ARM® Cortex®-M4 Processor

Revision: r0p1

Technical Reference Manual

ARM® Cortex®-M4 Processor

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
A	22 December 2009	Non-Confidential	First release for r0p0
B	02 March 2010	Non-Confidential	Second release for r0p0
C	29 June 2010	Non-Confidential	First release for r0p1
D	11 June 2013	Non-Confidential	Second release for r0p1
0001-00	23 February 2015	Non-Confidential	Document source updated to comply with DITA standards. Document number changed to 100166 following conversion to DITA-XML.

Non-Confidential Proprietary Notice

Product Status

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

Web Address

http://www.arm.com

Contents

ARM® Cortex®-M4 Processor Technical Reference Manual

Preface

About this book 7
Feedback 10

Chapter 1 **Introduction**

1.1 About the processor 1-12
1.2 Features 1-13
1.3 External interfaces 1-14
1.4 Configurable options 1-15
1.5 Product documentation 1-16
1.6 Product revisions 1-19

Chapter 2 **Functional Description**

2.1 About the functions 2-21
2.2 Processor features 2-22
2.3 Interfaces 2-24

Chapter 3 **Programmers' Model**

3.1 About the programmers' model 3-28
3.2 Modes of operation and execution 3-29
3.3 Instruction set summary 3-30
3.4 Processor memory model 3-40

3.5 Write buffer 3-43
3.6 Exclusive monitor 3-44
3.7 Bit-banding 3-45
3.8 Processor core register summary 3-47
3.9 Exceptions 3-49

Chapter 4 System Control
4.1 System control registers 4-52
4.2 Auxiliary Control Register, ACTLR 4-54
4.3 CPUID Base Register, CPUID 4-55
4.4 Auxiliary Fault Status Register, AFSR 4-56

Chapter 5 Memory Protection Unit
5.1 About the MPU 5-58
5.2 MPU functional description 5-59
5.3 MPU programmers model table 5-60

Chapter 6 Nested Vectored Interrupt Controller
6.1 NVIC functional description 6-62
6.2 NVIC programmers' model 6-63

Chapter 7 Floating-Point Unit
7.1 About the FPU 7-66
7.2 FPU functional description 7-67
7.3 FPU programmers' model 7-72

Chapter 8 Debug
8.1 Debug configuration 8-74
8.2 AHB-AP debug access port 8-79
8.3 Flash Patch and Breakpoint Unit (FPB) 8-82

Chapter 9 Data Watchpoint and Trace Unit
9.1 DWT functional description 9-85
9.2 DWT Programmers' model 9-86

Chapter 10 Instrumentation Trace Macrocell Unit
10.1 ITM functional description 10-89
10.2 ITM programmers' model 10-90
10.3 ITM Trace Privilege Register, ITM_TPR 10-91

Chapter 11 Trace Port Interface Unit
11.1 About the TPIU 11-93
11.2 TPIU functional description 11-94
11.3 TPIU programmers' model 11-96

Appendix A Revisions
A.1 Revisions Appx-A-107

Preface

This preface introduces the *ARM® Cortex®-M4 Processor Technical Reference Manual*.

It contains the following:

- *About this book* on page 7.
- *Feedback* on page 10.

About this book

ARM Cortex-M4 Technical Reference Manual (TRM). This manual contains documentation for the Cortex-M4 processor, the programmer's model, instruction set, registers, memory map,floating point, multimedia, trace and debug support.

Product revision status

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

r*m*	Identifies the major revision of the product, for example, r1.
p*n*	Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

This manual is written to help system designers, system integrators, verification engineers, and software programmers who are implementing a System-on-Chip (SoC) device based on the Cortex®-M4 processor.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces the Cortex-M4 processor and instruction set, processor features and interfaces, configurable options, and product documentation.

Chapter 2 Functional Description
This chapter introduces the Cortex-M4 processor and its external interfaces.

Chapter 3 Programmers' Model
This chapter describes the Cortex-M4 processor programmers' model.

Chapter 4 System Control
This chapter provides a summary of the system control registers whose implementation is specific to the Cortex-M4 processor.

Chapter 5 Memory Protection Unit
This chapter describes the processor *Memory Protection Unit* (MPU).

Chapter 6 Nested Vectored Interrupt Controller
This chapter describes the *Nested Vectored Interrupt Controller* (NVIC). The NVIC provides configurable interrupt handling abilities to the processor, facilitates low- latency exception and interrupt handling, and controls power management.

Chapter 7 Floating-Point Unit
This chapter describes the programmers' model of the *Floating-Point Unit* (FPU).

Chapter 8 Debug
This chapter describes how to debug and test software running on the processor.

Chapter 9 Data Watchpoint and Trace Unit
This chapter describes the *Data Watchpoint and Trace* (DWT) unit.

Chapter 10 Instrumentation Trace Macrocell Unit
This chapter describes the *Instrumentation Trace Macrocell* (ITM) unit.

Chapter 11 Trace Port Interface Unit
This chapter describes the Cortex-M4 TPIU, the Trace Port Interface Unit that is specific to the Cortex-M4 processor.

Appendix A Revisions
The technical changes between released issues of this manual.

Glossary

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

See the *ARM Glossary* for more information.

Typographic conventions

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

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

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

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

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

`monospace bold`
: Denotes language keywords when used outside example code.

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

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

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

Timing diagrams

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

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

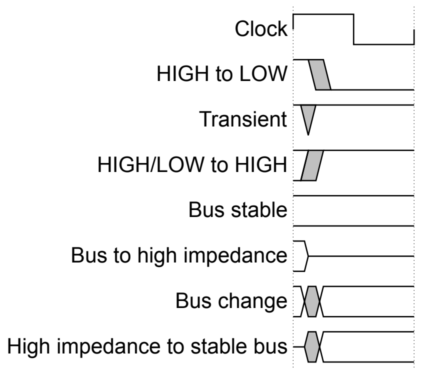


Figure 1 Key to timing diagram conventions

Signals

The signal conventions are:

Signal level

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

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

Lower-case n

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

Additional reading

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

ARM publications

- ARMv7-M Architecture Reference Manual (ARM DDI 0403).
- ARM® Cortex-M4 Integration and Implementation Manual (ARM DII 0239).
- ARM ETM-M4 Technical Reference Manual (ARM DDI 0440).
- ARM AMBA® 3 AHB-Lite Protocol (v1.0) (ARM IHI 0033).
- ARM AMBA 3 APB Protocol Specification (ARM IHI 0024).
- ARM CoreSight™ Components Technical Reference Manual (ARM DDI 0314).
- ARM Debug Interface v5 Architecture Specification (ARM IHI 0031).
- Cortex-M4 Lazy Stacking and Context Switching Application Note 298 (ARM DAI0298).

Other publications

- IEEE Standard [Test Access Port and Boundary-Scan Architecture]1149.1-2001 (JTAG).
- IEEE Standard [IEEE Standard for Binary Floating-Point Arithmetic] 754-2008.

Feedback

Feedback on this product

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

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

Feedback on content

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

- The title.
- The number ARM 100166_0001_00_en.
- The page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Introduction

This chapter introduces the Cortex-M4 processor and instruction set, processor features and interfaces, configurable options, and product documentation.

It contains the following sections:

- *1.1 About the processor* on page 1-12.
- *1.2 Features* on page 1-13.
- *1.3 External interfaces* on page 1-14.
- *1.4 Configurable options* on page 1-15.
- *1.5 Product documentation* on page 1-16.
- *1.6 Product revisions* on page 1-19.

1.1 About the processor

The Cortex-M4 processor is a low-power processor that features low gate count, low interrupt latency, and low-cost debug. The Cortex-M4 includes optional floating point arithmetic functionality. The processor is intended for deeply embedded applications that require fast interrupt response features.

Related references

Chapter 7 Floating-Point Unit on page 7-65.

1.2 Features

The Cortex-M4 processor incorporates a processor core, Nested Vectored Interrupt Controller (NVIC), high-performance bus interfaces, a low-cost debug solution, and an optional Floating Point Unit (FPU).

The Cortex-M4 processor incorporates the following features:

- A processor core.
- A *Nested Vectored Interrupt Controller* (NVIC) closely integrated with the processor core to achieve low-latency interrupt processing.
- Multiple high-performance bus interfaces.
- A low-cost debug solution with the optional ability to:
  - Implement breakpoints and code patches.
  - Implement watchpoints, tracing, and system profiling.
  - Support `printf()` style debugging.
  - Bridge to a *Trace Port Analyzer* (TPA).
- An optional *Memory Protection Unit* (MPU).
- An optional *Floating Point Unit* (FPU).

1.3 External interfaces

The processor incorporates three external bus interfaces, an ETM interface that allows the connection of an Embedded Trace Macrocell, an AHB Trace Macrocell interface that enables simple connection of an ETM to the processor, and an Advanced High-performance Bus Access Port (AHB-AP) interface for debug accesses.

The processor incorporates the following external interfaces:

- Multiple memory and device bus interfaces.
- ETM interface.
- Trace port interface.
- Debug port interface.

1.4 Configurable options

You can configure your Cortex-M4 implementation to include optional components, such as a Memory Protection Unit (MPU), a Flash Patch and Breakpoint Unit (FPB), and a Data Watchpoint and Trace Unit (DWT).

The following optional components can be configured for the Cortex-M4 processor:
- Memory Protection Unit (MPU) .
- Flash Patch and Breakpoint Unit (FPB).
- Data Watchpoint and Trace Unit (DWT).
- Instrumentation Trace Macrocell Unit (ITM).
- Embedded Trace Macrocell (ETM). See the*ETM-M4 Technical Reference Manual*.
- Advanced High-performance Bus Access Port (AHB-AP).
- AHB Trace Macrocell (HTM) interface.
- Trace Port Interface Unit (TPIU).
- Wake-up Interrupt Controller (WIC).
- Debug Port AHB-AP interface.
- Floating-Point Unit (FPU).
- Bit-banding.
- Constant AHB control.

Note

You can only configure trace functionality in the following combinations:
- No trace functionality.
- ITM and DWT.
- ITM, DWT, and ETM.
- ITM, DWT, ETM, and HTM.

You can configure the features provided in the DWT independently.

Related concepts

8.3 Flash Patch and Breakpoint Unit (FPB) on page 8-82.
8.2 AHB-AP debug access port on page 8-79.
2.3.3 AHB Trace Macrocell interface on page 2-26.
6.1.2 Low power modes on page 6-62.
2.3.4 Debug Port AHB-AP interface on page 2-26.
3.7 Bit-banding on page 3-45.
2.3.1 Bus interfaces on page 2-24.

Related references

Chapter 5 Memory Protection Unit on page 5-57.
Chapter 9 Data Watchpoint and Trace Unit on page 9-84.
Chapter 10 Instrumentation Trace Macrocell Unit on page 10-88.
Chapter 11 Trace Port Interface Unit on page 11-92.
Chapter 7 Floating-Point Unit on page 7-65.

1.5 Product documentation

Documentation provided with this product includes a Technical Reference Manual, an Integration and Implementation manual, together with design flow, architecture, and protocol information.

This section contains the following subsections:
- *1.5.1 Reference manuals* on page 1-16.
- *1.5.2 Design Flow* on page 1-17.
- *1.5.3 Architecture and protocol information* on page 1-17.

1.5.1 Reference manuals

This product is supplied with a complete set of reference manuals that describe processor functionality, build configuration options, and reference material that ARM partners might want to include in their own processor user guides.

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the Cortex-M4 processor. It is required at all stages of the design flow. Some behavior described in the TRM might not be relevant because of the way that the Cortex-M4 processor is implemented and integrated. If you are programming the Cortex-M4 processor then contact:
- The implementer to determine:
  - The build configuration of the implementation.
  - What integration, if any, was performed before implementing the processor.
- The integrator to determine the pin configuration of the SoC that you are using.

Integration and Implementation Manual

The *Integration and Implementation Manual* (IIM) describes:
- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- How to integrate the processor into a SoC. This includes a description of the integration kit and describes the pins that the integrator must tie off to configure the macrocell for the required integration.
- How to implement the processor into your design. This includes floorplanning guidelines, Design for Test (DFT) information, and how to perform netlist dynamic verification on the processor.
- The processes to sign off the integration and implementation of the design.

The ARM product deliverables include reference scripts and information about using them to implement your design.

Reference methodology documentation from your EDA tools vendor complements the IIM.

The IIM is a confidential book that is only available to licensees.

ETM-M4 Technical Reference Manual

The ETM-M4 TRM describes the functionality and behavior of the Cortex-M4 Embedded Trace Macrocell. It is required at all stages of the design flow. Typically the ETM-M4 is integrated with the Cortex-M4 processor prior to implementation as a single macrocell.

Cortex-M4 User Guide Reference Material

This document provides reference material that ARM partners can configure and include in a User Guide for an ARM Cortex-M4 processor. Typically:

- Each chapter in this reference material might correspond to a section in the User Guide.
- Each top-level section in this reference material might correspond to a chapter in the User Guide.

However, you can organize this material in any way, subject to the conditions of the license agreement under which ARM supplied the material.

1.5.2 Design Flow

The design flow of the processor includes steps for implementation, integration, and programming. These steps must be completed before the processor is ready for operation.

The processor is delivered as synthesizable RTL. Before it can be used in a product, it must go through the following process:

Implementation
The implementer configures and synthesizes the RTL.

Integration
The integrator connects the implemented design into a SoC. This includes connecting it to a memory system and peripherals.

Programming
The system programmer develops the software required to configure and initialize the processor, and tests the required application software.

Each stage in the process can be performed by a different party. Implementation and integration choices affect the behavior and features of the processor.

For MCUs, often a single design team integrates the processor before synthesizing the complete design. Alternatively, the team can synthesize the processor on its own or partially integrated, to produce a macrocell that is then integrated, possibly by a separate team.

The operation of the final device depends on:

Build configuration
The implementer chooses the options that affect how the RTL source files are pre-processed. These options usually include or exclude logic that affects one or more of the area, maximum frequency, and features of the resulting macrocell.

Configuration inputs
The integrator configures some features of the processor by tying inputs to specific values. These configurations affect the start-up behavior before any software configuration is made. They can also limit the options available to the software.

Software configuration
The programmer configures the processor by programming particular values into registers. This affects the behavior of the processor.

——— **Note** ———

This manual refers to implementation-defined features that are applicable to build configuration options. Reference to a feature that is included means that the appropriate build and pin configuration options are selected. Reference to an enabled feature means one that has also been configured by software.

————————

1.5.3 Architecture and protocol information

The processor complies with, or implements, specifications described in ARM, bus, debug, and other architecture reference manuals.

This book complements architecture reference manuals, architecture specifications, protocol specifications, and relevant external standards; it does not duplicate information from these sources.

ARM architecture

The processor implements the ARMv7E-M architecture profile.

See the *ARM®v7-M Architecture Reference Manual*.

Bus architecture

The processor implements an interface for CoreSight and other debug components using the AMBA 3 APB protocol.

The processor provides three primary bus interfaces implementing a variant of the AMBA 3 AHB-Lite protocol. See:
- The *ARM® AMBA® 3 AHB-Lite Protocol (v1.0)*.
- The *ARM® AMBA® 3 APB Protocol Specification*.

Debug

The debug features of the processor implement the ARM debug interface architecture.

See the *ARM® Debug Interface v5 Architecture Specification*.

Embedded Trace Macrocell

The trace features of the processor implement the ARM Embedded Trace Macrocell architecture.

See the *ARM® Embedded Trace Macrocell Architecture Specification*.

Floating Point Unit

The Cortex-M4 FPU implements ARMv7E-M architecture with FPv4-SP extensions.

It provides floating-point computation functionality that is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*. See the *ARM®v7M Architecture Reference Manual*.

Related references

Chapter 7 Floating-Point Unit on page 7-65.

1.6 Product revisions

A description of the differences in functionality between product revisions.

Differences in functionality between r0p0 and r0p1

In summary, the differences in functionality include:

- New implementation option to ensure constant AHB control during wait-stated transfers.

Chapter 2
Functional Description

This chapter introduces the Cortex-M4 processor and its external interfaces.

It contains the following sections:

- *2.1 About the functions* on page 2-21.
- *2.2 Processor features* on page 2-22.
- *2.3 Interfaces* on page 2-24.

2.1 About the functions

Block diagram showing the structure of the Cortex-M4 processor.

