on page 5-6. The non-compliance can be observed by enabling the AHB protocol checking blocks in the integration kit.

5.3.1 Scope of tests

Due to the nature of the integration kit, some signals which will not always be connected in a system are not tested. You should consider which signals must be tested in your implementation, and extend the tests provided as described in *Modifying IK tests* on page 5-22 to cover these. Signals to note include:

- Interrupts. Only the first 32 interrupts can be tested with the integration kit out of the box. ARM suggests that all peripherals which generate an interrupt are used to provide full coverage of the implemented interrupt inputs.
- AHB control signals, for example **HPROTx**, **MEMATTRx**.
- FPU status outputs, for example **FPIXC**.
- HTM trace port.
- Writes from the DCODE bus.

5.4 Configuring the testbench

This section describes how to configure the testbench.

The testbench instantiates the CM4IKMCU level of the integration kit. See *Testbench structure* on page 5-18 for more information.

Table 5-2 shows the Verilog command files in the `integration_kit/logical/tbench/verilog/` directory.

Table 5-2 Verilog command files

File	Description
`tbench.vc`	Used for all simulations
`rtl.vc`	Used for RTL simulations
`netlist.vc`	Used for netlist simulations
`dsm.vc`	Used for DSM simulations

Note

- The Verilog command files used depend on the type of simulation required.
- The `tbench.vc` file contains defines for vector capture. For information on Verilog defines related to vector capture see *Test vector capture* on page 5-24.

The `ARM_CM4IK_INTEGRATION_LEVEL` define in the `integration_kit/logical/tbench/verilog/tbench.vc` file controls the integration level of the processor when performing netlist simulations. Ensure this define is included if you want to use the `CORTEXM4INTEGRATION` level. Ensure this define is not included if you want to use the `CORTEXM4` level. The RTL for the integration level will be used in the testbench if your netlist is for the CORTEXM4 level.

The `ARM_CM4IK_SRPG` define in the `integration_kit/logical/tbench/verilog/tbench.vc` file controls whether *State Retention Power Gating* (SRPG) signals are included in the processor. Ensure this define is included if you want SRPG signals. Ensure this define is not included if you do not want SRPG signals. You may need to modify the SRPG signals which are included, depending on your implementation flow.

The `ARM_SRPG_ON` define in the `integration_kit/logical/tbench/verilog/rtl.vc or integration_kit/logical/tbench/verilog/netlist.vc` file controls whether you use *Unified Power Format* (UPF) and *Common Power Format* (CPF) RTL simulations or SRPG netlist simulations respectively. Ensure this define is included if you want UPF or CPF in RTL simulations or SRPG in netlist simulations. Ensure this define is not included if you do not want UPF or CPF in RTL simulations or SRPG in netlist simulations.

The `ARM_CM4IK_SDF` define in the `integration_kit/logical/tbench/verilog/netlist.vc` file determines if delay calculation and static timing analysis information is in SDF. Ensure this define is included if you want SDF from netlist simulation. Ensure this define is not included if you want do not want SDF from netlist simulation.

The Cortex-M4 processor RTL and Cortex-M4 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. To run simulations with these checkers

enabled, you must uncomment the appropriate lines in the logical/tbench/verilog/tbench.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.

The integration kit also includes AHB protocol checkers which use the OVL library. You must uncomment the appropriate lines in the logical/tbench/verilog/tbench.vc file and the defines AHB_ASSERT_ON and ARM_CM4IK_AHBPC_ON. These highlight the cases when Cortex-M4 does not comply to the AMBA AHB-Lite specification when used with the ahb_compliance_check test.

To configure the testbench options, open the integration kit definitions file integration_kit/logical/tbench/verilog/cm4ik_defs.v. Table 5-3 shows the testbench options to configure in the definitions file.

Table 5-3 cm4ik_defs.v testbench options

option	Description
ARM_CM4IK_CLK_PERIOD	Clock for CM4IKMCU
ARM_CM4IK_POR_CYCLES	Power-on-Reset duration in cycles of CM4IKMCU clock
ARM_CM4IK_TIMEOUT_CYCLES	Runaway simulation time out duration in cycles of CM4IKMCU clock
ARM_CM4IK_RAM_WAIT_STATES	Introduces wait states in the memory peripherals. The default is 0 wait states. The valid range is 0-7 wait states.

5.5 Configuring the IK RTL

This section describes how to configure the IK RTL depending on your requirements. See *Configuration options* on page 2-3 for more information.

5.5.1 Configuring the system level

The instantiation of the timing wrapper and integration level in `cm4ik_sys.v` provides a mechanism to override the parameters that determine the configuration of the processor.

Modify the file `integration_kit/logical/cm4ikmcu/verilog/cm4ik_sys.v` to ensure that the instantiation matches your requirements.

5.5.2 Setting the implementation pin options

Implementation pin options can be set from the cm4ik_sys level. To set the implementation pin options, edit the integration kit definitions file `integration_kit/logical/tbench/verilog/cm4ik_defs.v`. The implementation pin options are:

ARM_CM4IK_STCALIB

SysTick calibration, see Figure 4-14 on page 4-32 for an asynchronous SysTick clock example.

5.6 Configuring and compiling tests

This section describes how to set up the test programs before running them on hardware or running a testbench simulation.

The supplied tests are written in C and must be compiled before you execute them on the Cortex-M4 processor.

When compiling C code, you must specify the correct CPU target to ensure that VFP instructions are not generated for an implementation which does not include the FPU. When the FPU is present, you must configure your toolchain appropriately, for example:

- Set the macro `COMPILE_FPU` to 1 in the `integration_kit/validation/tests/Makefile` file if you are using RVCT to compile the tests
- edit the integration kit project files to enable FPU support if you are using RealView MDK to compile the tests.

Otherwise, it is likely that all tests will fail.

The tests have been developed and tested using the full ARM RVCT toolchain running under Linux, and using RealView MDK ARM under Windows. You can download an evaluation version of the:

- RealView Development Suite, which includes RVCT, from `http://www.arm.com`
- RealView MDK-ARM from `http://www.keil.com`.

Note

See the release note for the ARM RVCT and RealView 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
- `Makefile`, or use an alternative system, if you want to use a different OS.

Table 5-4 shows the test support files in the `integration_kit/validation/tests` directory.

Table 5-4 Test support files

Filename	Description
`boot.c`	This file provides the stack and heap initialization, vector table, default handlers and `_sys_exit` function that updates the **TESTPASS** and **TESTCOMPLETE** signals when test code completes. A dummy `_fp_init()` function is provided for compliance, but this is not used with the `-microlib` option so FPU initialisation is not carried out in this function. The CMSIS provides assembler example startup files that you can modify and use instead of `boot.c`.
`retarget_cm4ikmcu.h`	This file implements the functions necessary to retarget the C-library `printf()` function output to the `CM4IKMCU` **GPIO** pins.
`IKtests.h`	This header file describes and defines the allocation of **GPIO** pins used by the `CM4IKMCU` in the integration kit. It also declares the function prototypes the integration kit tests use to communicate with the debug driver.
`IKtests.c`	This file provides the functions the integration kit tests uses to initialize the GPIOs and to communicate with the debug driver. The function `IKInit()` contains code to check the presence of the FPU and enable it if required.

Table 5-4 Test support files (continued)

Filename	Description
IKConfig.h	This file includes some defines that you must edit to match the implemented configuration of CORTEXM4INTEGRATION or CORTEXM4.
Makefile	This file enables you to build the integration kit tests using the ARM RVCT compiler.
stackheap.s	Defines the initial states of the stack and heap when compiling with microlib. This should not used if microlib is not selected.

5.6.1 Integration kit test configuration

The file IKConfig.h in the integration_kit/validation/tests directory defines the expected values the IK tests check against. Ensure the following IKConfig.h configuration options match the configuration of your RTL, see *Configuration options* on page 2-3 for more information:

EXPECTED_BE

Specifies the endianness of the processor in the integration kit.