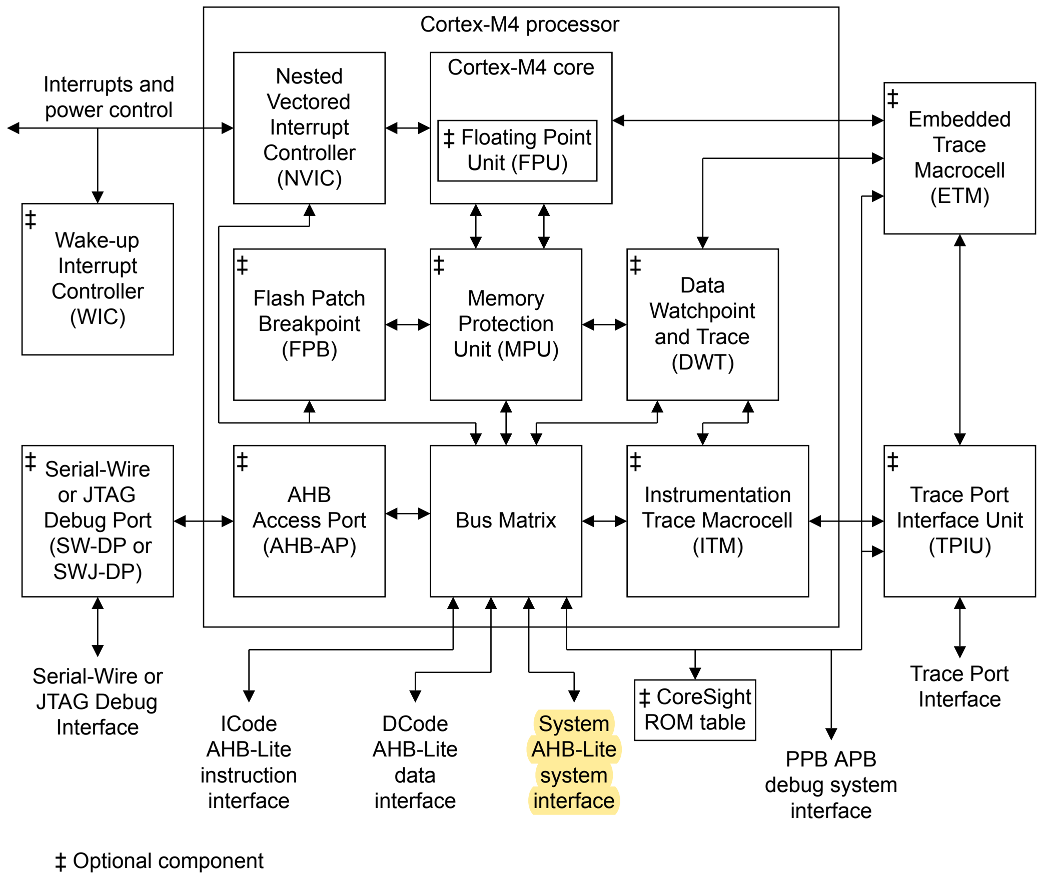

Cortex-M4 processor
Interrupts and power control
Nested Vectored Interrupt Controller (NVIC)
Cortex-M4 core
‡ Floating Point Unit (FPU)
‡ Embedded Trace Macrocell (ETM)
‡ Wake-up Interrupt Controller (WIC)
‡ Flash Patch Breakpoint (FPB)
‡ Memory Protection Unit (MPU)
‡ Data Watchpoint and Trace (DWT)
‡ Serial-Wire or JTAG Debug Port (SW-DP or SWJ-DP)
‡ AHB Access Port (AHB-AP)
Bus Matrix
‡ Instrumentation Trace Macrocell (ITM)
‡ Trace Port Interface Unit (TPIU)
Serial-Wire or JTAG Debug Interface
ICode AHB-Lite instruction interface
DCode AHB-Lite data interface
System AHB-Lite system interface
‡ CoreSight ROM table
PPB APB debug system interface
Trace Port Interface
‡ Optional component

Figure 2-1 Cortex-M4 block diagram

2.2 Processor features

The Cortex-M4 processor includes a low gate count processor core with low latency interrupt processing, an optional Floating Point Unit (FPU), a Nested Vectored Interrupt Controller (NVIC), and other features.

The complete processor features list comprises:

- A low gate count processor core, with low latency interrupt processing that has:
  - A subset of the Thumb instruction set, defined in the *ARM®v7-M Architecture Reference Manual*.
  - Banked *Stack Pointer* (SP).
  - Hardware integer divide instructions, `SDIV` and `UDIV`.
  - Handler and Thread modes.
  - Thumb and Debug states.
  - Support for interruptible-continued instructions `LDM`, `STM`, `PUSH`, and `POP` for low interrupt latency.
  - Automatic processor state saving and restoration for low latency *Interrupt Service Routine* (ISR) entry and exit.
  - Support for ARMv6 big-endian byte-invariant or little-endian accesses.
  - Support for ARMv6 unaligned accesses.
- Optional *Floating Point Unit* (FPU) providing:
  - 32-bit instructions for single-precision (C float) data-processing operations.
  - Combined Multiply and Accumulate instructions for increased precision (Fused MAC).
  - Hardware support for conversion, addition, subtraction, multiplication with optional accumulate, division, and square-root.
  - Hardware support for denormals and all IEEE rounding modes.
  - 32 dedicated 32-bit single-precision registers, also addressable as 16 double-word registers.
  - Decoupled three stage pipeline.
- *Nested Vectored Interrupt Controller* (NVIC) closely integrated with the processor core to achieve low-latency interrupt processing. Features include:
  - External interrupts, configurable from 1 to 240.
  - Bits of priority, configurable from 3 to 8.
  - Dynamic reprioritization of interrupts.
  - Priority grouping. This enables selection of preempting interrupt levels and non preempting interrupt levels.
  - Support for tail-chaining and late arrival of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
  - Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.
  - Optional *Wake-up Interrupt Controller* (WIC), providing ultra-low-power sleep mode support.
- *Memory Protection Unit* (MPU). An optional MPU for memory protection, including:
  - Eight memory regions.
  - *Sub Region Disable* (SRD), enabling efficient use of memory regions.
  - The ability to enable a background region that implements the default memory map attributes.
- Bus interfaces:
  - Three *Advanced High-performance Bus-Lite* (AHB-Lite) interfaces: ICode, DCode, and System bus interfaces.
  - *Private Peripheral Bus* (PPB) based on *Advanced Peripheral Bus* (APB) interface.
  - Bit-band support that includes atomic bit-band write and read operations.
  - Memory access alignment.
  - Write buffer for buffering of write data.
  - Exclusive access transfers for multiprocessor systems.
- Low-cost debug solution that features:

— Debug access to all memory and registers in the system, including access to memory-mapped devices, access to internal core registers when the core is halted, and access to debug control registers even while **SYSRESETn** is asserted.
— *Serial Wire Debug Port* (SW-DP) or *Serial Wire JTAG Debug Port* (SWJ-DP) debug access.
— Optional *Flash Patch and Breakpoint* (FPB) unit for implementing breakpoints and code patches.
— Optional *Data Watchpoint and Trace* (DWT) unit for implementing watchpoints, data tracing, and system profiling.
— Optional *Instrumentation Trace Macrocell* (ITM) for support of `printf()` style debugging.
— Optional *Trace Port Interface Unit* (TPIU) for bridging to a *Trace Port Analyzer* (TPA), including *Single Wire Output* (SWO) mode.
— Optional *Embedded Trace Macrocell* (ETM) for instruction trace.

2.3 Interfaces

The processor incorporates three external bus interfaces, an ETM interface that allows the connection of an Embedded Trace Macrocell, an AHB Trace Macrocell interface that enables simple connection of an ETM to the processor, and an Advanced High-performance Bus Access Port (AHB-AP) interface for debug accesses.

This section contains the following subsections:

- *2.3.1 Bus interfaces* on page 2-24.
- *2.3.2 ETM interface* on page 2-25.
- *2.3.3 AHB Trace Macrocell interface* on page 2-26.
- *2.3.4 Debug Port AHB-AP interface* on page 2-26.

2.3.1 Bus interfaces

The Cortex-M4 processor contains three external Advanced High-performance Bus (AHB)-Lite bus interfaces and one Advanced Peripheral Bus (APB) interface.

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 implement the design with the AHB_CONST_CTRL parameter set to 1. This ensures that when transfers are issued during a wait-stated response they are never retracted or modified and the original transfer is honored. The consequence of setting this parameter is that the performance of the core might decrease for wait-stated systems as a result of the interrupt and branch latency increasing.

ICode memory interface

Instruction fetches from Code memory space, `0x00000000` to `0x1FFFFFFC`, are performed over the 32-bit AHB-Lite bus.

The Debugger cannot access this interface. All fetches are word-wide. The number of instructions fetched per word depends on the code running and the alignment of the code in memory.

DCode memory interface

Data and debug accesses to Code memory space, `0x00000000` to `0x1FFFFFFF`, are performed over the 32-bit AHB-Lite bus.

The Code memory space available is dependent on the implementation. Core data accesses have a higher priority than debug accesses on this bus. This means that debug accesses are waited until core accesses have completed when there are simultaneous core and debug access to this bus.

Control logic in this interface converts unaligned data and debug accesses into two or three aligned accesses, depending on the size and alignment of the unaligned access. This stalls any subsequent data or debug access until the unaligned access has completed.

——— **Note** ———

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

———————

System interface

Instruction fetches and data and debug accesses to address ranges `0x20000000` to `0xDFFFFFFF` and `0xE0100000` to `0xFFFFFFFF` are performed over the 32-bit AHB-Lite bus.

For simultaneous accesses to the 32-bit AHB-Lite bus, the arbitration order in decreasing priority is:
- Data accesses.
- Instruction and vector fetches.
- Debug.

The system bus interface contains control logic to handle unaligned accesses, FPB remapped accesses, bit-band accesses, and pipelined instruction fetches.

Private Peripheral Bus (PPB)

Data and debug accesses to external PPB space, `0xE0040000` to `0xE00FFFFF`, are performed over the 32-bit Advanced Peripheral Bus (APB) bus. The *Trace Port Interface Unit* (TPIU) and vendor specific peripherals are on this bus.

Core data accesses have higher priority than debug accesses, so debug accesses are waited until core accesses have completed when there are simultaneous core and debug accesses to this bus. Only the address bits necessary to decode the External PPB space are supported on this interface.

The External PPB (EPPB) space, `0xE0040000` up to `0xE0100000`, is intended for CoreSight-compatible debug and trace components, and has a number of irregular limitations that make it less useful for regular system peripherals. ARM recommends that system peripherals are placed in suitable Device type areas of the System bus address space, with use of an AHB2APB protocol converter for APB-based devices.

Limitations of the EPPB space are:

- It is accessible in privileged mode only.
- It is accessed in little-endian fashion irrespective of the data endianness setting of the processor.
- Accesses behave as Strongly Ordered.
- Unaligned accesses have UNPREDICTABLE results.
- Only 32-bit data accesses are supported.
- It is accessible from the Debug Port and the local processor, but not from any other processor in the system.

2.3.2 ETM interface

The ETM interface enables simple connection of the ETM-M4 to the processor, and provides a channel for instruction trace to the ETM.

See the *ARM® Embedded Trace Macrocell Architecture Specification*.

2.3.3 AHB Trace Macrocell interface

The AHB Trace Macrocell (HTM) interface enables a simple connection of the AHB trace macrocell to the processor, and provides a channel for the data trace to the HTM.

Your implementation must include this interface to use the HTM interface. You must set TRCENA to 1 in the *Debug Exception and Monitor Control Register* (DEMCR) before you enable the HTM port to supply trace data. See the *ARM®v7-M Architecture Reference Manual*.

2.3.4 Debug Port AHB-AP interface

The processor contains an Advanced High-performance Bus Access Port (AHB-AP) interface for debug accesses. An external Debug Port (DP) component accesses this interface.

The Cortex-M4 system supports three possible DP implementations:

- The *Serial Wire JTAG Debug Port* (SWJ-DP). The SWJ-DP is a standard CoreSight debug port that combines JTAG-DP and *Serial Wire Debug Port* (SW-DP).
- The SW-DP. This provides a two-pin interface to the AHB-AP port.
- No DP present. If no debug functionality is present within the processor, a DP is not required.

The two DP implementations provide different mechanisms for debug access to the processor. Your implementation must contain only one of these components.

Note

Your implementation might contain an alternative implementer-specific DP instead of SW-DP or SWJ-DP. See your implementer for details.

For more detailed information on the DP components, see the *CoreSight™ Components Technical Reference manual*.

The DP and AP together are referred to as the *Debug Access Port* (DAP).

For more detailed information on the debug interface, see the *ARM® Debug Interface v5 Architecture Specification*.

Related references

Chapter 8 Debug on page 8-73.

Chapter 3
Programmers' Model

This chapter describes the Cortex-M4 processor programmers' model.

It contains the following sections:

- *3.1 About the programmers' model* on page 3-28.
- *3.2 Modes of operation and execution* on page 3-29.
- *3.3 Instruction set summary* on page 3-30.
- *3.4 Processor memory model* on page 3-40.
- *3.5 Write buffer* on page 3-43.
- *3.6 Exclusive monitor* on page 3-44.
- *3.7 Bit-banding* on page 3-45.
- *3.8 Processor core register summary* on page 3-47.
- *3.9 Exceptions* on page 3-49.

3.1 About the programmers' model

The Cortex-M4 programmers' model describes the processor's implementation-defined options.

For a complete description of the programmers' model, refer to the *ARM®v7-M Architecture Reference Manual*, which also contains the ARMv7-M Thumb instructions the model uses, and their cycle counts for the processor. In addition, other options of the programmers' model are described in the System Control, MPU, NVIC, FPU, Debug, DWT, ITM, and TPIU features topics.

Related references

Chapter 4 System Control on page 4-51.
Chapter 5 Memory Protection Unit on page 5-57.
Chapter 6 Nested Vectored Interrupt Controller on page 6-61.
Chapter 7 Floating-Point Unit on page 7-65.
Chapter 8 Debug on page 8-73.
Chapter 9 Data Watchpoint and Trace Unit on page 9-84.
Chapter 10 Instrumentation Trace Macrocell Unit on page 10-88.
Chapter 11 Trace Port Interface Unit on page 11-92.

3.2 Modes of operation and execution

The Cortex-M4 processor supports Thread and Handler operating modes, and may be run in Thumb or Debug operating states. In addition, the processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode.

See the *ARM®v7-M Architecture Reference Manual* for more information about these modes of operation and execution.

Operating modes

The conditions which cause the processor to enter Thread or Handler mode are as follows:

- The processor enters Thread mode on Reset, or as a result of an exception return. Privileged and Unprivileged code can run in Thread mode.
- The processor enters Handler mode as a result of an exception. All code is privileged in Handler mode.

Operating states

The processor can operate in thumb or debug state:

- Thumb state. This is normal execution running 16-bit and 32-bit halfword aligned Thumb instructions.
- Debug State. This is the state when the processor is in halting debug.

Privileged access and user access

Handler mode is always privileged. Thread mode can be privileged or unprivileged.

3.3 Instruction set summary

The processor implements the ARMv7-M Thumb instruction set, and is binary compatible with the instruction sets and features implemented in other Cortex-M profile processors. Instructions can be paired in a way that achieves optimum reductions in timing.

This section contains the following subsections:
- *3.3.1 Table of processor instructions* on page 3-30.
- *3.3.2 Table of processor DSP instructions* on page 3-35.
- *3.3.3 Load/store timings* on page 3-37.
- *3.3.4 Binary compatibility with other Cortex processors* on page 3-38.

3.3.1 Table of processor instructions

The table summarizes the Cortex-M4 processor instruction set. For brevity, not all load and store addressing modes are shown in the table. The cycle counts provided are based on a system with zero wait states.

Within the assembler syntax, depending on the operation, the `<op2>` field can be replaced with one of the following options:
- A simple register specifier, for example `Rm`.
- An immediate shifted register, for example `Rm, LSL #4`.
- A register shifted register, for example `Rm, LSL Rs`.
- An immediate value, for example `#0xE000E000`.

For brevity, not all load and store addressing modes are shown. See the *ARMv7-M Architecture Reference Manual* for more information.

The following abbreviations are used in the Cycles column:

P
The number of cycles required for a pipeline refill. This ranges from 1 to 3 depending on the alignment and width of the target instruction, and whether the processor manages to speculate the address early.

B
The number of cycles required to perform the barrier operation. For `DSB` and `DMB`, the minimum number of cycles is zero. For `ISB`, the minimum number of cycles is equivalent to the number required for a pipeline refill.

N
The number of registers in the register list to be loaded or stored, including PC or LR.

W
The number of cycles spent waiting for an appropriate event.

Table 3-1 Processor instruction set summary

Operation	Description	Assembler	Cycles	Notes
Move	Register	`MOV Rd, <op2>`	1	
	16-bit immediate	`MOVW Rd, #<imm>`	1	
	Immediate into top	`MOVT Rd, #<imm>`	1	
	To PC	`MOV PC, Rm`	1 + P	

Table 3-1 Processor instruction set summary (continued)

Operation	Description	Assembler	Cycles	Notes
Add	Add	`ADD Rd, Rn, <op2>`	1	
	Add to PC	`ADD PC, PC, Rm`	1 + P	
	Add with carry	`ADC Rd, Rn, <op2>`	1	
	Form address	`ADR Rd, <label>`	1	
Subtract	Subtract	`SUB Rd, Rn, <op2>`	1	
	Subtract with borrow	`SBC Rd, Rn, <op2>`	1	
	Reverse	`RSB Rd, Rn, <op2>`	1	
Multiply	Multiply	`MUL Rd, Rn, Rm`	1	
	Multiply accumulate	`MLA Rd, Rn, Rm`	1	
	Multiply subtract	`MLS Rd, Rn, Rm`	1	
	Long signed	`SMULL RdLo, RdHi, Rn, Rm`	1	
	Long unsigned	`UMULL RdLo, RdHi, Rn, Rm`	1	
	Long signed accumulate	`SMLAL RdLo, RdHi, Rn, Rm`	1	
	Long unsigned accumulate	`UMLAL RdLo, RdHi, Rn, Rm`	1	
Divide	Signed	`SDIV Rd, Rn, Rm`	2 to 12	Division operations terminate when the divide calculation completes, with the number of cycles required dependent on the values of the input operands. Division operations are interruptible, meaning that an operation can be abandoned when an interrupt occurs, with worst case latency of one cycle, and restarted when the interrupt completes.
	Unsigned	`UDIV Rd, Rn, Rm`	2 to 12	
Saturate	Signed	`SSAT Rd, #<imm>, <op2>`	1	
	Unsigned	`USAT Rd, #<imm>, <op2>`	1	
Compare	Compare	`CMP Rn, <op2>`	1	
	Negative	`CMN Rn, <op2>`	1	
Logical	AND	`AND Rd, Rn, <op2>`	1	
	Exclusive OR	`EOR Rd, Rn, <op2>`	1	
	OR	`ORR Rd, Rn, <op2>`	1	
	OR NOT	`ORN Rd, Rn, <op2>`	1	
	Bit clear	`BIC Rd, Rn, <op2>`	1	
	Move NOT	`MVN Rd, <op2>`	1	
	AND test	`TST Rn, <op2>`	1	
	Exclusive OR test	`TEQ Rn, <op1>`		

Table 3-1 Processor instruction set summary (continued)

Operation	Description	Assembler	Cycles	Notes
Shift	Logical shift left	`LSL Rd, Rn, #<imm>`	1	
	Logical shift left	`LSL Rd, Rn, Rs`	1	
	Logical shift right	`LSR Rd, Rn, #<imm>`	1	
	Logical shift right	`LSR Rd, Rn, Rs`	1	
	Arithmetic shift right	`ASR Rd, Rn, #<imm>`	1	
	Arithmetic shift right	`ASR Rd, Rn, Rs`	1	
Rotate	Rotate right	`ROR Rd, Rn, #<imm>`	1	
	Rotate right	`ROR Rd, Rn, Rs`	1	
	With extension	`RRX Rd, Rn`	1	
Count	Leading zeroes	`CLZ Rd, Rn`	1	
Load	Word	`LDR Rd, [Rn, <op2>]`	2	Neighboring load and store single instructions can pipeline their address and data phases but in some cases, such as 32-bit opcodes aligned on odd halfword boundaries, they might not pipeline optimally.
	To PC	`LDR PC, [Rn, <op2>]`	2 + P	Conditional branch completes in a single cycle if the branch is not taken.
	Halfword	`LDRH Rd, [Rn, <op2>]`	2	
	Byte	`LDRB Rd, [Rn, <op2>]`	2	
	Signed halfword	`LDRSH Rd, [Rn, <op2>]`	2	
	Signed byte	`LDRSB Rd, [Rn, <op2>]`	2	
	User word	`LDRT Rd, [Rn, #<imm>]`	2	
	User halfword	`LDRHT Rd, [Rn, #<imm>]`	2	
	User byte	`LDRBT Rd, [Rn, #<imm>]`	2	
	User signed halfword	`LDRSHT Rd, [Rn, #<imm>]`	2	
	User signed byte	`LDRSBT Rd, [Rn, #<imm>]`	2	
	PC relative	`LDR Rd,[PC, #<imm>]`	2	
	Doubleword	`LDRD Rd, Rd, [Rn, #<imm>]`	1 + N	
	Multiple	`LDM Rn, {<reglist>}`	1 + N	
	Multiple including PC	`LDM Rn, {<reglist>, PC}`	1 + N + P	

Table 3-1 Processor instruction set summary (continued)

Operation	Description	Assembler	Cycles	Notes
Store	Word	`STR Rd, [Rn, <op2>]`	2	Conditional branch completes in a single cycle if the branch is not taken.
	Halfword	`STRH Rd, [Rn, <op2>]`	2	
	Byte	`STRB Rd, [Rn, <op2>]`	2	
	Signed halfword	`STRSH Rd, [Rn, <op2>]`	2	
	Signed byte	`STRSB Rd, [Rn, <op2>]`	2	
	User word	`STRT Rd, [Rn, #<imm>]`	2	
	User halfword	`STRHT Rd, [Rn, #<imm>]`	2	
	User byte	`STRBT Rd, [Rn, #<imm>]`	2	
	User signed halfword	`STRSHT Rd, [Rn, #<imm>]`	2	
	User signed byte	`STRSBT Rd, [Rn, #<imm>]`	2	
	Doubleword	`STRD Rd, Rd, [Rn, #<imm>]`	1 + N	
	Multiple	`STM Rn, {<reglist>}`	1 + N	
Push	Push	`PUSH {<reglist>}`	1 + N	
	Push with link register	`PUSH {<reglist>, LR}`	1 + N	
Pop	Pop	`POP {<reglist>}`	1 + N	
	Pop and return	`POP {<reglist>, PC}`	1 + N + P	
Semaphore	Load exclusive	`LDREX Rd, [Rn, #<imm>]`	2	
	Load exclusive half	`LDREXH Rd, [Rn]`	2	
	Load exclusive byte	`LDREXB Rd, [Rn]`	2	
	Store exclusive	`STREX Rd, Rt, [Rn, #<imm>]`	2	
	Store exclusive half	`STREXH Rd, Rt, [Rn]`	2	
	Store exclusive byte	`STREXB Rd, Rt, [Rn]`	2	
	Clear exclusive monitor	`CLREX`	1	
Branch	Conditional	`B<cc> <label>`	1 or 1 + P	Conditional branch completes in a single cycle if the branch is not taken.
	Unconditional	`B <label>`	1 + P	
	With link	`BL <label>`	1 + P	
	With exchange	`BX Rm`	1 + P	
	With link and exchange	`BLX Rm`	1 + P	
	Branch if zero	`CBZ Rn, <label>`	1 or 1 + P	An `IT` instruction can be folded onto a preceding 16-bit Thumb instruction, enabling execution in zero cycles
	Branch if non-zero	`CBNZ Rn, <label>`	1 or 1 + P	
	Byte table branch	`TBB [Rn, Rm]`	2 + P	
	Halfword table branch	`TBH [Rn, Rm, LSL#1]`	2 + P	

Table 3-1 Processor instruction set summary (continued)

Operation	Description	Assembler	Cycles	Notes
State change	Supervisor call	`SVC #<imm>`	-	
	If-then-else	`IT... <cond>`	1	An `IT` instruction can be folded onto a preceding 16-bit Thumb instruction, enabling execution in zero cycles
	Disable interrupts	`CPSID <flags>`	1 or 2	
	Enable interrupts	`CPSIE <flags>`	1 or 2	
	Read special register	`MRS Rd, <specreg>`	1 or 2	
	Write special register	`MSR <specreg>, Rn`	1 or 2	
	Breakpoint	`BKPT #<imm>`	-	
Extend	Signed halfword to word	`SXTH Rd, <op2>`	1	
	Signed byte to word	`SXTB Rd, <op2>`	1	
	Unsigned halfword	`UXTH Rd, <op2>`	1	
	Unsigned byte	`UXTB Rd, <op2>`	1	
Bit field	Extract unsigned	`UBFX Rd, Rn, #<imm>, #<imm>`	1	
	Extract signed	`SBFX Rd, Rn, #<imm>, #<imm>`	1	
	Clear	`BFC Rd, Rn, #<imm>, #<imm>`	1	
	Insert	`BFI Rd, Rn, #<imm>, #<imm>`	1	
Reverse	Bytes in word	`REV Rd, Rm`	1	
	Bytes in both halfwords	`REV16 Rd, Rm`	1	
	Signed bottom halfword	`REVSH Rd, Rm`	1	
	Bits in word	`RBIT Rd, Rm`	1	
Hint	Send event	`SEV`	1	
	Wait for event	`WFE`	1 + W	
	Wait for interrupt	`WFI`	1 + W	
	No operation	`NOP`	1	
Barriers	Instruction synchronization	`ISB`	1 + B	
	Data memory	`DMB`	1 + B	
	Data synchronization	`DSB <flags>`	1 + B	

3.3.2 Table of processor DSP instructions

The table summarizes the Cortex-M4 DSP instruction set.

Table 3-2 Cortex-M4 DSP instruction set summary

Operation	Description	Assembler	Cycles
Multiply	32-bit multiply with 32-most-significant-bit accumulate	`SMMLA`	1
	32-bit multiply with 32-most-significant-bit subtract	`SMMLS`	1
	32-bit multiply returning 32-most-significant-bits	`SMMUL`	1
	32-bit multiply with rounded 32-most-significant-bit accumulate	`SMMLAR`	1
	32-bit multiply with rounded 32-most-significant-bit subtract	`SMMLSR`	1
	32-bit multiply returning rounded 32-most-significant-bits	`SMMULR`	1
Signed Multiply	Q setting 16-bit signed multiply with 32-bit accumulate, bottom by bottom	`SMLABB`	1
	Q setting 16-bit signed multiply with 32-bit accumulate, bottom by top	`SMLABT`	1
	16-bit signed multiply with 64-bit accumulate, bottom by bottom	`SMLALBB`	1
	16-bit signed multiply with 64-bit accumulate, bottom by top	`SMLALBT`	1
	Dual 16-bit signed multiply with single 64-bit accumulator	`SMLALD{X}`	1
	16-bit signed multiply with 64-bit accumulate, top by bottom	`SMLALTB`	1
	16-bit signed multiply with 64-bit accumulate, top by top	`SMLALTT`	1
	16-bit signed multiply yielding 32-bit result, bottom by bottom	`SMULBB`	1
	16-bit signed multiply yielding 32-bit result, bottom by top	`SMULBT`	1
	16-bit signed multiply yielding 32-bit result, top by bottom	`SMULTB`	1
	16-bit signed multiply yielding 32-bit result, top by top	`SMULTT`	1
	16-bit by 32-bit signed multiply returning 32-most-significant-bits, bottom	`SMULWB`	1
	16-bit by 32-bit signed multiply returning 32-most-significant-bits, top	`SMULWT`	1
	Dual 16-bit signed multiply returning difference	`SMUSD{X}`	1
	Q setting 16-bit signed multiply with 32-bit accumulate, top by bottom	`SMLATB`	1
	Q setting 16-bit signed multiply with 32-bit accumulate, top by top	`SMLATT`	1
	Q setting dual 16-bit signed multiply with single 32-bit accumulator	`SMLAD{X}`	1
	Q setting 16-bit by 32-bit signed multiply with 32-bit accumulate, bottom	`SMLAWB`	1
	Q setting 16-bit by 32-bit signed multiply with 32-bit accumulate, top	`SMLAWT`	1
	Q setting dual 16-bit signed multiply subtract with 32-bit accumulate	`SMLSD{X}`	1
	Q setting dual 16-bit signed multiply subtract with 64-bit accumulate	`SMLSLD{X}`	1
	Q setting sum of dual 16-bit signed multiply	`SMUAD{X}`	1
Unsigned Multiply	32-bit unsigned multiply with double 32-bit accumulation yielding 64-bit result	`UMAAL`	1
Saturate	Q setting dual 16-bit saturate	`SSAT16`	1
	Q setting dual 16-bit unsigned saturate	`USAT16`	1

Table 3-2 Cortex-M4 DSP instruction set summary (continued)

Operation	Description	Assembler	Cycles
Packing and Unpacking	Pack half word top with shifted bottom	PKHTB	1
	Pack half word bottom with shifted top	PKHBT	1
	Extract 8-bits and sign extend to 32-bits	SXTB	1
	Dual extract 8-bits and sign extend each to 16-bits	SXTB16	1
	Extract 16-bits and sign extend to 32-bits	SXTH	1
	Extract 8-bits and zero-extend to 32-bits	UXTB	1
	Dual extract 8-bits and zero-extend to 16-bits	UXTB16	1
	Extract 16-bits and zero-extend to 32-bits	UXTH	1
	Extract 8-bit to 32-bit unsigned addition	UXTAB	1
	Dual extracted 8-bit to 16-bit unsigned addition	UXTAB16	1
	Extracted 16-bit to 32-bit unsigned addition	UXTAH	1
	Extracted 8-bit to 32-bit signed addition	SXTAB	1
	Dual extracted 8-bit to 16-bit signed addition	SXTAB16	1
	Extracted 16-bit to 32-bit signed addition	SXTAH	1
Miscellaneous Data Processing	Select bytes based on GE bits	SEL	1
	Unsigned sum of quad 8-bit unsigned absolute difference	USAD8	1
	Unsigned sum of quad 8-bit unsigned absolute difference with 32-bit accumulate	USADA8	1
Addition	Dual 16-bit unsigned saturating addition	UQADD16	1
	Quad 8-bit unsigned saturating addition	UQADD8	1
	Q setting saturating add	QADD	1
	Q setting dual 16-bit saturating add	QADD16	1
	Q setting quad 8-bit saturating add	QADD8	1
	Q setting saturating double and add	QDADD	1
	GE setting quad 8-bit signed addition	SADD8	1
	GE setting dual 16-bit signed addition	SADD16	1
	Dual 16-bit signed addition with halved results	SHADD16	1
	Quad 8-bit signed addition with halved results	SHADD8	1
	GE setting dual 16-bit unsigned addition	UADD16	1
	GE setting quad 8-bit unsigned addition	UADD8	1
	Dual 16-bit unsigned addition with halved results	UHADD16	1
	Quad 8-bit unsigned addition with halved results	UHADD8	1

Table 3-2 Cortex-M4 DSP instruction set summary (continued)

Operation	Description	Assembler	Cycles
Subtraction	Q setting saturating double and subtract	`QDSUB`	1
	Dual 16-bit unsigned saturating subtraction	`UQSUB16`	1
	Quad 8-bit unsigned saturating subtraction	`UQSUB8`	1
	Q setting saturating subtract	`QSUB`	1
	Q setting dual 16-bit saturating subtract	`QSUB16`	1
	Q setting quad 8-bit saturating subtract	`QSUB8`	1
	Dual 16-bit signed subtraction with halved results	`SHSUB16`	1
	Quad 8-bit signed subtraction with halved results	`SHSUB8`	1
	GE setting dual 16-bit signed subtraction	`SSUB16`	1
	GE setting quad 8-bit signed subtraction	`SSUB8`	1
	Dual 16-bit unsigned subtraction with halved results	`UHSUB16`	1
	Quad 8-bit unsigned subtraction with halved results	`UHSUB8`	1
	GE setting dual 16-bit unsigned subtract	`USUB16`	1
	GE setting quad 8-bit unsigned subtract	`USUB8`	1
Parallel Addition and Subtraction	Dual 16-bit unsigned saturating addition and subtraction with exchange	`UQASX`	1
	Dual 16-bit unsigned saturating subtraction and addition with exchange	`UQSAX`	1
	GE setting dual 16-bit addition and subtraction with exchange	`SASX`	1
	Q setting dual 16-bit add and subtract with exchange	`QASX`	1
	Q setting dual 16-bit subtract and add with exchange	`QSAX`	1
	Dual 16-bit signed addition and subtraction with halved results	`SHASX`	1
	Dual 16-bit signed subtraction and addition with halved results	`SHSAX`	1
	GE setting dual 16-bit signed subtraction and addition with exchange	`SSAX`	1
	GE setting dual 16-bit unsigned addition and subtraction with exchange	`UASX`	1
	Dual 16-bit unsigned addition and subtraction with halved results and exchange	`UHASX`	1
	Dual 16-bit unsigned subtraction and addition with halved results and exchange	`UHSAX`	1
	GE setting dual 16-bit unsigned subtract and add with exchange	`USAX`	1

3.3.3 Load/store timings

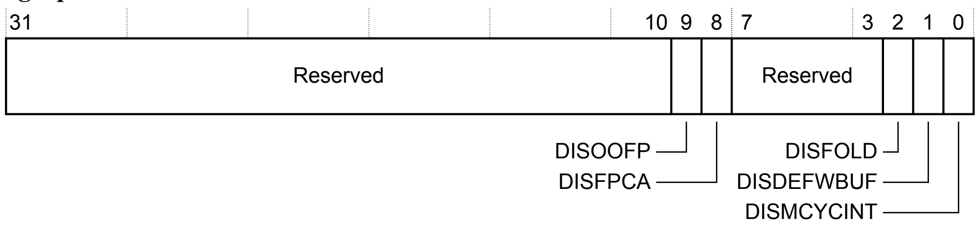
Instructions can be optimally paired to achieve more reductions in load and store timings.