Note

- If the Cortex-M4 processor is in big-endian configuration, you must also compile the IK tests as big-endian.
- If you are compiling tests using:
  - ARM RVCT and the Makefile, edit the COMPILE_BE variable
  - RealView MDK, consult the MDK-ARM documentation for details of the changes you must make to your project files.

EXPECTED_BKPT

Specifies the number of breakpoint comparators.

EXPECTED_TRACE_LVL

Specifies the level of trace.

EXPECTED_DBG_PORT

Specifies whether or not debug is included.

EXPECTED_JTAGnSW

Specifies the Cortex-M4 DAP protocol to use.

EXPECTED_NUMIRQ

Specifies the number of external interrupts.

EXPECTED_LVL_WIDTH

Specifies the interrupt priority width.

EXPECTED_BB

Specifies whether bit-banding is included.

EXPECTED_MPU

Specifies whether an MPU is present.

EXPECTED_FPU

Specifies whether an FPU is present.

Note

- If the Cortex-M4F processor is used, you must also compile the IK tests for Floating Point.
- If you are compiling tests using:
  - ARM RVCT and the `Makefile`, edit the `CPUTARGET` variable
  - RealView MDK, consult the MDK-ARM documentation for details of the changes you must make to your project files.

EXPECTED_WPT

Specifies the number of watchpoint comparators.

EXPECTED_DMATCH

Specifies whether data matching is enabled in the DWT.

EXPECTED_WIC

Specifies whether a WIC is present.

EXPECTED_WIC_LINES

Specifies the number of WIC lines.

Ensure the following `IKConfig.h` configuration options match the tied-off values of the corresponding signals:

EXPECTED_STCALIB

The expected value of **STCALIB[25:0]**. See *SysTick signals* on page 4-31 for information about how to determine this value for your design.

EXPECTED_BASEADDR

The expected value of the CORTEXM4DAP CoreSight Component pointer. This should point to the DAP entry in the ROM table in your system.

Note

If you use RealView MDK-ARM to build your tests, you can use the Configuration Wizard to update the values in `IKConfig.h` after opening the project in RealView MDK-ARM.

5.6.2 Test program compilation, ARM RVCT

This section describes how to compile the test programs with ARM RVCT 4.0 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.

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 is in the format:

   `make <test_program>` or `make all`

   where:

   `<test_program>` Selects an individual test program to compile. See Table 5-1 on page 5-4 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

   - The 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 5-1 on page 5-4.
   - To remove the compiled tests and prepare for a fresh compilation, type:
     ```
     make clean
     ```
   - To compile using the `--microlib` option, set the macro `USE_MICROLIB` to 1 in the Makefile in `integration_kit/validation/tests`.