The following information may help you to achieve further reductions in timing when pairing instructions:

- `STR Rx,[Ry,#imm]` is always one cycle. This is because the address generation is performed in the initial cycle, and the data store is performed at the same time as the next instruction is executing. If the store is to the write buffer, and the write buffer is full or not enabled, the next instruction is delayed until the store can complete. If the store is not to the write buffer, for example to the Code segment, and that transaction stalls, the impact on timing is only felt if another load or store operation is executed before completion.

- `LDR PC,[any]` is always a blocking operation. This means at least two cycles for the load, and three cycles for the pipeline reload. So this operation takes at least five cycles, or more if stalled on the load or the fetch.
- Any load or store that generates an address dependent on the result of a preceding data processing operation stalls the pipeline for an additional cycle while the register bank is updated. There is no forwarding path for this scenario.
- `LDR Rx,[PC,#imm]` might add a cycle because of contention with the fetch unit.
- `TBB` and `TBH` are also blocking operations. These are at least two cycles for the load, one cycle for the add, and three cycles for the pipeline reload. This means at least six cycles, or more if stalled on the load or the fetch.
- `LDR [any]` are pipelined when possible. This means that if the next instruction is an `LDR` or `STR`, and the destination of the first `LDR` is not used to compute the address for the next instruction, then one cycle is removed from the cost of the next instruction. So, an `LDR` might be followed by an `STR`, so that the `STR` writes out what the `LDR` loaded. More multiple `LDR`s can be pipelined together. Some optimized examples are:
  - — `LDR R0,[R1]; LDR R1,[R2]` - normally three cycles total.
  - — `LDR R0,[R1,R2]; STR R0,[R3,#20]` - normally three cycles total.
  - — `LDR R0,[R1,R2]; STR R1,[R3,R2]` - normally three cycles total.
  - — `LDR R0,[R1,R5]; LDR R1,[R2]; LDR R2,[R3,#4]` - normally four cycles total.
- Other instructions cannot be pipelined after `STR` with register offset. `STR` can only be pipelined when it follows an `LDR`, but nothing can be pipelined after the store. Even a stalled `STR` normally only takes two cycles, because of the write buffer.
- `LDREX` and `STREX` can be pipelined exactly as `LDR`. Because `STREX` is treated more like an `LDR`, it can be pipelined as explained for `LDR`. Equally `LDREX` is treated exactly as an `LDR` and so can be pipelined.
- `LDRD` and `STRD` cannot be pipelined with preceding or following instructions. However, the two words are pipelined together. So, this operation requires three cycles when not stalled.
- `LDM` and `STM` cannot be pipelined with preceding or following instructions. However, all elements after the first are pipelined together. So, a three element `LDM` takes 2+1+1 or 5 cycles when not stalled. Similarly, an eight element store takes nine cycles when not stalled. When interrupted, `LDM` and `STM` instructions continue from where they left off when returned to. The continue operation adds one or two cycles to the first element when started.
- Unaligned word or halfword loads or stores add penalty cycles. A byte aligned halfword load or store adds one extra cycle to perform the operation as two bytes. A halfword aligned word load or store adds one extra cycle to perform the operation as two halfwords. A byte-aligned word load or store adds two extra cycles to perform the operation as a byte, a halfword, and a byte. These numbers increase if the memory stalls. A `STR` or `STRH` cannot delay the processor because of the write buffer.

Example graphic



3.3.4 Binary compatibility with other Cortex processors

The processor implements a subset of the instruction set and features provided by the ARMv7-M architecture profile, and is binary compatible with the instruction sets and features implemented in other

Cortex-M profile processors. You can move software, including system level software, from the Cortex-M4 processor to other Cortex-M profile processors.

To ensure a smooth transition, ARM recommends that code designed to operate on other Cortex-M profile processor architectures obey the following rules and configure the *Configuration and Control Register* (CCR) appropriately:

- Use word transfers only to access registers in the NVIC and *System Control Space* (SCS).
- Treat all unused SCS registers and register fields on the processor as Do-Not-Modify.
- Configure the following fields in the CCR:
  — STKALIGN bit to 1.
  — UNALIGN_TRP bit to 1.
  — Leave all other bits in the CCR register as their original value.

3.4 Processor memory model

The processor contains a bus matrix that arbitrates accesses to both the external memory system and to the internal System Control Space (SCS) and debug components, supports ARMv7 unaligned accesses, and performs all accesses as single, unaligned accesses.

Priority is always given to the processor to ensure that any debug accesses are as non-intrusive as possible. For a zero wait state system, all debug accesses to system memory, SCS, and debug resources are completely non-intrusive.

See the ARMv7-M Architecture Reference Manual for more information about the memory model.

The following figure shows the system address map.

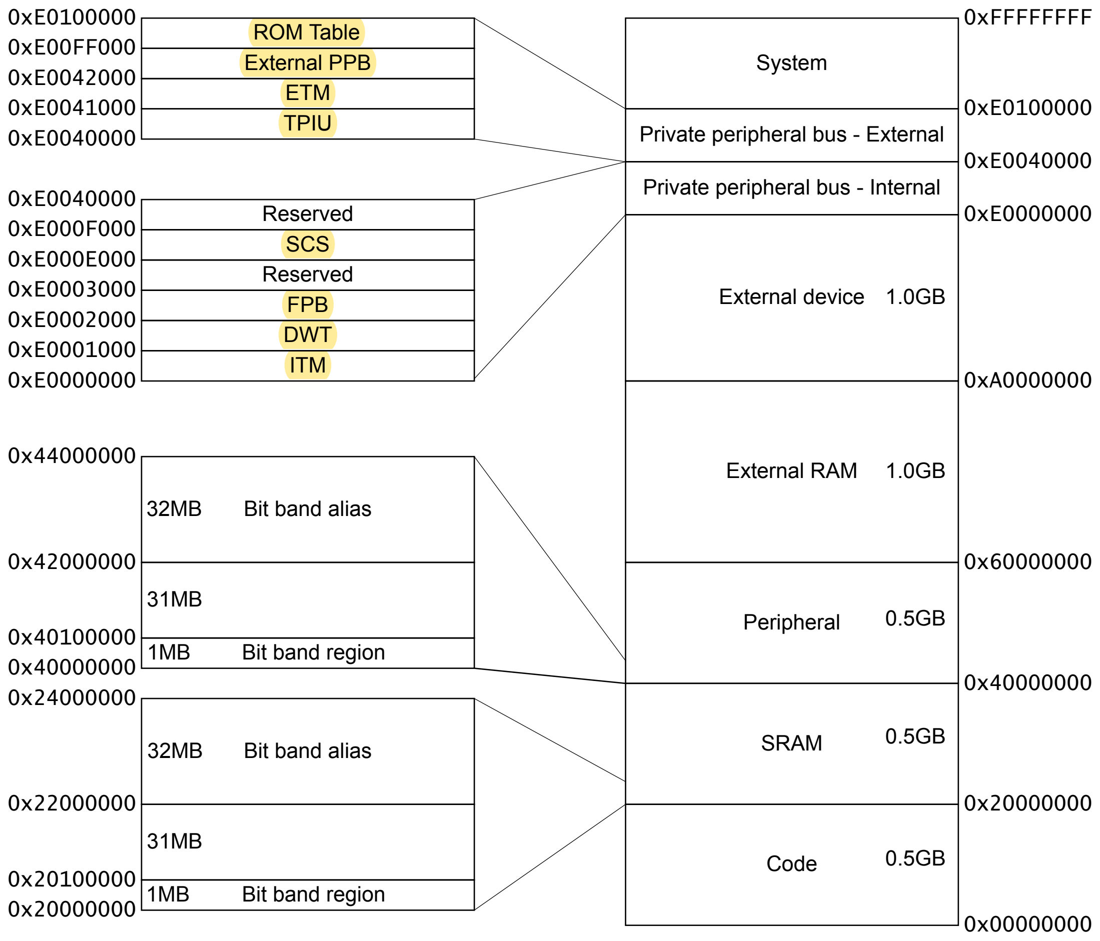


Figure 3-1 System address map

This section contains the following subsections:
- *3.4.1 Memory regions table* on page 3-41.
- *3.4.2 Private Peripheral Bus* on page 3-41.
- *3.4.3 Unaligned accesses that cross regions* on page 3-41.

3.4.1 Memory regions table

The table shows the processor interfaces that are addressed by the different memory map regions.

Table 3-3 Memory regions

Memory Map	Region
Code	Instruction fetches are performed over the ICode bus. Data accesses are performed over the DCode bus.
SRAM	Instruction fetches and data accesses are performed over the system bus.
SRAM bit-band	Alias region. Data accesses are aliases. Instruction accesses are not aliases.
Peripheral	Instruction fetches and data accesses are performed over the system bus.
Peripheral bit-band	Alias region. Data accesses are aliases. Instruction accesses are not aliases.
External RAM	Instruction fetches and data accesses are performed over the system bus.
External Device	Instruction fetches and data accesses are performed over the system bus.
Private Peripheral Bus	External and internal *Private Peripheral Bus* (PPB) interfaces. This memory region is *Execute Never* (XN), and so instruction fetches are prohibited. An MPU, if present, cannot change this.
System	System segment for vendor system peripherals. This memory region is XN, and so instruction fetches are prohibited. An MPU, if present, cannot change this.

3.4.2 Private Peripheral Bus

The Private Peripheral Bus (PPB) interface provides access to internal and external processor resources.

The internal *Private Peripheral Bus* (PPB) interface provides access to:
- The *Instrumentation Trace Macrocell* (ITM).
- The *Data Watchpoint and Trace* (DWT).
- The *Flashpatch and Breakpoint* (FPB).
- The *System Control Space* (SCS), including the *Memory Protection Unit* (MPU) and the *Nested Vectored Interrupt Controller* (NVIC).

The external PPB interface provides access to:
- The *Trace Point Interface Unit* (TPIU).
- The *Embedded Trace Macrocell* (ETM).
- The ROM table.
- Implementation-specific areas of the PPB memory map.

3.4.3 Unaligned accesses that cross regions

The Cortex-M4 processor supports ARMv7 unaligned accesses, and performs all accesses as single, unaligned accesses. They are converted into two or more aligned accesses by the DCode and System bus interfaces.

Note

All Cortex-M4 external accesses are aligned.

Unaligned support is only available for load/store singles (`LDR`, `LDRH`, `STR`, `STRH`). Load/store double already supports word aligned accesses, but does not permit other unaligned accesses, and generates a fault if this is attempted.

Unaligned accesses that cross memory map boundaries are architecturally UNPREDICTABLE. The processor behavior is boundary dependent, as follows:

- DCode accesses wrap within the region. For example, an unaligned halfword access to the last byte of Code space (`0x1FFFFFFF`) is converted by the DCode interface into a byte access to `0x1FFFFFFF` followed by a byte access to `0x00000000`.
- System accesses that cross into PPB space do not wrap within System space. For example, an unaligned halfword access to the last byte of System space (`0xDFFFFFFF`) is converted by the System interface into a byte access to `0xDFFFFFFF` followed by a byte access to `0xE0000000`. `0xE0000000` is not a valid address on the System bus.
- System accesses that cross into Code space do not wrap within System space. For example, an unaligned halfword access to the last byte of System space (`0xFFFFFFFF`) is converted by the System interface into a byte access to `0xFFFFFFFF` followed by a byte access to `0x00000000`. `0x00000000` is not a valid address on the System bus.
- Unaligned accesses are not supported to PPB space, and so there are no boundary crossing cases for PPB accesses.

Unaligned accesses that cross into the bit-band alias regions are also architecturally UNPREDICTABLE. The processor performs the access to the bit-band alias address, but this does not result in a bit-band operation. For example, an unaligned halfword access to `0x21FFFFFF` is performed as a byte access to `0x21FFFFFF` followed by a byte access to `0x22000000` (the first byte of the bit-band alias).

Unaligned loads that match against a literal comparator in the FPB are not remapped. FPB only remaps aligned addresses.

3.5 Write buffer

To prevent bus wait cycles from stalling the processor during data stores, buffered stores to the DCode and System buses go through a one-entry write buffer. If the write buffer is full, subsequent accesses to the bus stall until the write buffer has drained.

The write buffer is only used if the bus waits the data phase of the buffered store, otherwise the transaction completes on the bus.

`DMB` and `DSB` instructions wait for the write buffer to drain before completing. If an interrupt comes in while `DMB` or `DSB` is waiting for the write buffer to drain, the processor returns to the instruction following the `DMB` or `DSB` after the interrupt completes. This is because interrupt processing acts as a memory barrier operation.

3.6 Exclusive monitor

The Cortex-M4 processor implements a local exclusive monitor. The local monitor within the processor has been constructed so that it does not hold any physical address, but instead treats any access as matching the address of the previous LDREX. This means that the implemented exclusives reservation granule is the entire memory address range.

The Cortex-M4 processor does not support exclusive accesses to bit-band regions.

For more information about semaphores and the local exclusive monitor, see the *ARM®v7-M Architecture Reference Manual*.

3.7 Bit-banding

Bit-banding is an optional feature of the Cortex-M4 processor. Bit-banding maps a complete word of memory onto a single bit in the bit-band region. For example, writing to one of the alias words sets or clears the corresponding bit in the bit-band region.

This section contains the following subsections:

- *3.7.1 About bit-banding* on page 3-45.
- *3.7.2 Directly accessing an alias region* on page 3-46.
- *3.7.3 Directly accessing a bit-band region* on page 3-46.

3.7.1 About bit-banding

Bit-banding enables every individual bit in the bit-banding region to be directly accessible from a word-aligned address using a single `LDR` instruction. It also enables individual bits to be toggled without performing a read-modify-write sequence of instructions.

The processor memory map includes two bit-band regions. These occupy the lowest 1MB of the SRAM and Peripheral memory regions respectively. These bit-band regions map each word in an alias region of memory to a bit in a bit-band region of memory.

The System bus interface contains logic that controls bit-band accesses as follows:

- It remaps bit-band alias addresses to the bit-band region.
- For reads, it extracts the requested bit from the read byte, and returns this in the *Least Significant Bit* (LSB) of the read data returned to the core.
- For writes, it converts the write to an atomic read-modify-write operation.
- The processor does not stall during bit-band operations unless it attempts to access the System bus while the bit-band operation is being carried out.

The memory map has two 32MB alias regions that map to two 1MB bit-band regions:

- Accesses to the 32MB SRAM alias region map to the 1MB SRAM bit-band region.
- Accesses to the 32MB peripheral alias region map to the 1MB peripheral bit-band region.

A mapping formula shows how to reference each word in the alias region to a corresponding bit, or target bit, in the bit-band region. The mapping formula is:

```
bit_word_offset = (byte_offset × 32) + (bit_number × 4)
```

```
bit_word_addr = bit_band_base + bit_word_offset
```

where:

- `bit_word_offset` is the position of the target bit in the bit-band memory region.
- `bit_word_addr` is the address of the word in the alias memory region that maps to the targeted bit.
- `bit_band_base` is the starting address of the alias region.
- `byte_offset` is the number of the byte in the bit-band region that contains the targeted bit.
- `bit_number` is the bit position, 0 to 7, of the targeted bit.

The following figure shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at `0x23FFFFE0` maps to bit [0] of the bit-band byte at `0x200FFFFF`: `0x23FFFFE0` = `0x22000000` + (`0xFFFFF*32`) + `0*4`.
- The alias word at `0x23FFFFFC` maps to bit [7] of the bit-band byte at `0x200FFFFF`: `0x23FFFFFC` = `0x22000000` + (`0xFFFFF*32`) + `7*4`.
- The alias word at `0x22000000` maps to bit [0] of the bit-band byte at `0x20000000`: `0x22000000` = `0x22000000` + (`0*32`) + `0*4`.

- The alias word at `0x2200001C` maps to bit [7] of the bit-band byte at `0x20000000`: `0x2200001C` = `0x22000000` + (`0*32`) + `7*4`.

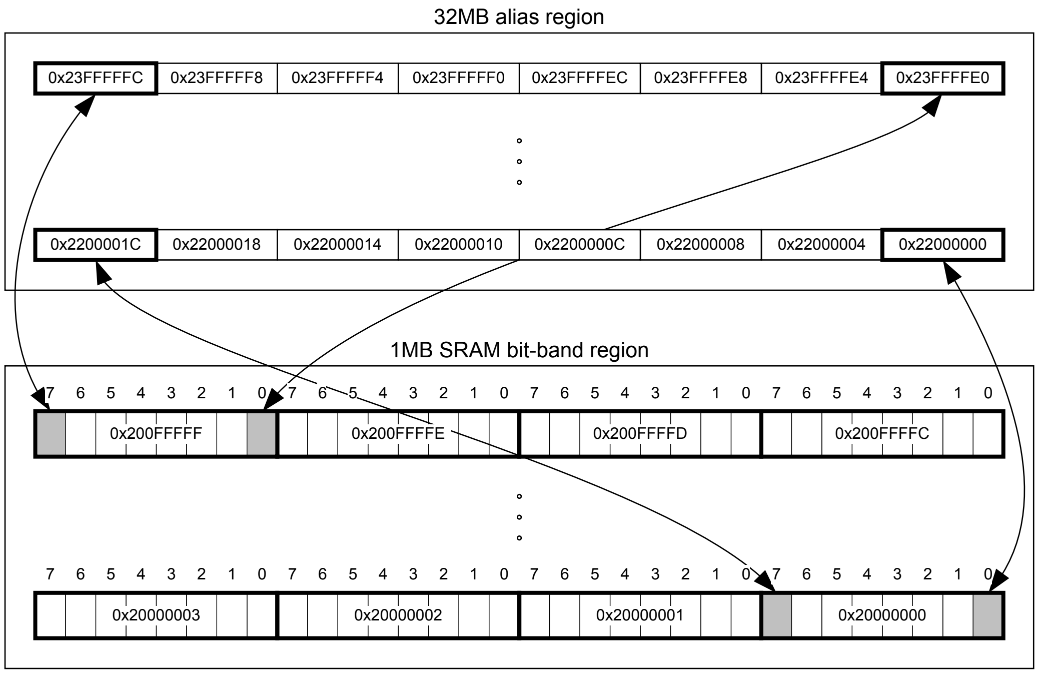


Figure 3-2 Bit-band mapping

3.7.2 Directly accessing an alias region

Writing to a word in the alias region has the same effect as a read-modify-write operation on the targeted bit in the bit-band region.

Bit [0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit [0] set writes a 1 to the bit-band bit, and writing a value with bit [0] cleared writes a 0 to the bit-band bit.

Bits [31:1] of the alias word have no effect on the bit-band bit. Writing `0x01` has the same effect as writing `0xFF`. Writing `0x00` has the same effect as writing `0x0E`.

Reading a word in the alias region returns either `0x01` or `0x00`. A value of `0x01` indicates that the targeted bit in the bit-band region is set. A value of `0x00` indicates that the targeted bit is clear. Bits [31:1] are zero.

3.7.3 Directly accessing a bit-band region

You can directly access the bit-band region with normal reads and writes to that region.

3.8 Processor core register summary

The processor has 32-registers that includes 13 general-purpose registers and several special-purpose registers.

The processor has the following 32-bit registers:
- 13 general-purpose registers, R0-R12.
- *Stack Pointer* (SP), R13 alias of banked registers, SP_process and SP_main.
- *Link Register* (LR), R14.
- *Program Counter* (PC), R15.
- Special-purpose *Program Status Registers*, (*x*PSR).

The following figure shows the processor register set.

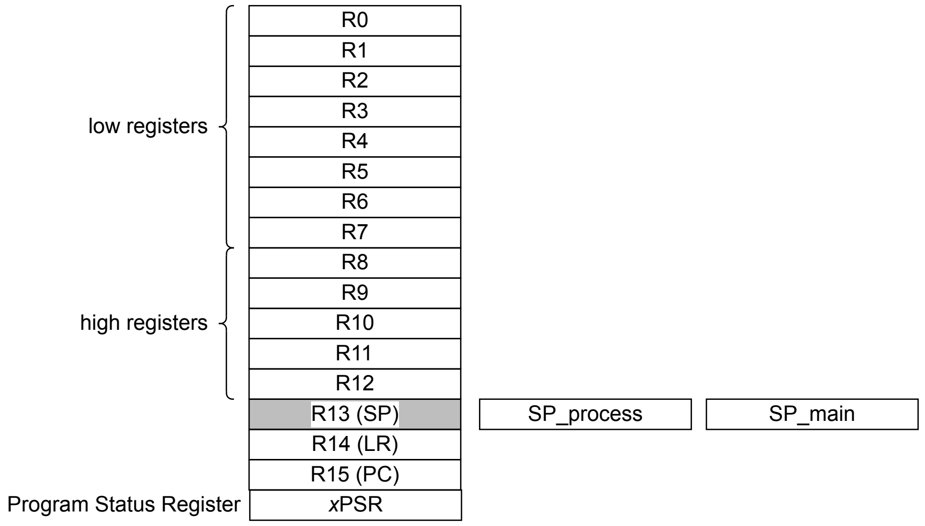


Figure 3-3 Processor register set

The general-purpose registers R0-R12 have no special architecturally-defined uses. Most instructions that can specify a general-purpose register can specify R0-R12.

Low registers
Registers R0-R7 are accessible by all instructions that specify a general-purpose register.

High registers
Registers R8-R12 are accessible by all 32-bit instructions that specify a general-purpose register.

Registers R8-R12 are not accessible by most 16-bit instructions.

Registers R13, R14, and R15 have the following special functions:

Stack pointer
Register R13 is used as the *Stack Pointer* (SP). Because the SP ignores writes to bits [1:0], it is autoaligned to a word, four-byte boundary.

Handler mode always uses SP_main, but you can configure Thread mode to use either SP_main or SP_process.

Link register

Register R14 is the subroutine *Link Register* (LR).

The LR receives the return address from PC when a *Branch and Link* (`BL`) or *Branch and Link with Exchange* (`BLX`) instruction is executed.

The LR is also used for exception return.

At all other times, you can treat R14 as a general-purpose register.

Program counter

Register R15 is the *Program Counter* (PC).

Bit [0] is always 0, so instructions are always aligned to word or halfword boundaries.

See the *ARM®v7-M Architecture Reference Manual* for more information.

3.9 Exceptions

Exceptions are handled and prioritized by the processor and the NVIC. In addition to architecturally defined behavior, the processor implements advanced exception and interrupt handling that reduces interrupt latency and includes implementation defined behavior.

This section contains the following subsections:

- *3.9.1 Exception handling and prioritization* on page 3-49.
- *3.9.2 Interrupt latency* on page 3-49.
- *3.9.3 Base register update in LDM and STM operations* on page 3-50.

3.9.1 Exception handling and prioritization

The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions.

When handling exceptions:

- All exceptions are handled in Handler mode.
- Processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the *Interrupt Service Routine* (ISR).
- The vector is fetched in parallel to the state saving, enabling efficient interrupt entry.

The processor supports tail-chaining that enables back-to-back interrupts without the overhead of state saving and restoration.

You configure the number of interrupts, and bits of interrupt priority, during implementation. Software can choose only to enable a subset of the configured number of interrupts, and can choose how many bits of the configured priorities to use.

Note

Vector table entries are compatible with interworking between ARM and Thumb instructions. This causes bit [0] of the vector value to load into the *Execution Program Status Register* (EPSR) T-bit on exception entry. All populated vectors in the vector table entries must have bit [0] set. Creating a table entry with bit [0] clear generates an INVSTATE fault on the first instruction of the handler corresponding to this vector.

3.9.2 Interrupt latency

The processor implements advanced exception and interrupt handling that reduces interrupt latency, and includes implementation defined behavior in addition to the architecturally defined behavior.

To reduce interrupt latency, the processor implements both interrupt late-arrival and interrupt tail-chaining mechanisms, as defined by the ARMv7-M architecture:

- There is a maximum of a twelve cycle latency from asserting the interrupt to execution of the first instruction of the ISR when the memory being accessed has no wait states being applied. When the FPU option is implemented and a floating point context is active and the lazy stacking is not enabled, this maximum latency is increased to twenty nine cycles. The first instructions to be executed are fetched in parallel to the stack push.

  See the *Cortex®-M4 Lazy Stacking and Context Switching Application Note 298* for more information about how to use lazy stacking.
- Returns from interrupts similarly take ten cycles where the instruction being returned to is fetched in parallel to the stack pop. If the floating point option is implemented and an active floating point context is included in the stack frame, the return from interrupt takes twenty-seven cycles.
- Tail chaining requires six cycles when using zero wait state memory. No stack pushes or pops are performed and only the instruction for the next ISR is fetched.

The processor exception model has the following implementation-defined behavior in addition to the architecturally defined behavior:

- Exceptions on stacking from HardFault to NMI lockup at NMI priority.
- Exceptions on unstacking from NMI to HardFault lockup at HardFault priority.

To minimize interrupt latency, the processor abandons any divide instruction to take any pending interrupt. On return from the interrupt handler, the processor restarts the divide instruction from the beginning. The processor implements the Interruptible-continuable Instruction field. Load multiple (`LDM`) operations and store multiple (`STM`) operations are interruptible. The EPSR holds the information required to continue the load or store multiple from the point where the interrupt occurred.

This means that software must not use load-multiple or store-multiple instructions to access a device or access a memory region that is read-sensitive or sensitive to repeated writes. The software must not use these instructions in any case where repeated reads or writes might cause inconsistent results or unwanted side-effects.

For more information, see the *ARM®v7-M Architecture Reference Manual*.

3.9.3 Base register update in LDM and STM operations

When the instruction specifies base register write-back, the base register changes to the updated address (an abort restores the original base value). When the base register is in the register list of an LDM, and is not the last register in the list, the base register changes to the loaded value.

An `LDM` or `STM` is restarted rather than continued if:
- The instruction faults.
- The instruction is inside an `IT`.

If an `LDM` has completed a base load, it is continued from before the base load.

Chapter 4
System Control

This chapter provides a summary of the system control registers whose implementation is specific to the Cortex-M4 processor.

Registers not described here are described in the *ARM®v7-M Architecture Reference Manual*.

It contains the following sections:

- *4.1 System control registers* on page 4-52.
- *4.2 Auxiliary Control Register, ACTLR* on page 4-54.
- *4.3 CPUID Base Register, CPUID* on page 4-55.
- *4.4 Auxiliary Fault Status Register, AFSR* on page 4-56.

4.1 System control registers

List of system control registers whose implementation is specific to the Cortex-M4 processor.

Table 4-1 System control registers

Address	Name	Type	Reset	Description
`0xE000E008`	ACTLR	RW	`0x00000000`	Auxiliary Control Register, ACTLR
`0xE000E010`	STCSR	RW	`0x00000000`	SysTick Control and Status Register
`0xE000E014`	STRVR	RW	Unknown	SysTick Reload Value Register
`0xE000E018`	STCVR	RW clear	Unknown	SysTick Current Value Register
`0xE000E01C`	STCR	RO	STCALIB	SysTick Calibration Value Register
`0xE000ED00`	CPUID	RO	`0x410FC241`	Refer to the CPUID Base Register, CPUID
`0xE000ED04`	ICSR	RW or RO	`0x00000000`	Interrupt Control and State Register
`0xE000ED08`	VTOR	RW	`0x00000000`	Vector Table Offset Register
`0xE000ED0C`	AIRCR	RW	`0x00000000`	Application Interrupt and Reset Control Register. Bits [10:8] are reset to zero. The ENDIANNESS bit, bit [15], can reset to either state, depending on the implementation.
`0xE000ED10`	SCR	RW	`0x00000000`	System Control Register
`0xE000ED14`	CCR	RW	`0x00000200`	Configuration and Control Register. The processor implements bit[9] of CCR, STKALIGN, as RW.
`0xE000ED18`	SHPR1	RW	`0x00000000`	System Handler Priority Register 1
`0xE000ED1C`	SHPR2	RW	`0x00000000`	System Handler Priority Register 2
`0xE000ED20`	SHPR3	RW	`0x00000000`	System Handler Priority Register 3
`0xE000ED24`	SHCSR	RW	`0x00000000`	System Handler Control and State Register
`0xE000ED28`	CFSR	RW	`0x00000000`	Configurable Fault Status Registers
`0xE000ED2C`	HFSR	RW	`0x00000000`	HardFault Status Register
`0xE000ED30`	DFSR	RW	`0x00000000`	Debug Fault Status Register
`0xE000ED34`	MMFAR	RW	Unknown	MemManage Fault Address Register. BFAR and MMFAR are the same physical register. Because of this, the BFARVALID and MMFARVALID bits are mutually exclusive.
`0xE000ED38`	BFAR	RW	Unknown	BusFault Address Register. ID_DFR0 reads as 0 if no debug support is implemented.
`0xE000ED3C`	AFSR	RW	`0x00000000`	See the Auxiliary Fault Status Register, AFSR
`0xE000ED40`	ID_PFR0	RO	`0x00000030`	Processor Feature Register 0
`0xE000ED44`	ID_PFR1	RO	`0x00000200`	Processor Feature Register 1
`0xE000ED48`	ID_DFR0	RO	`0x00100000`	Debug Features Register 0. ID_DFR0 reads as 0 if no debug support is implemented.
`0xE000ED4C`	ID_AFR0	RO	`0x00000000`	Auxiliary Features Register 0

Table 4-1 System control registers (continued)

Address	Name	Type	Reset	Description
`0xE000ED50`	ID_MMFR0	RO	`0x00100030`	Memory Model Feature Register 0
`0xE000ED54`	ID_ MMFR1	RO	`0x00000000`	Memory Model Feature Register 1
`0xE000ED58`	ID_MMFR2	RO	`0x01000000`	Memory Model Feature Register 2
`0xE000ED5C`	ID_MMFR3	RO	`0x00000000`	Memory Model Feature Register 3
`0xE000ED60`	ID_ISAR0	RO	`0x01141110`	Instruction Set Attributes Register 0
`0xE000ED64`	ID_ISAR1	RO	`0x02112000`	Instruction Set Attributes Register 1
`0xE000ED68`	ID_ISAR2	RO	`0x21232231`	Instruction Set Attributes Register 2
`0xE000ED6C`	ID_ISAR3	RO	`0x01111131`	Instruction Set Attributes Register 3
`0xE000ED70`	ID_ISAR4	RO	`0x01310132`	Instruction Set Attributes Register 4
`0xE000ED88`	CPACR	RW	-	Coprocessor Access Control Register
`0xE000EF00`	STIR	WO	`0x00000000`	Software Triggered Interrupt Register

Related references

4.4 Auxiliary Fault Status Register, AFSR on page 4-56.

4.2 Auxiliary Control Register, ACTLR

Characteristics and bit assignments of the ACTLR register.

Purpose
Disables certain aspects of functionality within the processor.

Usage Constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
See the System control registers table.

The following figure shows the ACTLR bit assignments.

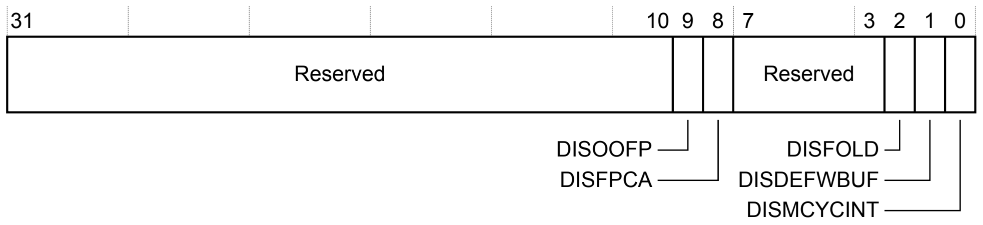


Figure 4-1 ACTLR bit assignments

The following table shows the ACTLR bit assignments.