5.6.3 Test program compilation, RealView MDK-ARM

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

The `integration_kit/validation/mdk` directory contains a multiproject workspace that includes project files for each of the integration kit 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
   ```

Intermediate build files are created in the `mdk` directory. Test binaries are created in the `tests` directory.

5.7 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*
- *Running RunIK with the -dsm option* on page 5-14
- *Running RunIK with -netlist option* on page 5-14
- *Simulation logs* on page 5-15
- *Modifying the RunIK script* on page 5-15
- *UPF or CPF SRPG simulation considerations* on page 5-15.

5.7.1 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 this switch to compile the integration kit 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 RVCT tools, if these tools are present. If you do not have the ARM RVCT tools, compile the tests before you run this script.
-all	Use this switch to run all integration kit tests.
-dsm	Use a DSM for the CORTEXM4 level.
-netlist	Use this switch to run the integration kit 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.
-gui	Run simulation interactively for debugging.
-microlib	Use the --microlib option when compiling tests.

<testname> gives the name of the test image file to run. This file is in the integration_kit/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.

5.7.2 Running RunIK with the -dsm option

Ensure that the DSM has been 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/dsmdotchrc is provided to set up the DSM. In addition, ensure that the file integration_kit/logical/tbench/verilog/dsm.vc points to the CORTEXM4 DSM.

5.7.3 Running RunIK with -netlist option

Ensure that the file integration_kit/logical/tbench/verilog/netlist.vc points to the netlist at the required level to run RunIK with -netlist option. To simulate netlist with back-annotated timing, ensure that:

- The Verilog define to include SDF is set. See *Configuring the testbench* on page 5-6.
- A symbolic link at the required level, either CORTEXM4IMP.sdf.gz or CORTEXM4INTEGRATIONIMP.sdf.gz, exists in integration_kit/validation and points to the actual .sdf file.

When using the CORTEXM4 level, the RTL for the integration level is used in the simulation.

5.7.4 Simulation logs

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

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

A test that passes but is unable to perform useful checking because of the device configuration will pass, and in addition the log will contain the following message:

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

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_CM4IK_TIMEOUT_CYCLES`, the runaway simulation timer terminates the test and the following message appears in the simulation log:

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

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

5.7.5 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.

5.7.6 UPF or CPF SRPG simulation considerations

Support for power gating flows is provided as an example, and must be modified to suit your chosen flow.

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

- SRPG signals are included. See *Configuring the testbench* on page 5-6 to see how SPRG signals are included.
- The file `integration_kit/validation/power_file` exists and it has a symbolic link to the appropriate UPF or CPF file.

  Note

  The UPF or CPF files are not included with the integration kit. Contact ARM for further details if you require these files.

- The `mvtools` suite is on your path for UPF simulation.

5.8 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.

The recommended debug strategy is as follows:

Prioritize the failures

The simplest test is `hellow`. If this test is among your failing tests, start debugging it first. If the code is being compiled for a VFP target, check that the FPU is present.

The `config_check` test is more complex, but it is the only test that checks the configuration of the processor against the values set in `IKConfig.h`. If this test is failing, you must resolve the issues, because other tests might assume that the `EXPECTED_*` values in `IKConfig.h` are correct. See *Integration kit test configuration* on page 5-10.

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 5-6 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 `CM4IKMCU_PRINTF` in `IKConfig.h` before compiling the tests. You might want to do this to reduce the runtime of the tests, and therefore the vectors.

If a test is failing, enable message printing by defining `CM4IKMCU_PRINTF`, for example by uncommenting it in `IKConfig.h`, and recompile and rerun the test to help determine the reason for failure.

You can add message printing in the fault handlers if you suspect that unexpected exceptions are occurring.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`, `sleep`.

You can enable printing of decompressed trace data by defining `DEBUGTRACE_PRINTF` in `IKConfig.h`.

Run the tests on an unmodified version of the integration kit, or see the golden logs in directory `glogs`, to view the output the tests should 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 produce a log file, `tarmac.log`, of the instructions executed on the processor. This 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 RVCT `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 RVCT to compile your tests.

5.9 Modifying the IK RTL for your SoC

This section describes the testbench structure and the integration kit level, CM4IKMCU, memory map. Read this section if you want to modify the IK RTL for your own SoC requirements.

5.9.1 Testbench structure

Figure 5-2 shows the testbench structure and its instantiated integration kit components.

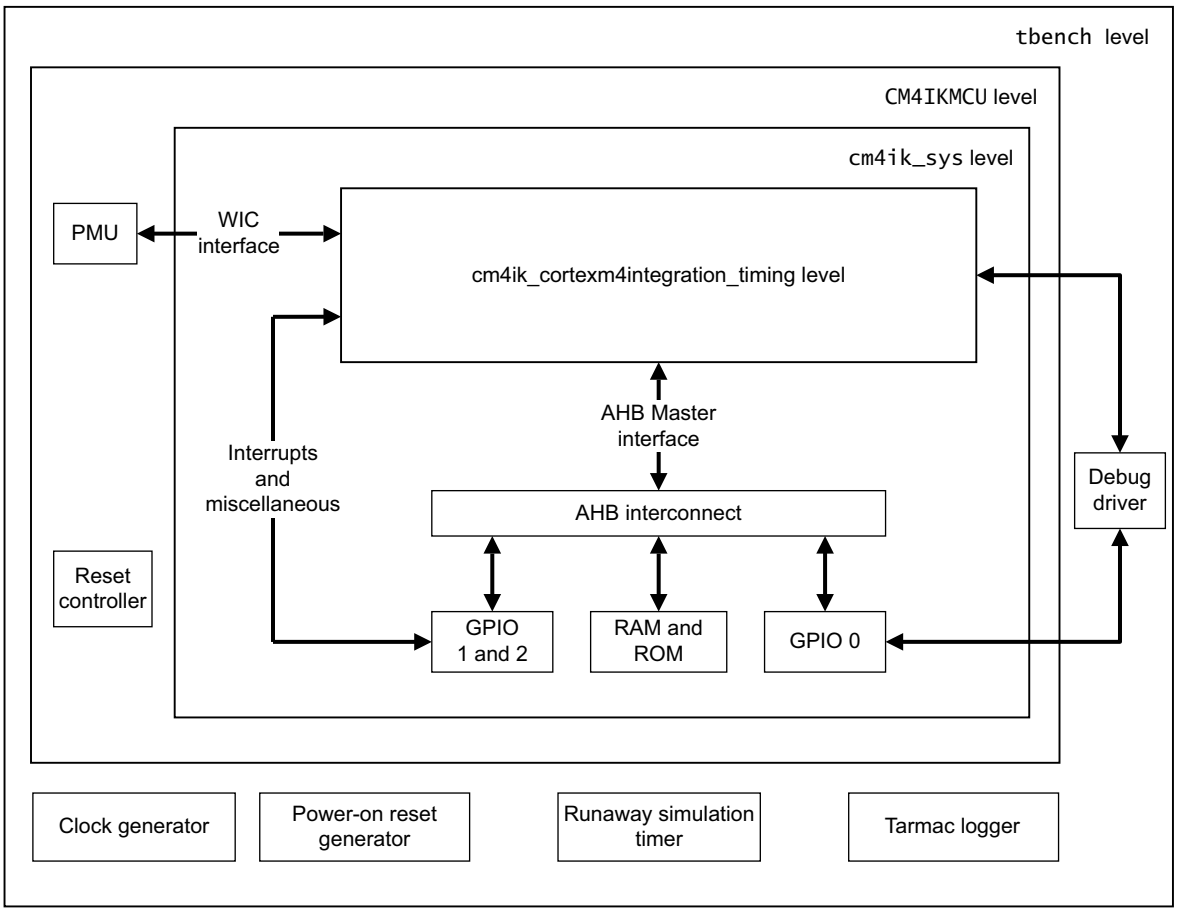


Figure 5-2 Integration kit

The integration kit contains these levels:

- *cm4ik_cortexm4integration_timing level*
- *cm4ik_sys level* on page 5-19
- *CM4IKMCU level* on page 5-20
- *tbench level* on page 5-20.

The `integration_kit/` path in Figure 1-6 on page 1-9 shows the supplied integration kit directory structure.

cm4ik_cortexm4integration_timing level

This level instantiates the `CORTEXM4INTEGRATION` level. This level delays all inputs to the `CORTEXM4INTEGRATION` level to support netlist simulations with timing.

cm4ik_sys level

This section describes the cm4ik_sys level and its components. The cm4ik_sys level contains:

General Purpose Input Output (GPIO)

GPIO is a general purpose I/O device.

You can configure the GPIO pins individually as inputs or outputs.

You can configure the GPIO to generate an interrupt when specific input values change.

Use the GPIO to:

- drive signals at the testbench level
- control the debug driver block to access the DAP.

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, and stop the simulation
- 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.

For more information see Appendix B *CM4IKMCU GPIO Integration*.

General Purpose Input Output 1

This GPIO drives signals in the example CM4IKMCU, for test purposes only. It drives PMU signals and other processor signals.

For more information see Appendix B *CM4IKMCU GPIO Integration*.

General Purpose Input Output 2

This GPIO drives signals in the example CM4IKMCU, 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.

For more information see Appendix B *CM4IKMCU GPIO Integration*.

AHB default slave

Accesses to unused address locations in the system go to the AHB default slave. The AHB default slave responds with an error each time it is accessed.

AHB ROM Memory

This is a zero wait-state, read-only AHB memory model that includes support for retention of state on power-down during power-aware simulations. The integration test code image is loaded here. The example code mux provided with the Cortex-M4 Integration Kit is used to combine the I-Code and D-Code buses.

Wait states can be introduced into the ROM memory by changing the define `ARM_CM4IK_RAM_WAIT_STATES` as described in Table 5-3 on page 5-7

SRAM controller and SRAM

This is a synchronous SRAM model with an example AHB SRAM controller that guarantees zero wait-state responses to all AHB accesses by supporting write data buffer forwarding.

Wait states can be introduced into the ROM memory by changing the define `ARM_CM4IK_RAM_WAIT_STATES` as described in Table 5-3 on page 5-7

AHB bus interconnect

This block does the AHB address decoding and multiplexing.

The single slave port is connected to the AHB-Lite master interface from the `CORTEXM4INTEGRATION` 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.

Miscellaneous logic

Miscellaneous logic used for integration test purposes.

System level ROM table

This module instantiates an example system level CoreSight ROM table. This permits debuggers to uniquely identify the Cortex-M4 based system. If you want to support system-level ROM tables, you must modify this module to include your JEP106 manufacturer ID and a part number that identifies your system.

CM4IKMCU level

The CM4IKMCU level is designed to be an example MCU and contains:

cm4ik_sys The provided example system level components. See *cm4ik_sys level* on page 5-19 for more information.

PMU

The PMU is an example system power controller that:

1. Implements a system power domain for the processor and manages the sequencing of isolation, retention, and power down.
2. Interfaces with the WIC to ensure that power-down and wake-up behavior is transparent to software.
3. Interfaces with the processor to manage extended sleep and ensure clean power-down.

The PMU provided is an example module that you can modify to suit your requirements.

Reset controller

The integration kit does not provide a full implementation of a reset controller or power-down extension. These must be implemented as appropriate for your system.

tbench level

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

CM4IKMCU level

The provided example CM4IKMCU level.

Clock generator This generates clocks for the CM4IKMCU level.

Power-on reset generator

This generates power-on reset.

Debug driver This programmable block controls the Cortex-M4 DAP. See Appendix C *Debug Driver* for more information.

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-M4 processor during a test run.

5.9.2 CM4IKMCU memory map

Figure 5-3 shows the CM4IKMCU memory map.

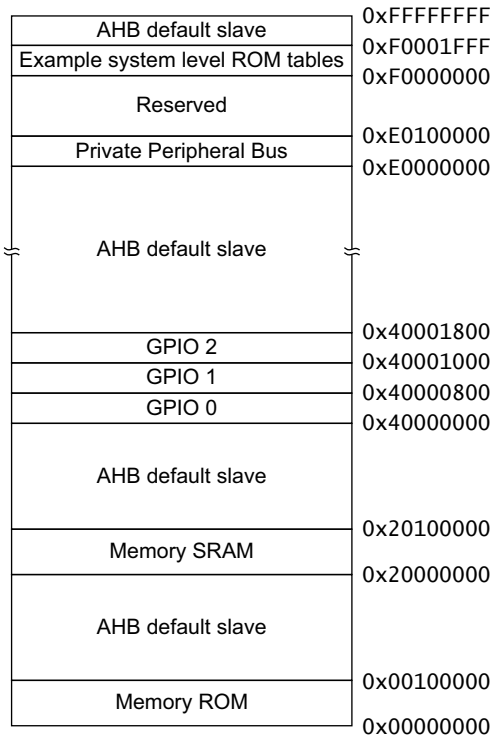


Figure 5-3 CM4IKMCU memory map

The CM4IKMCU memory map is characterized by the following:

- CM4IKMCU 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.
- The *Private Peripheral Bus* (PPB) space of the Cortex-M4 processor is 1MB.
- Example system level ROM tables occupy 8KB of space.
- All remaining memory regions are mapped to the AHB default slave, including the reserved space. Any access to the default slave results in a fault.

5.10 Modifying IK tests

The supplied test programs rely on the integration kit 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.

Some tests might have to be excluded if the environment does not use a particular interface or cannot support self-checking of interface signals.

Test programs are supplied as C source code with the integration kit. The header of each source code 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 integration kit 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 integration kit includes a minimal subset of the CMSIS for the Cortex-M4 processor that is sufficient to support the supplied tests. See `http://onarm.com` for the full version of the CMSIS and 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-M4-based device.

Table 5-5 shows the files in the `integration_kit/validation/tests` and `integration_kit/validation/tests/CMSIS` directories that constitute the CMSIS.

Table 5-5 CMSIS file descriptions

Filename	Location	Supplied by	Description
`CMSIS_Core.htm`	`tests/CMSIS/Core/Documentation/`	ARM	HTML format documentation for the CMSIS Core files.
`core_cm4.h`	`tests/CMSIS/Core/CM4/`	ARM	Defines the core peripherals for the Cortex-M4 processor and core peripherals.
`core_cm4.c`	`tests/CMSIS/Core/CM4/`	ARM	Provides helper functions that access processor registers.

Table 5-5 CMSIS file descriptions (continued)

Filename	Location	Supplied by	Description
cm4ikmcu.h	tests/	Device vendor	Device specific file that defines the peripherals for the CM4IKMCU example mircrocontroller device.
system_cm4ikmcu.h	tests/	Device vendor	Header file that provides device specific configuration for the CM4IKMCU example microcontroller device.
system_cm4ikmcu.c	tests/	Device vendor	C file that provides device specific configuration for the CM4IKMCU example microcontroller device.

5.11 Test vector capture

The vector capture testbenches provided permit you to capture vectors at the CORTEXM4INTEGRATION, CORTEXM4, and CM4ETM levels of hierarchy.

You can capture vectors for any code that you run in the integration kit. If vectors are captured from simulations using the RunIK script, they can be found at integration_kit/validation/vectors. The vectors are in the CRF format. The .crf filename includes the name of the level at which the vectors are captured. For example, vectors for sample_test.c at the CORTEXM4 level are named sample_test_CORTEXM4.crf and those at the CORTEXM4INTEGRATION level are named sample_test_CORTEXM4INTEGRATION.crf.

The verilog vector capture modules are:

- integration_kit/logical/tbench/verilog/CORTEXM4INTEGRATION_Capture.v
- integration_kit/logical/tbench/verilog/CORTEXM4_Capture.v
- integration_kit/logical/tbench/verilog/CM4ETM_Capture.v.

If you change the pinout of a level, you must update the corresponding vector capture module accordingly.

The integration_kit/logical/tbench/verilog/tbench.vc file contains Verilog defines that affect the way vectors are captured. They are:

ARM_CM4IK_VECTOR_CAPTURE

If this is defined, vectors are captured.

ARM_CM4IK_XVAL

An X detected at the input of the capture level is captured as logic LOW or L in the .crf file.

If this is not defined, an X at the input is captured as logic HIGH or H in the .crf file.

ARM_CM4IK_SKIPVECNUM

This takes an integer value and specifies the number of initial vectors whose outputs are not checked during replay. For example, the first few clock ticks before reset. In the .crf file, the output for a skipped vector is marked with a dot.

If you want to capture vectors at a higher level of hierarchy in your SoC, you must create your own vector capture testbench or modify either of the provided capture testbenches to reflect the I/O and clocking structure of your SoC, and ensure the vectors are saved to the appropriate directory.

Replay the test vectors using the replay testbench to perform quick netlist checks or obtain characterization information like speed and power from your netlist. You can use the replay testbench to verify that your netlist matches the cycle-by-cycle behavior of the RTL reference. See *Flow for dynamic verification* on page 9-5 for more information.

See the Reference methodology documents from your EDA tool vendor for information about how to replay test vectors.

Chapter 6
Key Implementation Points

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

- *About key implementation points* on page 6-2
- *Key implementation tasks* on page 6-3.
- *Other considerations for implementation* on page 6-4.

Note

This chapter mentions implementation steps that are described in more detail in the Reference Methodology documents from your EDA tool vendor. See *Additional reading* on page xiii.

6.1 About key implementation points

This chapter lists the main points to consider when you implement the Cortex-M4. Read this chapter in conjunction with the rest of the information in this guide and your Cortex-M4 processor reference methodology documentation.

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

6.2 Key implementation tasks

Table 6-1 lists the key tasks for implementation.

Table 6-1 Key implementation tasks

Key task		Description
1.	Select level of hierarchy to implement. This can be either: a. The integration level, `CORTEXM4INTEGRATION`. b. The processor component level, `CORTEXM4`. c. A higher level in your SoC that includes the Cortex-M4 processor.	See *About configuration guidelines* on page 2-2 and *Configuration options* on page 2-3
2.	Configure the processor parameters.	See *Configuration options* on page 2-3
3.	Select appropriate library cells for clock gating and *Clock-Domain Crossing* (CDC) purposes.	*Other considerations for implementation* on page 6-4
4.	Determine optimum floorplan.	See *Considerations for floorplans* on page 7-6
5.	Perform synthesis and scan insertion.	See the Reference methodology documents from your EDA tool vendor for information on equivalence checking tools.
6.	Create layout.	
7.	Perform LVS and DRC checks.	
8.	Perform timing verification.	
9.	Perform characterization.	
10.	Run ATPG.	See Chapter 8 *Design For Test* and the Reference methodology documents from your EDA tool vendor.
11.	Perform netlist dynamic verification.	See Chapter 9 *Netlist Dynamic Verification*.
12.	Perform functional verification using logical equivalence checking tools. Optionally, you can also replay test vectors.	See the Reference methodology documents from your EDA tool vendor.