Table 4-2 ACTLR bit assignments

Bits	Name	Function
[31:10]	-	Reserved.
[9]	DISOOFP	Disables floating point instructions completing out of order with respect to integer instructions.
[8]	DISFPCA	SBZP.
[7:3]	-	Reserved
[2]	DISFOLD	Disables folding of `IT` instructions.
[1]	DISDEFWBUF	Disables write buffer use during default memory map accesses. This causes all bus faults to be precise, but decreases the performance of the processor because stores to memory must complete before the next instruction can be executed.
[0]	DISMCYCINT	Disables interruption of multi-cycle instructions. This increases the interrupt latency of the processor because load/store and multiply/divide operations complete before interrupt stacking occurs.

Related references

4.1 System control registers on page 4-52.

4.3 CPUID Base Register, CPUID

Characteristics and bit assignments of the CPUID register.

Purpose

Specifies:

- The ID number of the processor core.
- The version number of the processor core.
- The implementation details of the processor core.

Usage Constraints

There are no usage constraints.

Configurations

This register is available in all processor configurations.

Attributes

Described in the System control registers table.

The following figure shows the CPUID bit assignments.

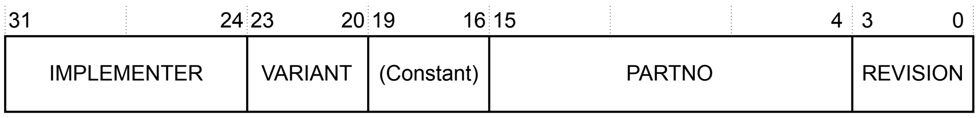


Figure 4-2 CPUID bit assignments

The following table shows the CPUID bit assignments.

Table 4-3 CPUID bit assignments

Bits	NAME	Function
[31:24]	IMPLEMENTER	Indicates implementer: `0x41` = ARM
[23:20]	VARIANT	Indicates processor revision: `0x0` = Revision 0
[19:16]	(Constant)	Reads as `0xF`
[15:4]	PARTNO	Indicates part number: `0xC24` = Cortex-M4
[3:0]	REVISION	Indicates patch release: `0x1`= Patch 1.

4.4 Auxiliary Fault Status Register, AFSR

Characteristics and bit assignments of the AFSR register.

Purpose

Specifies additional system fault information to software.

Usage Constraints

The AFSR flags map directly onto the AUXFAULT inputs of the processor, and a single-cycle high level on an external pin causes the corresponding AFSR bit to become latched as one. The bit can only be cleared by writing a one to the corresponding AFSR bit.

When an AFSR bit is written or latched as one, an exception does not occur. To make use of AUXFAULT input signals, software must poll the AFSR.

Configurations

This register is available in all processor configurations.

Attributes

See the System control registers table.

The following figure shows the AFSR bit assignments.

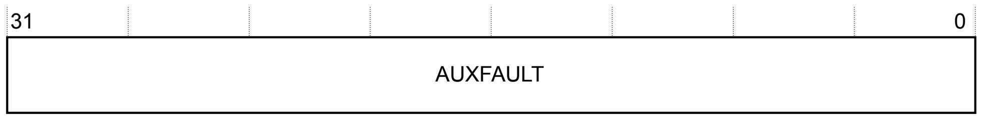


Figure 4-3 AFSR bit assignments

The following table shows the AFSR bit assignments.

Table 4-4 AFSR bit assignments

Bits	Name	Function
[31:0]	AUXFAULT	Latched version of the AUXFAULT inputs.

Related references

4.1 System control registers on page 4-52.

4.3 CPUID Base Register, CPUID on page 4-55.

Chapter 5
Memory Protection Unit

This chapter describes the processor *Memory Protection Unit* (MPU).

It contains the following sections:

- *5.1 About the MPU* on page 5-58.
- *5.2 MPU functional description* on page 5-59.
- *5.3 MPU programmers model table* on page 5-60.

5.1 About the MPU

The MPU enforces privilege rules, separates processes, and enforces access rules to memory. The MPU is an optional component and supports the standard ARMv7 Protected Memory System Architecture model.

The MPU provides full support for:

- Protection regions.
- Overlapping protection regions, with ascending region priority:
  - 7 = highest priority.
  - 0 = lowest priority.
- Access permissions.
- Exporting memory attributes to the system.

MPU mismatches and permission violations invoke the programmable-priority MemManage fault handler. See the *ARM®v7-M Architecture Reference Manual* for more information.

You can use the MPU to:

- Enforce privilege rules.
- Separate processes.
- Enforce access rules.

5.2 MPU functional description

The access permission bits, TEX, C, B, AP, and XN, of the Region Access Control Register control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, a permission fault is raised.

For more information, see the *ARM®v7-M Architecture Reference Manual*.

5.3 MPU programmers model table

Table of MPU registers, with address, name, type, reset, and description information.

These registers are described in the *ARMv7-M Architecture Reference Manual*.

Table 5-1 MPU registers

Address	Name	Type	Reset	Description
`0xE000ED90`	MPU_TYPE	RO	`0x00000800`	MPU Type Register If the MPU is not present in the implementation this register reads as zero.
`0xE000ED94`	MPU_CTRL	RW	`0x00000000`	MPU Control Register
`0xE000ED98`	MPU_RNR	RW	`0x00000000`	MPU Region Number Register
`0xE000ED9C`	MPU_RBAR	RW	`0x00000000`	MPU Region Base Address Register
`0xE000EDA0`	MPU_RASR	RW	`0x00000000`	MPU Region Attribute and Size Register
`0xE000EDA4`	MPU_RBAR_A1		`0x00000000`	MPU alias registers
`0xE000EDA8`	MPU_RASR_A1		`0x00000000`	
`0xE000EDAC`	MPU_RBAR_A2		`0x00000000`	
`0xE000EDB0`	MPU_RASR_A2		`0x00000000`	
`0xE000EDB4`	MPU_RBAR_A3		`0x00000000`	
`0xE000EDB8`	MPU_RASR_A3		`0x00000000`	

Chapter 6
Nested Vectored Interrupt Controller

This chapter describes the *Nested Vectored Interrupt Controller* (NVIC). The NVIC provides configurable interrupt handling abilities to the processor, facilitates low- latency exception and interrupt handling, and controls power management.

It contains the following sections:

- *6.1 NVIC functional description* on page 6-62.
- *6.2 NVIC programmers' model* on page 6-63.

6.1 NVIC functional description

The NVIC supports up to 240 interrupts, each with up to 256 levels of priority that can be changed dynamically.The processor and NVIC can be put into a very low-power sleep mode, leaving the Wake Up Controller (WIC) to identify and prioritize interrupts. Also, the processor supports both level and pulse interrupts.

This section contains the following subsections:

- *6.1.1 NVIC interrupts* on page 6-62.
- *6.1.2 Low power modes* on page 6-62.
- *6.1.3 Level versus pulse interrupts* on page 6-62.

6.1.1 NVIC interrupts

The NVIC supports up to 240 interrupts, each with up to 256 levels of priority. You can change the priority of an interrupt dynamically.

The NVIC and the processor core interface are closely coupled, to enable low latency interrupt processing and efficient processing of late arriving interrupts. The NVIC maintains knowledge of the stacked, or nested, interrupts to enable tail-chaining of interrupts. You can only fully access the NVIC from privileged mode, but you can cause interrupts to enter a pending state in user mode if you enable the Configuration and Control Register. Any other user mode access causes a bus fault.

You can access all NVIC registers using byte, halfword, and word accesses unless otherwise stated. NVIC registers are located within the SCS.

All NVIC registers and system debug registers are little-endian regardless of the endianness state of the processor.

Related concepts

3.9 Exceptions on page 3-49.

6.1.2 Low power modes

Your processor implementation can include a Wake-up Interrupt Controller (WIC). This enables the processor and NVIC to be put into a very low-power sleep mode leaving the WIC to identify and prioritize interrupts.

The processor fully implements the *Wait For Interrupt* (`WFI`), *Wait For Event* (`WFE`) and the *Send Event* (`SEV`) instructions. In addition, the processor also supports the use of SLEEPONEXIT, that causes the processor core to enter sleep mode when it returns from an exception handler to Thread mode. See the *ARM®v7-M Architecture Reference Manual* for more information.

6.1.3 Level versus pulse interrupts

The processor supports both level and pulse interrupts. A level interrupt is held asserted until it is cleared by the ISR accessing the device. A pulse interrupt is a variant of an edge model.

You must ensure that the pulse is sampled on the rising edge of the Cortex-M4 clock, **FCLK**, instead of being asynchronous.

For level interrupts, if the signal is not deasserted before the return from the interrupt routine, the interrupt again enters the pending state and re-activates. This is particularly useful for FIFO and buffer-based devices because it ensures that they drain either by a single ISR or by repeated invocations, with no extra work. This means that the device holds the signal in assert until the device is empty.

A pulse interrupt can be reasserted during the ISR so that the interrupt can be in the pending state and active at the same time. If another pulse arrives while the interrupt is still pending, the interrupt remains pending and the ISR runs only once.

Pulse interrupts are mostly used for external signals and for rate or repeat signals.

6.2 NVIC programmers' model

Summary of the NVIC registers whose implementation is specific to the Cortex-M4 processor.

Registers not described here are described in the *ARM®v7M Architecture Reference Manual*.

This section contains the following subsections:

- *6.2.1 Table of NVIC registers* on page 6-63.
- *6.2.2 Interrupt Controller Type Register, ICTR* on page 6-63.

6.2.1 Table of NVIC registers

Table showing the NVIC registers, with address, name, type, reset and description information for each register.

Table 6-1 NVIC registers

Address	Name	Type	Reset	Description
`0xE000E004`	ICTR	RO	-	Interrupt Controller Type Register, ICTR
`0xE000E100 - 0xE000E11C`	NVIC_ISER0 - NVIC_ISER7	RW	`0x00000000`	Interrupt Set-Enable Registers
`0xE000E180 - 0xE000E19C`	NVIC_ICER0 - NVIC_ICER7	RW	`0x00000000`	Interrupt Clear-Enable Registers
`0xE000E200 - 0xE000E21C`	NVIC_ISPR0 - NVIC_ISPR7	RW	`0x00000000`	Interrupt Set-Pending Registers
`0xE000E280- 0xE000E29C`	NVIC_ICPR0 - NVIC_ICPR7	RW	`0x00000000`	Interrupt Clear-Pending Registers
`0xE000E300 - 0xE000E31C`	NVIC_IABR0 - NVIC_IABR7	RO	`0x00000000`	Interrupt Active Bit Register
`0xE000E400- 0xE000E4EC`	NVIC_IPR0 - NVIC_IPR59	RW	`0x00000000`	Interrupt Priority Register

Related references

6.2.2 Interrupt Controller Type Register, ICTR on page 6-63.

6.2.2 Interrupt Controller Type Register, ICTR

Characteristics and bit assignments of the ICTR register.

Purpose
Shows the number of interrupt lines that the NVIC supports.

Usage Constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
See the register summary information.

The following figure shows the ICTR bit assignments.

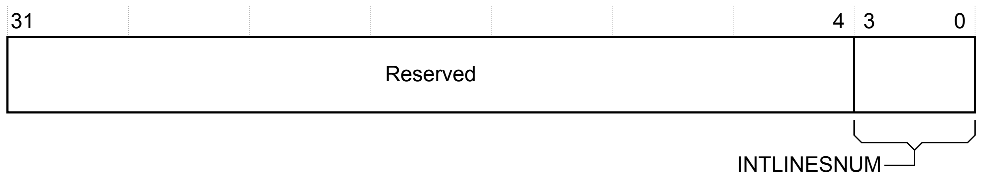


Figure 6-1 ICTR bit assignments

The following table shows the ICTR bit assignments.

Table 6-2 ICTR bit assignments

Bits	Name	Function	Notes
[31:4]	-	Reserved.	
[3:0]	INTLINESNUM	Total number of interrupt lines in groups of 32: `0b0000` = 0...32 `0b0001` = 33...64 `0b0010` = 65...96 `0b0011` = 97...128 `0b0100` = 129...160 `0b0101` = 161...192 `0b0110` = 193...224 `0b0111` = 225...256	The processor supports a maximum of 240 external interrupts.

Chapter 7
Floating-Point Unit

This chapter describes the programmers' model of the *Floating-Point Unit* (FPU).

It contains the following sections:

- *7.1 About the FPU* on page 7-66.
- *7.2 FPU functional description* on page 7-67.
- *7.3 FPU programmers' model* on page 7-72.

7.1 About the FPU

The Cortex-M4 FPU is an implementation of the single precision variant of the ARMv7-M Floating-Point Extension (FPv4-SP). It provides floating-point computation functionality that is compliant with the ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic, referred to as the IEEE 754 standard.

The FPU supports all single-precision data-processing instructions and data types described in the *ARM®v7-M Architecture Reference Manual*.

7.2 FPU functional description

The FPU fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

This section contains the following subsections:

- *7.2.1 FPU views of the register bank* on page 7-67.
- *7.2.2 Modes of operation* on page 7-68.
- *7.2.3 FPU instruction set table* on page 7-68.
- *7.2.4 Compliance with the IEEE 754 standard* on page 7-70.
- *7.2.5 Complete implementation of the IEEE 754 standard* on page 7-70.
- *7.2.6 IEEE 754 standard implementation choices* on page 7-70.
- *7.2.7 Exceptions* on page 7-71.

7.2.1 FPU views of the register bank

The FPU provides an extension register file containing 32 single-precision registers.

These can be viewed as:

- Sixteen 64-bit doubleword registers, D0-D15.
- Thirty-two 32-bit single-word registers, S0-S31.
- A combination of registers from these views:

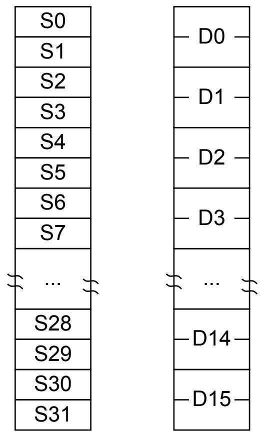


Figure 7-1 FPU register bank

The mapping between the registers is as follows:

- `S<2n>` maps to the least significant half of `D<n>`.
- `S<2n+1>` maps to the most significant half of `D<n>`.

For example, you can access the least significant half of the value in `D6` by accessing `S12`, and the most significant half of the elements by accessing `S13`.

7.2.2 Modes of operation

The FPU provided full-compliance, flush-to-zero, and default NaN modes of operation.

Full-compliance mode

In full-compliance mode, the FPU processes all operations according to the IEEE 754 standard in hardware.

Flush-to-zero mode

Setting the FZ bit of the Floating-point Status and Control Register FPSCR[24], enables flush-to-zero mode. In this mode, the FPU treats all subnormal input operands of arithmetic CDP operations as zeros in the operation. Exceptions that result from a zero operand are signaled appropriately.

`VABS`, `VNEG`, and `VMOV` are not considered arithmetic `CDP` operations and are not affected by flush-to-zero mode. A result that is *tiny*, as described in the IEEE 754 standard, where the destination precision is smaller in magnitude than the minimum normal value *before rounding*, is replaced with a zero. The IDC flag, FPSCR[7], indicates when an input flush occurs. The UFC flag, FPSCR[3], indicates when a result flush occurs.

Default NaN mode

Setting the DN bit, FPSCR[25], enables default NaN mode. In this mode, the result of any arithmetic data processing operation that involves an input NaN, or that generates a NaN result, returns the default NaN.

Propagation of the fraction bits is maintained only by `VABS`, `VNEG`, and `VMOV` operations. All other `CDP` operations ignore any information in the fraction bits of an input NaN.

Related concepts

Full-compliance mode on page 7-68.
Flush-to-zero mode on page 7-68.
Default NaN mode on page 7-68.

7.2.3 FPU instruction set table

Table showing the FPU instruction set, with description, assembler, and cycles information for each operation.