13.	Perform sign-off in accordance with the agreed criteria and your sign-off obligations.	See Chapter 10 *Sign-off*
14.	Sign off your implementation.	

Note

The outputs from the tasks in Table 6-1 must produce complete and verified deliverables as described in *Obligations for sign-off* on page 10-3.

6.3 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*.

6.3.1 Architectural clock gating

The Cortex-M4 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 `cm4_clk_gate.v`. If using the ETM, you must also provide module `cm4_etm_clk_gate.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 `CLKGATE_PRESENT` parameter set to 0.
2. Ensure that your `cm4_clk_gate` module drives the **CLKOUT** output directly from the **CLKIN** input.

If architectural clock gating is required:

1. Configure your processor to have the `CLKGATE_PRESENT` parameter set to 1.
2. Ensure that your `cm4_clk_gate` 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/cm4_clk_gate.v` is a configurable module that you can use for simulation and logical equivalence checking. Do not use this version for synthesis.

Chapter 7
Floorplan Guidelines

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

7.1 About floorplanning

Floorplanning is the compact physical arrangement of IP and memory components under a given design level, in preparation for layout as a hardened macrocell. ARM recommends that you perform the synthesis and layout from the CORTEXM4INTEGRATION level of hierarchy, or from your complete SoC level of hierarchy.

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

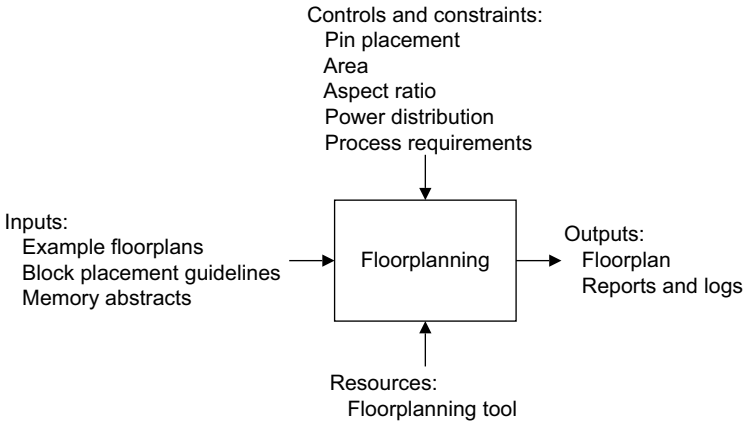


Figure 7-1 Floorplanning process

7.2 Resource requirements for floorplans

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

7.3 Controls and constraints for floorplans

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 Cortex-M4 processor timing performance.

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

7.4 Inputs for floorplans

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

- example floorplans
- block placement guidelines
- memory abstracts.

7.5 Considerations for floorplans

ARM recommends that you incorporate the power grid in the floorplan 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.

7.6 Outputs from floorplans

Output files are specific to your floorplanning tool. They might include:

- logs
- floorplan with power grid and pin locations
- placement and route guides.

Chapter 8
Design For Test

This chapter describes ARM-specific provisions for *Design for Test* (DFT). It contains the following sections:

8.1 About Design for Test

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

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

Figure 8-1 shows the top level inputs, resources, outputs, and controls and constraints for DFT.

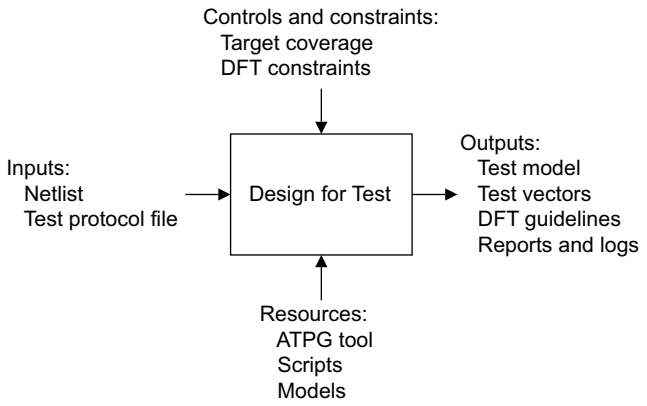


Figure 8-1 Design for Test process

8.2 Requirements for DFT

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

8.3 Controls and constraints for DFT

The following sections describe the controls and constraints for DFT:

- *Clock gating*
- *Reset*.

8.3.1 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 on when scan enable is asserted. This applies to:

- clock gates that are inferred in the RTL and inserted by Power Compiler
- architectural clock gates.

You can use **CGBYPASS** to permanently enable the clock gating cells during scan testing.

You can tie **CGBYPASS** HIGH during scan testing, and LOW during normal operation.

8.3.2 Reset

You can use **RSTBYPASS** to bypass the reset synchronizers during scan testing.

You can tie **RSTBYPASS** HIGH during scan testing, and LOW during normal operation.

Note

RSTBYPASS is available only in the `CORTEXM4INTEGRATION` level. The `CORTEXM4` level does not implement reset synchronizers.

8.4 DFT features

See the Reference Methodology documents from your EDA tool vendor for information on DFT features.

8.5 Reference data for DFT

This section gives reference data for scan test insertion:

8.5.1 Scan test insertion

Table 8-1 lists the scan test ports that access the internal scan chains in the macrocell.

Table 8-1 Scan test ports

Name	Direction	Description
SE	Input	Enable Scan chains. High fan out of this signal means this must be a multicycle path, and cannot be switched at-speed. This signal must be tied LOW during functional mode.
RSTBYPASS	Input	Bypass reset synchronizers during scan testing so that test tools have direct control over **PORESETn** and **SYSRESETn**.
CGBYPASS	Input	Enable clock gating cells during scan testing.

Note

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

Chapter 9
Netlist Dynamic Verification

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

9.1 About netlist dynamic verification

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

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 test vectors to perform quick netlist checks before you perform formal or functional verification. This is described in this chapter, and Figure 9-1 shows the top level inputs, resources, outputs, and controls and constraints for this dynamic verification.

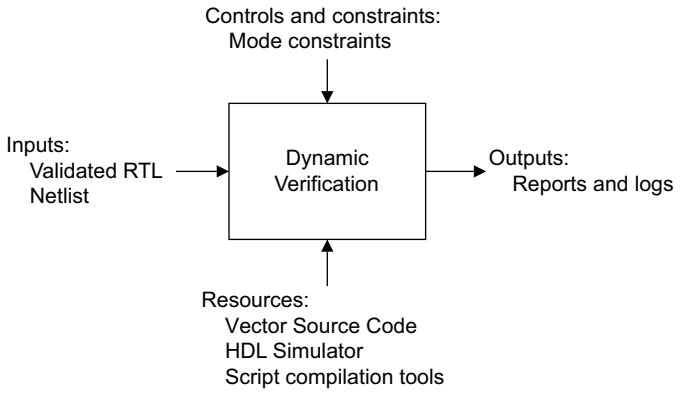


Figure 9-1 Dynamic verification process

This verification is described in *Verification using test vectors* on page 9-5.

9.2 Resource requirements for netlist dynamic verification

To run netlist dynamic verification, you require various test vectors. You must generate test vectors by using your own testbench, using the supplied capture modules. See *Test vector capture* on page 5-24 for information about capturing test vectors from your configured RTL.

Table 9-1 shows the netlist dynamic verification test vector locations and descriptions.

Table 9-1 Netlist dynamic verification test vector descriptions

Test vector directory	Content description
`implementation/vectors/CORTEXM4/crf`	The I/O specification file, `CTRM`, for the `CORTEXM4` level. Test vectors captured from your configured RTL.
`implementation/vectors/CORTEXM4/tbench`	Verilog testbench for replaying the test vectors at the `CORTEXM4` implementation level.
`implementation/vectors/CORTEXM4INTEGRATION/crf`	The I/O specification file, `CTRM`, for the `CORTEXM4INTEGRATION` implementation level. Test vectors captured from your configured RTL.
`implementation/vectors/CORTEXM4INTEGRATION/tbench`	Verilog testbench for replaying the test vectors at the `CORTEXM4INTEGRATION` implementation level.
`implementation/vectors/CM4ETM/tbench`	Verilog testbench for replaying the test vectors at the `CM4ETM` implementation level.
`implementation/vectors/CM4ETM/crf`	The I/O specification file, `CTRM`, for the `CM4ETM` level. Test vectors captured from your configured RTL.
`implementation/vectors/tools`	Scripts to process vectors and run vector replay testbench.
`logical`	Verilog source for the macrocell.

9.3 Inputs for dynamic verification

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

- validated RTL
- netlist
- test vectors.

9.4 Flow for dynamic verification

This section describes verification using test vectors in the following subsections:

- *Verification using test vectors*
- *Test vectors directory structure*
- *Test vector simulation environment and vector replay*.

Note

You must also perform verification using equivalence checking tests. See the Reference Methodology documentation from your EDA tools vendor for more information.

9.4.1 Verification using test vectors

You can use the test vectors to perform quick netlist checks before performing formal verification. Because the vectors are captured using the source RTL as a reference, replaying the vectors checks that your netlist matches the cycle-by-cycle behavior of the RTL reference. The test vectors only cover a small amount of the Cortex-M4 functionality and are therefore not suitable for design sign-off. It is important that the RTL configuration used for vector capture matches the configuration used for replay.

9.4.2 Test vectors directory structure

You can capture and replay test vectors at each of three levels of the Cortex-M4 hierarchy:

- `CORTEXM4`
- `CORTEXM4INTEGRATION`
- `CM4ETM`.

The directories you can use for test vector replay are identified in Table 9-1 on page 9-3.

Note

The maximum speed and power of your Cortex-M4 implementation depends on many factors, such as process, floorplan, and Cortex-M4 configuration. The provided vectors are generic, and it is possible that they might not target the maximum speed and power of your implementation.

9.4.3 Test vector simulation environment and vector replay

This section describes how to build the replay testbench and replay the vectors on the testbench.

From the `implementation/vectors` directory execute the following command:

```
source dotcshrc
```

Run the testbench from the `implementation/vectors/<level>/tbench` directory, depending on the level at which you want to replay vectors. The directory `<level>` is one of `CORTEXM4`, `CORTEXM4INTEGRATION`, or `CM4ETM`. The `Replay` script is provided as an example script to build the testbench and replay the test vectors. Ensure that the RTL or netlist configuration in the replay testbench is the same as the one used to capture vectors in the integration kit.

Replay script usage and options

The `Replay` script has the following command format:

```
Replay <options> <testname>
```

Table 9-2 lists the options for the Replay script.

Table 9-2 Replay script options

Option	Description
-build	Compile the testbench before running the specified test.
-all	Run a batch of tests.
-integration	Use the CORTEXM4INTEGRATION level. Default is CORTEXM4.
-etm	Use the CM4ETM level. Default is CORTEXM4.
-netlist	Use netlist for simulation.
-sdf <delay type>	Use SDF with the netlist. <delay type> can be min or max. If this option is not used, zero delay is assumed.
-srpg	Use and check the SRPG pins for vector replay.
-mti	Use the MTI simulator. This is the default simulator used unless you specify the NC or VCS simulator.
-nc	Use the NC simulator.
-vcs	Use the VCS simulator.

<testname> gives the name of the .crf file to run. Depending on the level at which you are replaying the vectors, this file is in either the implementation/vectors/CORTEXM4/crf directory, the implementation/vectors/CORTEXM4INTEGRATION/crf directory, or the implementation/vectors/CM4ETM/crf directory. <testname> must give the name of the file without the .crf extension. You must indicate the level at which the vectors are being replayed to the Replay script, using the relevant option, for example -integration or -etm.

Replaying testbench vectors

Depending on the simulator you specify, the Replay script creates the following directories for compilation and simulation in the tbench directory:

- MTI
- VCS
- NC.

Ensure that your chosen simulator is on your path before running the Replay script. You can modify the Replay script to include extra simulator options you want to use. For example, modify the array @testlist in Replay to add tests for batch test replay.

Note

Not all tests are included by default.

The testbench prints a summary at the end of simulation. Verify that all vectors replay with zero errors.

9.5 Outputs from dynamic verification

The log files output are specific to your verification.

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

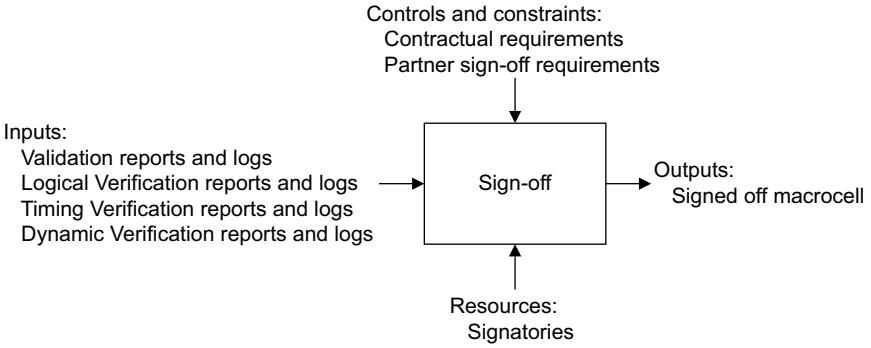


Figure 10-1 Sign-off process

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.

For more information see *Implementation obligations* on page 1-5.

10.3 Criteria 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 your EDA tool vendor.

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

A certain minimum set of deliverable outputs is required at the end of the implementation. 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 criteria 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)	See the Reference Methodology documents supplied by your EDA tool vendor.
Layout Versus Schematic (LVS)	
Characterization	
Timing verification through *Static Timing Analysis* (STA)	
Vector replay with back-annotated netlist	If you are using a pre-hardened processor at the `CORTEXM4` or `CORTEXM4INTEGRATION` levels.

10.4 Steps for sign-off

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

1. *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. Running these tests demonstrates that the RTL has been successfully installed and configured.

Note

You must use the files provided in the `models/cells/` directory for RTL integration. Do not use the implementation version of files for RTL integration. See *Other considerations for implementation* on page 6-4 for more 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 your EDA tool vendor.

10.4.3 Post place-and-route timing

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

- all of the functional vectors appropriate to your configured build
- back-annotated timing, as a final check.

10.5 Completion of sign-off

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

- GDS II output
- test vectors
- extracted timing model
- simulation model
- all required reports and logs.

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.

Table A-1 lists the GPIO registers.

Table A-1 GPIO registers

Address offset	Name	Type	Description
0x00000000-0x000003FF	**GPIODATA**	R/W	Reads current value of **GPIOIN** pins, or sets 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.

See Appendix B *CM4IKMCU GPIO Integration* for details of the connections to **GPIOIN** and **GPIOOUT**.

A.1.1 Data Register - GPIODATA

Use this register to read the value of the **GPIOIN** pins when the corresponding **GPIODIR** is 0.

Use this register to drive the value onto the **GPIOOUT** pins when the corresponding **GPIODIR** is 1.

Note