Table 7-1 FPU instruction set

Operation	Description	Assembler	Cycles
Absolute value	Of float	`VABS.F32`	1
Addition	Floating-point	`VADD.F32`	1
Compare	Float with register or zero	`VCMP.F32`	1
	Float with register or zero	`VCMPE.F32`	1
Convert	Between integer, fixed-point, half-precision and float	`VCVT.F32`	1
Divide	Floating-point	`VDIV.F32`	14
Load	Multiple doubles	`VLDM.64`	1+2*N, where N is the number of doubles.
	Multiple floats	`VLDM.32`	1+N, where N is the number of floats.
	Single double	`VLDR.64`	3
	Single float	`VLDR.32`	2

Table 7-1 FPU instruction set (continued)

Operation	Description	Assembler	Cycles
Move	top/bottom half of double to/from core register	`VMOV`	1
	immediate/float to float-register	`VMOV`	1
	Two floats/one double to/from two core registers or one float to/from one core register	`VMOV`	2
	floating-point control/status to core register	`VMRS`	1
	Core register to floating-point control/status	`VMSR`	1
Multiply	Float	`VMUL.F32`	1
	Then accumulate float	`VMLA.F32`	3
	Then subtract float	`VMLS.F32`	3
	Then accumulate then negate float	`VNMLA.F32`	3
	Then subtract then negate float	`VNMLS.F32`	3
Multiply (fused)	Then accumulate float	`VFMA.F32`	3
	Then subtract float	`VFMS.F32`	3
	Then accumulate then negate float	`VFNMA.F32`	3
	Then subtract then negate float	`VFNMS.F32`	3
Negate	Float	`VNEG.F32`	1
	And multiply float	`VNMUL.F32`	1
Pop	Double registers from stack	`VPOP.64`	1+2*N, where N is the number of double registers.
	Float registers from stack	`VPOP.32`	1+N where N is the number of registers.
Push	Double registers to stack	`VPUSH.64`	1+2*N, where N is the number of double registers.
	Float registers to stack	`VPUSH.32`	1+N, where N is the number of registers.
Square-root	Of float	`VSQRT.F32`	14
Store	Multiple double registers	`VSTM.64`	1+2*N, where N is the number of doubles.
	Multiple float registers	`VSTM.32`	1+N, where N is the number of floats.
	Single double register	`VSTR.64`	3
	Single float registers	`VSTR.32`	2
Subtract	Float	`VSUB.F32`	1

Note

- Integer-only instructions following `VDIVR` or `VSQRT` instructions complete out-of-order. `VDIV` and `VSQRT` instructions take one cycle if no more floating-point instructions are executed.
- Floating-point arithmetic data processing instructions, such as add, subtract, multiply, divide, square-root, all forms of multiply with accumulate, in addition to conversions of all types take one cycle longer if their result is consumed by the following instruction.

- Both fused and chained multiply with accumulate instructions consume their addend one cycle later, so the result of an arithmetic instruction that is followed by a multiply with accumulate instruction is consumed as the addend of the MAC instruction.

7.2.4 Compliance with the IEEE 754 standard

When Default NaN (DN) and Flush-to-Zero (FZ) modes are disabled, FPv4 functionality is compliant with the IEEE 754 standard in hardware. No support code is required to achieve this compliance.

See the *ARM®v7-M Architecture Reference Manual* for information about FP architecture compliance with the IEEE 754 standard.

7.2.5 Complete implementation of the IEEE 754 standard

The Cortex-M4 FPU supports fused MAC operations as described in the IEEE standard. For complete implementation of the IEEE 754-2008 standard, floating-point functionality must be augmented with library functions.

The Cortex-M4 floating point instruction set does not support all operations defined in the IEEE 754-2008 standard. Unsupported operations include, but are not limited to the following:
- Remainder.
- Round floating-point number to integer-valued floating-point number.
- Binary-to-decimal conversions.
- Decimal-to-binary conversions.
- Direct comparison of single-precision and double-precision values.

7.2.6 IEEE 754 standard implementation choices

The Cortex-M4 processor implements IEEE 754-2008 standard implementation choices for NaN handling, comparisons, underflow, and exceptions.

Some of the implementation choices permitted by the IEEE 754-2008 standard and used in the FPv4 architecture are described in the ARMv7-M Architecture Reference Manual.

NaN handling

All single-precision values with the maximum exponent field value and a nonzero fraction field are valid NaNs. A most significant fraction bit of zero indicates a *Signaling NaN* (SNaN). A one indicates a *Quiet NaN* (QNaN). Two NaN values are treated as different NaNs if they differ in any bit.

The following table shows the default NaN values.

Table 7-2 Default NaN values

Sign	Fraction	Fraction
0	`0xFF`	bit [22] = 1, bits [21:0] are all zeros

Processing of input NaNs for ARM floating-point functionality and libraries is defined as follows:

- In full-compliance mode, NaNs are handled as described in the *ARM®v7-M Architecture Reference Manual*. The hardware processes the NaNs directly for arithmetic `CDP` instructions. For data transfer operations, NaNs are transferred without raising the Invalid Operation exception. For the non-arithmetic `CDP` instructions, `VABS`, `VNEG`, and `VMOV`, NaNs are copied, with a change of sign if specified in the instructions, without causing the Invalid Operation exception.
- In default NaN mode, arithmetic `CDP` instructions involving NaN operands return the default NaN regardless of the fractions of any NaN operands. SNaNs in an arithmetic `CDP` operation set the IOC flag (IOC is the Invalid Operation exception flag, FPSCR[0]), FPSCR[0]. NaN handling by data transfer and non-arithmetic `CDP` instructions is the same as in full-compliance mode.

The following table summarizes the effects of NaN operands on instruction execution.

Table 7-3 QNaN and SNaN handling

Instruction type	Default NaN mode	With QNaN operand	With SNaN operand
Arithmetic `CDP`	Off	The QNaN or one of the QNaN operands, if there is more than one, is returned according to the rules given in the *ARM®v7-M Architecture Reference Manual*.	IOC set. The SNaN is quieted and the result NaN is determined by the rules given in the *ARM®v7-M Architecture Reference Manual*.
	On	Default NaN returns.	IOC set. Default NaN returns.
Non-arithmetic `CDP`	Off	NaN passes to destination with sign changed as appropriate.	
	On		
`FCMP(Z)`	-	Unordered compare.	IOC set. Unordered compare.
`FCMPE(Z)`	-	IOC set. Unordered compare.	IOC set. Unordered compare.
Load/store	Off	All NaNs transferred.	
	On		

Comparisons

You can use the `MVRS APSR_nzcv` instruction (formerly `FMSTAT`) to transfer the current flags from the FPSCR to the APSR. See the *ARM®v7-M Architecture Reference Manual* for mapping of IEEE 754-2008 standard predicates to ARM conditions. The flags used are chosen so that subsequent conditional execution of ARM instructions can test the predicates defined in the IEEE standard.

Underflow

The FPU uses the *before rounding* form of *tininess* and the *inexact result* form of *loss of accuracy* as described in the IEEE 754-2008 standard to generate Underflow exceptions.

In flush-to-zero mode, results that are tiny before rounding, as described in the IEEE standard, are flushed to a zero, and the UFC flag, FPSCR[3], is set. See the *ARM®v7-M Architecture Reference Manual* for information on flush-to-zero mode.

When the FPU is not in flush-to-zero mode, operations are performed on subnormal operands. If the operation does not produce a tiny result, it returns the computed result, and the UFC flag, FPSCR[3], is not set. The IXC flag, FPSCR[4], is set if the operation is inexact. If the operation produces a tiny result, the result is a subnormal or zero value, and the UFC flag, FPSCR[3], is set if the result was also inexact.

7.2.7 Exceptions

The FPU sets the cumulative exception status flag in the FPSCR register as required for each instruction, in accordance with the FPv4 architecture. The FPU does not support exception traps.

The processor also has six output pins, **FPIXC**, **FPUFC**, **FPOFC**, **FPDZC**, **FPIDC**, and **FPIOC**, that each reflect the status of one of the cumulative exception flags. See the *Cortex®-M4 Integration and Implementation Manual* for a description of these outputs.

The processor can reduce the exception latency by using lazy stacking. This means that the processor reserves space on the stack for the FP state, but does not save that state information to the stack unless the processor executes an FPU instruction in the current exception handler.

The lazy save of the FP state is interruptible by a higher priority exception. The FP state saving operation starts over after that exception returns.

See the *ARM®v7-M Architecture Reference Manual* for more information.

7.3 FPU programmers' model

Summary of the FPU registers, and example code sequence for enabling the FPU in both privileged and user modes.

This section contains the following subsections:
- *7.3.1 Floating Point system registers* on page 7-72.
- *7.3.2 Enabling the FPU* on page 7-72.

7.3.1 Floating Point system registers

Summary of the FP system registers in the Cortex-M4 processor, if your implementation includes the FPU.

All Cortex-M4 FPU registers are described in the *ARM®v7-M Architecture Reference Manual*.

Table 7-4 Cortex-M4 Floating Point system registers

Address	Name	Type	Reset	Description
`0xE000EF34`	FPCCR	RW	`0xC0000000`	FP Context Control Register
`0xE000EF38`	FPCAR	RW	-	FP Context Address Register
`0xE000EF3C`	FPDSCR	RW	`0x00000000`	FP Default Status Control Register
`0xE000EF40`	MVFR0	RO	`0x10110021`	Media and VFP Feature Register 0, MVFR0
`0xE000EF44`	MVFR1	RO	`0x11000011`	Media and VFP Feature Register 1, MVFR1

7.3.2 Enabling the FPU

Example code sequence for enabling the FPU in both privileged and user modes. The processor must be in privileged mode to read from and write to the CPACR.

Enabling the FPU

```
; CPACR is located at address 0xE000ED88

LDR.W   R0, 0xE000ED88

; Read CPACR

LDR     R1, [R0]

Set bits 20-23 to enable CP10 and CP11 coprocessors

ORR     R1, R1, #0xF << 20)

; Write back the modified value to the CPACR

STR     R1, [R0]
```

Chapter 8
Debug

This chapter describes how to debug and test software running on the processor.

It contains the following sections:

- *8.1 Debug configuration* on page 8-74.
- *8.2 AHB-AP debug access port* on page 8-79.
- *8.3 Flash Patch and Breakpoint Unit (FPB)* on page 8-82.

8.1 Debug configuration

The processor implementation determines the debug configuration, including whether debug is implemented. Basic debug functionality includes processor halt, single-step, processor core register access, Vector Catch, unlimited software breakpoints, and full system memory access.

If the processor does not implement debug, no ROM table is present and the halt, breakpoint, and watchpoint functionality is not present.

The debug option might include:

- A breakpoint unit supporting two literal comparators and six instruction comparators, or only two instruction comparators.
- A watchpoint unit supporting one or four watchpoints.

See the *ARM®v7-M Architectural Reference Manual* for more information.

This section contains the following subsections:

- *8.1.1 CoreSight™ discovery* on page 8-74.
- *8.1.2 Debugger actions for identifying the processor* on page 8-75.
- *8.1.3 ROM table identification and entries* on page 8-75.
- *8.1.4 ROM table components* on page 8-76.
- *8.1.5 System Control Space registers* on page 8-77.
- *8.1.6 Debug register summary* on page 8-78.

8.1.1 CoreSight™ discovery

For processors that implement debug, ARM recommends that a debugger identify and connect to the debug components using the CoreSight debug infrastructure.

The following figure shows the recommended flow that a debugger can follow to discover the components in the CoreSight debug infrastructure. In this case a debugger reads the peripheral and component ID registers for each CoreSight component in the CoreSight system.

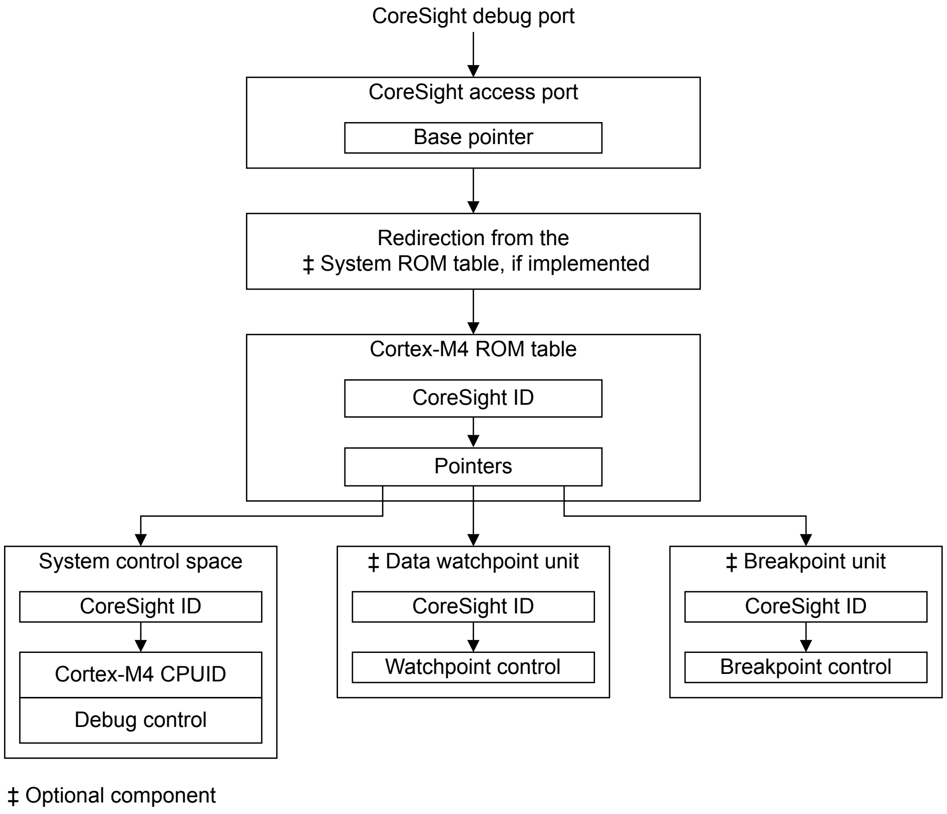


Figure 8-1 CoreSight discovery

8.1.2 Debugger actions for identifying the processor

When a debugger identifies the SCS from its CoreSight identification, it can identify the processor and its revision number from the CPUID register in the SCS at address `0xE000ED00`.

To identify the Cortex-M4 processor within the CoreSight system, ARM recommends that a debugger perform the following actions:

1. Locate and identify the Cortex-M4 ROM table using its CoreSight identification. See the Cortex-M4 ROM table identification values table.
2. Follow the pointers in that Cortex-M4 ROM table:
   a. *System Control Space* (SCS).
   b. *Breakpoint unit* (BPU).
   c. *Data Watchpoint and Trace unit* (DWT).

   See the Cortex-M4 ROM table components table.

A debugger cannot rely on the Cortex-M4 ROM table being the first ROM table encountered. One or more system ROM tables are required between the access port and the Cortex-M4 ROM table if other CoreSight components are in the system. If a system ROM table is present, this can include a unique identifier for the implementation.

8.1.3 ROM table identification and entries

The table shows the ROM table identification registers and values for debugger detection. The values allow debuggers to identify the processor and its debug capabilities. The values for the Peripheral ID registers identify this as a generic ROM table for the Cortex-M4 processor. Your implementation might use these registers to identify the manufacturer and part number for the device.

The Component ID registers identify this as a CoreSight ROM table.

Note

The Cortex-M4 ROM table only supports word size transactions.

See the *ARM®v7-M Architectural Reference Manual* and the *ARM® CoreSight™ Components Technical Reference Manual* for more information about the ROM table ID and component registers, and their addresses and access types.

Table 8-1 Cortex-M4 ROM table identification values

Address	Register	Value	Description
`0xE00FFFD0`	Peripheral ID4	`0x00000004`	*Component and Peripheral ID register formats* in the *ARM®v7-M Architectural Reference Manual*
`0xE00FFFD4`	Peripheral ID5	`0x00000000`	
`0xE00FFFD8`	Peripheral ID6	`0x00000000`	
`0xE00FFFDC`	Peripheral ID7	`0x00000000`	
`0xE00FFFE0`	Peripheral ID0	`0x000000C4`	
`0xE00FFFE4`	Peripheral ID1	`0x000000B4`	
`0xE00FFFE8`	Peripheral ID2	`0x0000000B`	
`0xE00FFFEC`	Peripheral ID3	`0x00000000`	
`0xE00FFFF0`	Component ID0	`0x0000000D`	
`0xE00FFFF4`	Component ID1	`0x00000010`	
`0xE00FFFF8`	Component ID2	`0x00000005`	
`0xE00FFFFC`	Component ID3	`0x000000B1`	

Related concepts

8.3 Flash Patch and Breakpoint Unit (FPB) on page 8-82.

Related references

8.1.5 System Control Space registers on page 8-77.
9.2 DWT Programmers' model on page 9-86.
10.2 ITM programmers' model on page 10-90.
11.3.1 TPIU registers on page 11-96.

8.1.4 ROM table components

The table shows the CoreSight components that the Cortex-M4 ROM table points to. The values depend on the implemented debug configuration. The ROM table entries point to the debug components of the processor. The offset for each entry is the offset of that component from the ROM table base address, E00FF000.

Table 8-2 Cortex-M4 ROM table components

Address	Component	Value	Description
`0xE00FF000`	SCS	`0xFFF0F003`	Refer to information for the System Control Space.
`0xE00FF004`	DWT	`0xFFF02003`	Refer to information for the Data Watchpoint programmer's model. Value reads as `0xFFF02002` if no watchpoints are implemented.