Reading **GPIODATA** when **GPIOEN** is 1 returns the value 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]** as a mask. This enables you to perform various bit set and bit clear operations efficiently because it avoids the requirement for a read-modify-write to the **GPIODATA** register.

Bit set example

To set bit [9] of GPIO 0 Data Register, perform a byte write access to address 0x40000009 with **HWDATA** set to 0x200 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 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 `0b00`.

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.

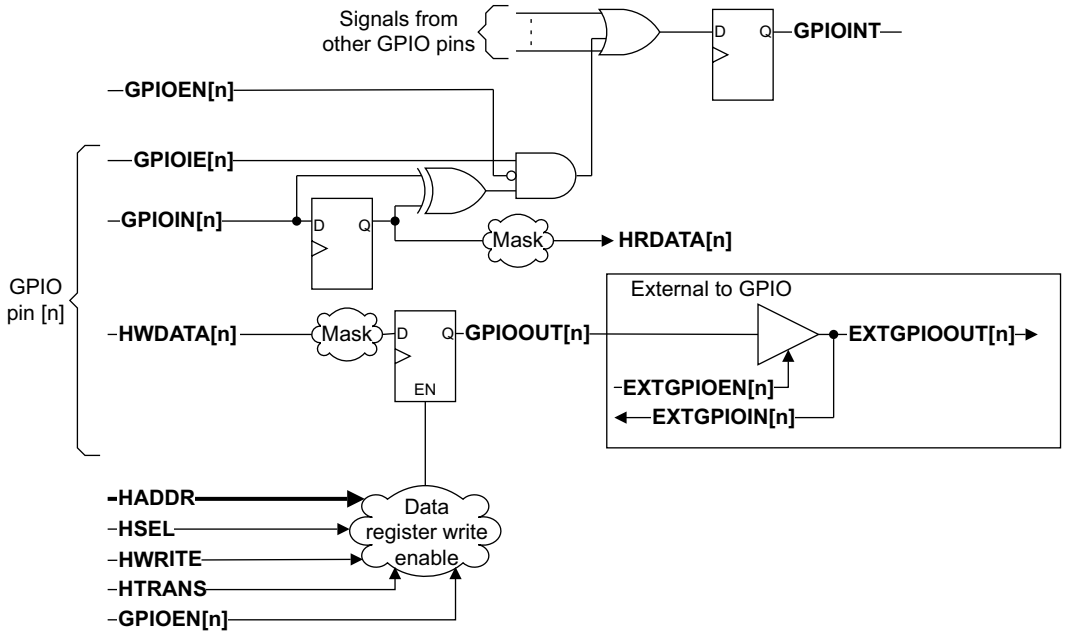


Figure A-1 GPIO Data Register

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 `0b1` when you want to use the bit as an output.

The register address, access type, and reset state are:

Address `GPIO_BASE + 0x00000400`
Access Read/write
Reset state `0x00000000`

A.1.3 Interrupt Enable Register - GPIOIE

Use this register to enable input signal changes on **GPIOIN** to trigger an interrupt through the **GPIOINT** output.

You can set **GPIOIE[n]** to enable changes on **GPIOIN[n]** to pulse the **GPIOINT** pin.

The register address, access type, and reset state are:

Address `GPIO_BASE + 0x00000410`
Access Read/write
Reset state `0x00000000`

Appendix B
CM4IKMCU GPIO Integration

This appendix describes the integration of the GPIO within the CM4IKMCU 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-4
- *GPIO 2 bit assignments* on page B-5.

The Cortex-M4 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.

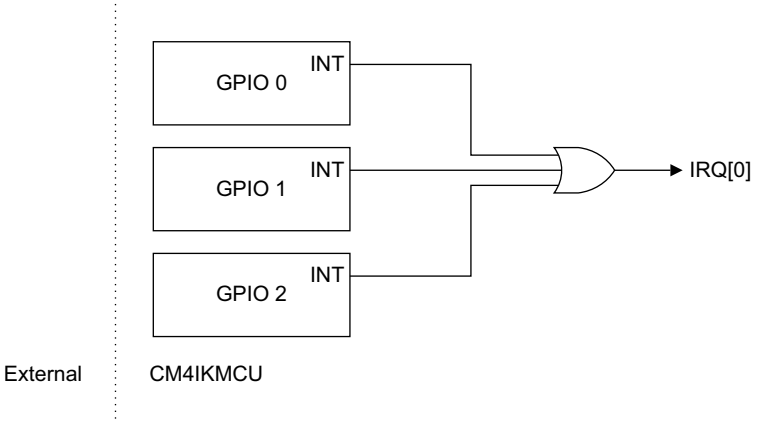


Figure B-1 GPIO interrupt line connections

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.

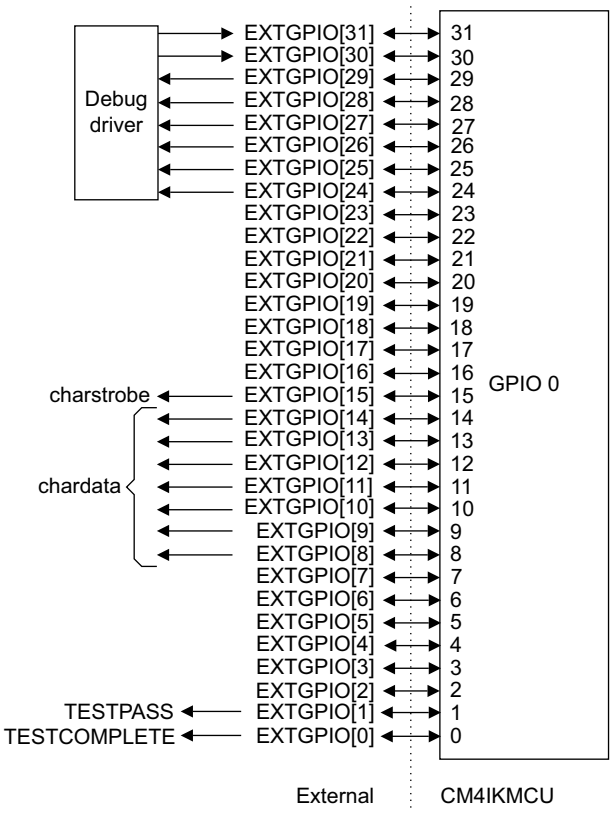


Figure B-2 GPIO 0 connections

Figure B-3 shows the GPIO 1 connections.

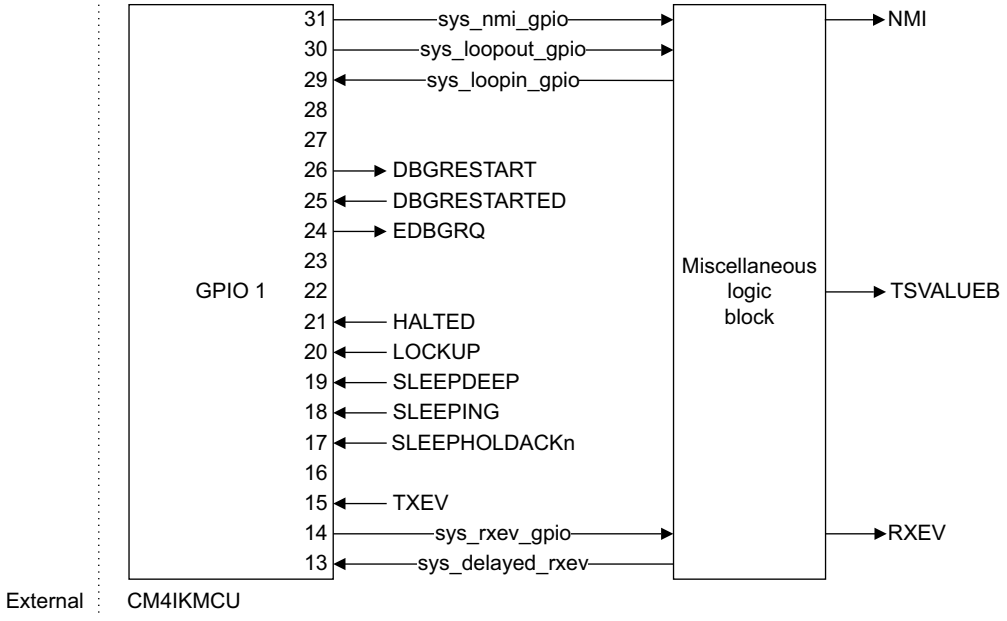


Figure B-3 GPIO 1 connections

Figure B-4 on page B-4 shows the GPIO 2 connections.

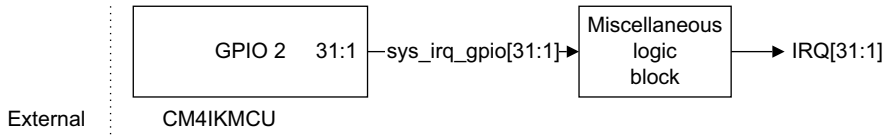


Figure B-4 GPIO 2 connections

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 CM4IKMCU, with its signals routed to the top level of the CM4IKMCU.

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-M4 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 CM4IKMCU 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-M4 processor, after a delay
[30]	-	Miscellaneous logic block	Drives **GPIOIN[29]**, after a delay
[29]	Miscellaneous logic block	-	Delayed version of **GPIOOUT[30]**

Table B-2 GPIO 1 bit assignments (continued)

Bit	Receive input from	Send output to	Connection information
[28:27]	-	-	Not used
[26]	-	**DBGRESTART**	Drives the **DBGRESTART** pin of the Cortex-M4 processor
[25]	**DBGRESTARTED**	-	Connects to the **DBGRESTARTED** pin of the Cortex-M4 processor
[24]	-	**EDBGRQ**	Drives the **EDBGRQ** pin of the Cortex-M4 processor
[23:22]	-	-	Not used
[21]	**HALTED**	-	Connects to the **HALTED** pin of the Cortex-M4 processor
[20]	**LOCKUP**	-	Connects to the **LOCKUP** pin of the Cortex-M4 processor
[19]	**SLEEPDEEP**	-	Connects to the **SLEEPDEEP** pin of the Cortex-M4 processor
[18]	**SLEEPING**	-	Connects to the **SLEEPING** pin of the Cortex-M4 processor
[17]	**SLEEPHOLDACKn**	-	Connects to the **SLEEPHOLDACKn** pin of the Cortex-M4 processor.
[16]	-	-	Not used
[15]	**TXEV**	-	Connects to the **TXEV** pin of the Cortex-M4 processor
[14]	-	**RXEV**	Drives the **RXEV** pin of the Cortex-M4 processor
[13]	Miscellaneous logic block	-	Delayed version of the **RXEV** pin of the Cortex-M4 processor
[12:0]	-	-	Not used

B.1.3 GPIO 2 bit assignments

GPIO 2 is used internally to the example CM4IKMCU 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-M4 processor, after a delay. For example, bit[31] drives **IRQ[31]**.
[0]	-	-	Not used.

Appendix C
Debug Driver

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

C.1 Debug driver

The debug driver block is a testbench component that controls the Cortex-M4 DAP using the JTAG or SW pins of CM4IKMCU. The debug driver contains an instantiation of the `cm4ik_cortexm4_timing` level or `cm4ik_cortexm4integration_timing` level, memories, and a GPIO. You can control the debug driver block using the pins **EXTGPIO[31:24]** of CM4IKMCU to initiate one of several predetermined operations. This arrangement enables testing of the integration kit even when the processor in the CM4IKMCU is sleeping.

The debug driver also provides functionality to capture trace data from the TRACEDATA or SWV pins of CM4IKMCU. The trace data can be uncompressed by the debug driver software. For integration testing, this is limited to confirming that the trace data conforms to the correct protocol for ETM and ITM.

The debug driver always executes the binary file `debugdriver.bin`, regardless of the test run on CM4IKMCU. The `debugdriver.bin` binary must be built from the supplied source code. See *Test program compilation, ARM RVCT* on page 5-11 for information on how to compile and build the integration kit tests.

Table 3-1 lists the software source files used by the debug driver.

Table 3-1 Debug driver source files

File	Location	Description
`core_cm4.h`	`tests/CMSIS/Core/CM4/`	CMSIS file that defines the core peripherals for the Cortex-M4 processor and core peripherals.
`core_cm4.c`	`tests/CMSIS/Core/CM4/`	CMSIS file that provides helper functions that access processor registers.
`cm4ikdebugdriver.h`	`tests/`	CMSIS device specific file that defines the peripherals for the `cm4ikdebugdriver` block.
`system_cm4ikdebugdriver.h`	`tests/`	CMSIS device vendor Header file that provides device specific configuration for the `cm4ikdebugdriver` block.
`system_cm4ikdebugdriver.c`	`tests/`	CMSIS device vendor C file that provides device specific configuration for the `cm4ikdebugdriver` block.
`boot_cm4debugdriver.c`	`tests/`	This file provides the stack and heap initialization, vector table, default handlers and `_sys_exit` function used by `cm4ikdebugdriver`.
`retarget_cm4debugdriver.c`	`tests/`	This file implements the functions necessary to retarget the C-library `printf()` function output to the `cm4ikdebugdriver` 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 CM4IKMCU through the GPIO and the routines to drive the CM4IKMCU 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 Cm4IKMCU.

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

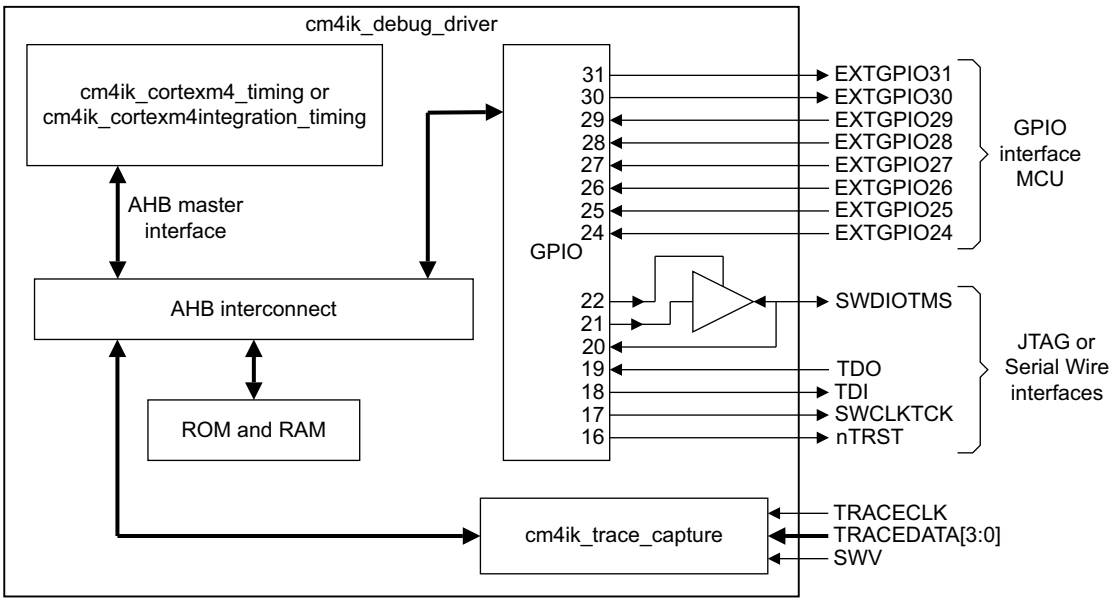


Figure C-1 Debug driver

Note

- The timing wrapper in the debug driver is the same as that used in the CM4IKMCU level.
- During netlist simulations, the implementation levels in the timing wrappers in CM4IKMCU and the debug driver are replaced by netlists.
- 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 D
SysTick Examples

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

D.1 SysTick examples

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

Example D-1 25MHz FCLK, 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 D-2 shows 14.31818MHz **FCLK** with no reference provided.

Example D-2 14.31818MHz FCLK, 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 D-3 shows 1MHz **FCLK** with **STCLK** enabled once every four cycles.

Example D-3 1MHz FCLK, STCLK 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 D-4 shows an 32.768kHz asynchronous example.

Example D-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 E
Revisions

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

Table E-1 Issue A

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

Table E-2 Differences between issue A and issue B

Change	Location	Affects
Additional references added.	*ARM publications* on page xiii	All
`AHB_CONST_CTRL` added.	Table 2-1 on page 2-3	All
Path for trace support changed.	*Trace support* on page 2-5	All
Path for floating point support changed.	*Floating point support* on page 2-5	All
Updated External AHB-Lite bus interface information.	*External AHB-Lite bus interfaces* on page 4-8	All
Added to WIC interface signals information.	Table 4-19 on page 4-28	All
CortexM4Integration changed to CORTEXM4INTEGRATION.	Figure 4-15 on page 4-38 Figure 4-16 on page 4-39 Figure 4-17 on page 4-39	All
Additional test programs added.	Table 5-1 on page 5-4	All
Process for AHB protocol checkers added.	*Configuring the testbench* on page 5-6	All

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

Change	Location	Affects
Additional testbench option added.	Table 5-3 on page 5-7	All
Additional test support file added.	Table 5-4 on page 5-9	All
`USE_MICROLIB` macro note added.	*Test program compilation, ARM RVCT* on page 5-11	All
RunIK options added.	*RunIK script usage and options* on page 5-13	All
Wait states description added.	*cm4ik_cortexm4integration_timing level* on page 5-18	All

Glossary

This glossary describes some of the terms used in technical documents from ARM Limited.

AHB-Lite AHB-Lite is a subset of the full AHB specification. It is intended for use in designs where only a single AHB master is used. This can be a simple single AHB master system or a multi-layer AHB system where there is only one AHB master on a layer.

Aligned Refers to data items stored so that their address is divisible by the highest power of two that divides their size. Aligned words and halfwords therefore have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore refer to addresses that are divisible by four and two respectively. The terms byte-aligned and doubleword-aligned are defined similarly.

ATPG *See* Automatic Test Pattern Generation.

Automatic Test Pattern Generation (ATPG)
The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.

Clock gating Gating a clock signal for a macrocell with a control signal, and using the modified clock that results to control the operating state of the macrocell.

Register A temporary storage location used to hold binary data until it is ready to be used.

Scan chain A scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

TAP *See* Test Access Port.

Test Access Port (TAP)

The collection of four mandatory terminals and one optional terminal that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **TRST**.

Unaligned

Memory accesses that are not appropriately word-aligned or halfword-aligned.

See also Aligned.