Table 8-2 Cortex-M4 ROM table components (continued)

Address	Component	Value	Description
`0xE00FF008`	FPB	`0xFFF03003`	Refer to information for the Flash Patch and Breakpoint Unit (FPB). Value reads as `0xFFF03002` if no breakpoints are implemented.
`0xE00FF00C`	ITM	`0xFFF01003`	Refer to information for the Instrumentation Trace Macrocell Unit. Value reads as `0xFFF01002` if no ITM is implemented.
`0xE00FF010`	TPIU	`0xFFF41003`	Refer to information for the Trace Port Interface Unit. Value reads as `0xFFF41002` if no TPIU is implemented.
`0xE00FF014`	ETM	`0xFFF42003`	See the *ETM-M4 Technical Reference Manual*. Value reads as `0xFFF42002` if no ETM is implemented.
`0xE00FF018`	End marker	`0x00000000`	See *DAP accessible ROM table* in the *ARM®v7-M Architectural Reference Manual*.
`0xE00FFFCC`	SYSTEM ACCESS	`0x00000001`	

8.1.5 System Control Space registers

If debug is implemented, the processor provides debug through registers in the System Control Space (SCS).

The following table shows the SCS CoreSight identification registers and values for debugger detection. Final debugger identification of the Cortex-M4 processor is through the CPUID register in the SCS.

Table 8-3 SCS identification values

Address	Register	Value	Description
`0xE000EFD0`	Peripheral ID4	`0x00000004`	*Component and Peripheral ID register formats* in the *ARM®v7-M Architectural Reference Manual*.
`0xE000EFE0`	Peripheral ID0	`0x00000000`	SCS identification value for implementations without FPU.
		`0x0000000C`	SCS identification value for implementations with FPU.
`0xE000EFE4`	Peripheral ID1	`0x000000B0`	
`0xE000EFE8`	Peripheral ID2	`0x0000000B`	
`0xE000EFEC`	Peripheral ID3	`0x00000000`	
`0xE000EFF0`	Component ID0	`0x0000000D`	
`0xE000EFF4`	Component ID1	`0x000000E0`	
`0xE000EFF8`	Component ID2	`0x00000005`	
`0xE000EFFC`	Component ID3	`0x000000B1`	

See the *ARM®v7-M Architectural Reference Manual* and the *ARM® CoreSight™ Components Technical Reference Manual* for more information about the SCS CoreSight identification registers, and their addresses and access types.

Related references

4.3 CPUID Base Register, CPUID on page 4-55.

8.1.6 Debug register summary

Summary of the debug registers. Each register is 32 bits wide. Core debug is an optional component. If core debug is removed then halt mode debugging is not supported, and there is no halt, stepping, or register transfer functionality. Debug monitor mode is still supported.

Debug registers are described in the *ARM®v7-M Architectural Reference Manual*.

Table 8-4 Debug registers

Address	Name	Type	Reset	Description
`0xE000ED30`	DFSR	RW	`0x00000000`	Debug Fault Status Register Power-on reset only.
`0xE000EDF0`	DHCSR	RW	`0x00000000`	Debug Halting Control and Status Register
`0xE000EDF4`	DCRSR	WO	-	Debug Core Register Selector Register
`0xE000EDF8`	DCRDR	RW	-	Debug Core Register Data Register
`0xE000EDFC`	DEMCR	RW	`0x00000000`	Debug Exception and Monitor Control Register

8.2 AHB-AP debug access port

The AHB-AP is an optional debug access port into the processor system that provides access to all memory and registers in the system, including processor registers through the SCS. System access is independent of the processor status. Either SW-DP or SWJ-DP is used to access the AHB-AP.

The AHB-AP is a *Memory Access Port* (MEM-AP) as defined in the *ARM® Debug Interface v5 Architecture Specification*.

The AHB-AP is a master into the Bus Matrix. Transactions are made using the AHB-AP programmers' model, which generates AHB-Lite transactions into the Bus Matrix.

This section contains the following subsections:
- *8.2.1 AHB-AP transaction types* on page 8-79.
- *8.2.2 AHB-AP programmers model* on page 8-79.

8.2.1 AHB-AP transaction types

The AHB-AP can perform unaligned and bit-band transactions.

The Bus Matrix handles AHB-AP transactions. The AHB-AP does not perform back-to-back transactions on the bus, and so all transactions are non-sequential. The AHB-AP transactions are not subject to MPU lookups. AHB-AP transactions bypass the FPB, and so the FPB cannot remap AHB-AP transactions.

AHB-AP transactions are little-endian.

8.2.2 AHB-AP programmers model

The programmers model lists all AHB-AP registers and describes those registers whose implementation is specific to the processor.

Other registers are described in the *CoreSight™ Components Technical Reference Manual*.

AHB-AP registers

Table showing the AHB-AP registers. Any register not specified in this table reads as zero. The offset given in this table is relative to the location of the AHB-AP in the DAP memory space. This space is only visible from the access port. It is not part of the processor memory map.

Table 8-5 AHB-AP register summary

Offset	Name	Type	Reset	Description
`0x00`	CSW	RW	See register	AHB-AP Control and Status Word Register, CSW
`0x04`	TAR	RW	-	AHB-AP Transfer Address Register
`0x0C`	DRW	RW	-	AHB-AP Data Read/Write Register
`0x10`	BD0	RW	-	AHB-AP Banked Data Register0
`0x14`	BD1	RW	-	AHB-AP Banked Data Register1
`0x18`	BD2	RW	-	AHB-AP Banked Data Register2
`0x1C`	BD3	RW	-	AHB-AP Banked Data Register3
`0xF8`	DBGDRAR	RO	`0xE00FF003`	AHB-AP ROM Address Register
`0xFC`	IDR	RO	`0x24770011`	AHB-AP Identification Register

AHB-AP Control and Status Word Register, CSW

Characteristics and bit assignments of the CSW register.

Purpose
Configures and controls transfers through the AHB interface.

Usage constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the AHB-AP register summary table.

The following figure shows the CSW bit assignments.

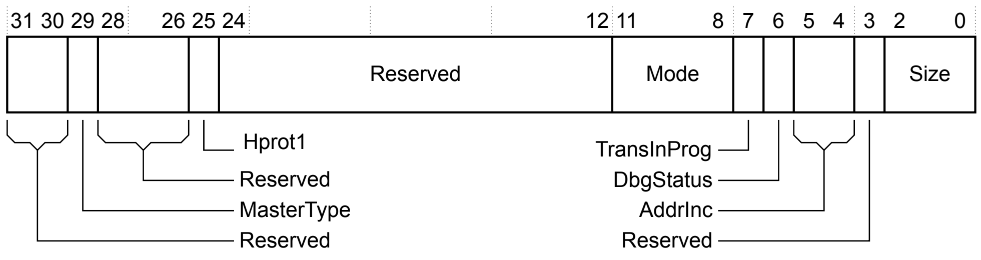


Figure 8-2 CSW bit assignments

The following table shows the CSW bit assignments.

Table 8-6 CSW bit assignments

Bits	Name	Function
[31:30]	-	Reserved. Read as `0b00`.
[29]	MasterType	0 = core. 1 = debug. This bit must not be changed if a transaction is outstanding. A debugger must first check bit [7], TransInProg. Reset value = `0b1`. An implementation can configure this bit to be read only with a value of 1. In that case, transactions are always indicated as debug. Note: When clear, this bit prevents the debugger from setting the C_DEBUGEN bit in the Debug Halting Control and Status Register, and so prevents the debugger from being able to halt the processor.
[28:26]	-	Reserved, `0b000`.
[25]	Hprot1	User and Privilege control - HPROT[1]. Reset value = `0b1`.
[24]	-	Reserved, `0b1`.
[23:12]	-	Reserved, `0x000`.
[11:8]	Mode	Mode of operation bits: `0b0000` = normal download and upload mode `0b0001-0b1111` are reserved. Reset value = `0b0000`.
[7]	TransInProg	Transfer in progress. This field indicates if a transfer is in progress on the AHB master port.

Table 8-6 CSW bit assignments (continued)

Bits	Name	Function
[6]	DbgStatus	Indicates the status of the DAPEN port. 0 = AHB transfers not permitted. 1 = AHB transfers permitted.
[5:4]	AddrInc	Auto address increment and pack mode on Read or Write data access. Only increments if the current transaction completes with no error. Auto address incrementing and packed transfers are not performed on access to Banked Data registers 0x10 - `0x1C`. The status of these bits is ignored in these cases. Increments and wraps within a 4KB address boundary, for example from `0x1000` to `0x1FFC`. If the start is at `0x14A0`, the counter increments to `0x1FFC`, wraps to `0x1000`, then continues incrementing to `0x149C`. `0b00` = auto increment off. `0b01` = increment single. Single transfer from corresponding byte lane. `0b10` = increment packed. See the definition of packed transfers in the *ARM® Debug Interface v5 Architecture Specification* `0b11` = reserved. No transfer. Size of address increment is defined by the Size field [2:0]. Reset value: `0b00`.
[3]	-	Reserved.
[2:0]	Size	Size of access field: `0b000` = 8 bits `0b001` = 16 bits `0b010` = 32 bits `0b011`-111 are reserved. Reset value: `0b000`.

8.3 Flash Patch and Breakpoint Unit (FPB)

The Cortex-M4 processor contains a Flash Patch and Breakpoint (FPB) unit that implements hardware breakpoints, and patches code and data from Code space to System space.

This section contains the following subsections:

• *8.3.1 FPB full and reduced units* on page 8-82.
• *8.3.2 FPB functional description* on page 8-82.
• *8.3.3 FPB programmers' model* on page 8-83.

8.3.1 FPB full and reduced units

The FPB is available as a full unit or as a reduced unit.

A full FPB unit contains:

- Two literal comparators for matching against literal loads from Code space, and remapping to a corresponding area in System space.
- Six instruction comparators for matching against instruction fetches from Code space, and remapping to a corresponding area in System space. Alternatively, you can configure the comparators individually to return a *Breakpoint Instruction* (`BKPT`) to the processor core on a match, to provide hardware breakpoint capability.

A reduced FPB unit contains:

- Two instruction comparators. You can configure each comparator individually to return a Breakpoint Instruction to the processor on a match, to provide hardware breakpoint capability.

8.3.2 FPB functional description

The FPB contains both a global enable and individual enables for the eight comparators.

If the comparison for an entry matches, the address is either:

- Remapped to the address set in the remap register plus an offset corresponding to the comparator that matched.
- Remapped to a `BKPT` instruction if that feature is enabled.

The comparison happens dynamically, but the result of the comparison occurs too late to stop the original instruction fetch or literal load taking place from the Code space. The processor ignores this transaction however, and only the remapped transaction is used.

If an MPU is present, the MPU lookups are performed for the original address, not the remapped address.

You can remove the FPB if no debug is required, or you can reduce the number of breakpoints it supports to two. If the FPB supports only two breakpoints then only comparators 0 and 1 are used, and the FPB does not support flash patching.

Note

- Unaligned literal accesses are not remapped. The original access to the DCode bus takes place in this case.
- Load exclusive accesses can be remapped. However, it is UNPREDICTABLE whether they are performed as exclusive accesses or not.
- Setting the flash patch remap location to a bit-band alias is not supported and results in UNPREDICTABLE behavior.

8.3.3 FPB programmers' model

Table showing the FPB registers. Depending on the implementation of your processor, some of the registers might not be present. Any register that is configured as not present reads as zero.

Table 8-7 FPB register summary

Address	Name	Type	Reset	Description	Notes
0xE0002000	FP_CTRL	RW	0x260	FlashPatch Control Register	
0xE0002004	FP_REMAP	RW	-	FlashPatch Remap Register	
0xE0002008	FP_COMP0	RW	b0	FlashPatch Comparator Register0	For FP_COMP0 to FP_COMP7, bit 0 is reset to 0. Other bits in these registers are not reset.
0xE000200C	FP_COMP1	RW	b0	FlashPatch Comparator Register1	
0xE0002010	FP_COMP2	RW	b0	FlashPatch Comparator Register2	
0xE0002014	FP_COMP3	RW	b0	FlashPatch Comparator Register3	
0xE0002018	FP_COMP4	RW	b0	FlashPatch Comparator Register4	
0xE000201C	FP_COMP5	RW	b0	FlashPatch Comparator Register5	
0xE0002020	FP_COMP6	RW	b0	FlashPatch Comparator Register6	
0xE0002024	FP_COMP7	RW	b0	FlashPatch Comparator Register7	
0xE0002FD0	PID4	RO	0x04	Peripheral identification registers	
0xE0002FD4	PID5	RO	0x00		
0xE0002FD8	PID6	RO	0x00		
0xE0002FDC	PID7	RO	0x00		
0xE0002FE0	PID0	RO	0x03		
0xE0002FE4	PID1	RO	0xB0		
0xE0002FE8	PID2	RO	0x2B		
0xE0002FEC	PID3	RO	0x00		
0xE0002FF0	CID0	RO	0x0D	Component identification registers	
0xE0002FF4	CID1	RO	0xE0		
0xE0002FF8	CID2	RO	0x05		
0xE0002FFC	CID3	RO	0xB1		

All FPB registers are described in the *ARMv7-M Architecture Reference Manual*.

Chapter 9
Data Watchpoint and Trace Unit

This chapter describes the *Data Watchpoint and Trace* (DWT) unit.

It contains the following sections:

- *9.1 DWT functional description* on page 9-85.
- *9.2 DWT Programmers' model* on page 9-86.

9.1 DWT functional description

A full DWT contains four comparators that you can configure as hardware watchpoint, an ETM trigger, a PC sampler event trigger, or a data address sampler event trigger.

The first comparator, DWT_COMP0, can also compare against the clock cycle counter, CYCCNT. You can also use the second comparator, DWT_COMP1, as a data comparator.

A reduced DWT contains one comparator that you can use as a watchpoint or as a trigger. It does not support data matching.

The DWT, if present, contains counters for:

- Clock cycles (CYCCNT).
- Folded instructions.
- *Load Store Unit* (LSU) operations.
- Sleep cycles.
- CPI, that is all instruction cycles except for the first cycle.
- Interrupt overhead.

——— **Note** ———

An event is generated each time a counter overflows.

———————————

You can configure the DWT to generate PC samples at defined intervals, and to generate interrupt event information.

The DWT provides periodic requests for protocol synchronization to the ITM and the TPIU, if your implementation includes the Cortex-M4 TPIU.

9.2 DWT Programmers' model

Table showing the DWT registers. Depending on the implementation of your processor, some of these registers might not be present. Any register that is configured as not present reads as zero.

Table 9-1 DWT register summary

Address	Name	Type	Reset	Description
`0xE0001000`	DWT_CTRL	RW	Possible reset values are: • 0x40000000 if four comparators for watchpoints and triggers are present. • 0x4F000000 if four comparators for watchpoints only are present. • 0x10000000 if only one comparator is present. • 0x1F000000 if one comparator for watchpoints and not triggers is present. • 0x00000000 if DWT is not present.	Control Register.
`0xE0001004`	DWT_CYCCNT	RW	`0x00000000`	Cycle Count Register
`0xE0001008`	DWT_CPICNT	RW	-	CPI Count Register
`0xE000100C`	DWT_EXCCNT	RW	-	Exception Overhead Count Register
`0xE0001010`	DWT_SLEEPCNT	RW	-	Sleep Count Register
`0xE0001014`	DWT_LSUCNT	RW	-	LSU Count Register
`0xE0001018`	DWT_FOLDCNT	RW	-	Folded-instruction Count Register
`0xE000101C`	DWT_PCSR	RO	-	Program Counter Sample Register
`0xE0001020`	DWT_COMP0	RW	-	Comparator Register0
`0xE0001024`	DWT_MASK0	RW	-	Mask Register0. The maximum mask size is 32KB.
`0xE0001028`	DWT_FUNCTION0	RW	`0x00000000`	Function Register0
`0xE0001030`	DWT_COMP1	RW	-	Comparator Register1
`0xE0001034`	DWT_MASK1	RW	-	Mask Register1. The maximum mask size is 32KB.
`0xE0001038`	DWT_FUNCTION1	RW	`0x00000000`	Function Register1
`0xE0001040`	DWT_COMP2	RW	-	Comparator Register2
`0xE0001044`	DWT_MASK2	RW	-	Mask Register2. The maximum mask size is 32KB.
`0xE0001048`	DWT_FUNCTION2	RW	`0x00000000`	Function Register2
`0xE0001050`	DWT_COMP3	RW	-	Comparator Register3
`0xE0001054`	DWT_MASK3	RW	-	Mask Register3. The maximum mask size is 32KB.
`0xE0001058`	DWT_FUNCTION3	RW	`0x00000000`	Function Register3

Table 9-1 DWT register summary (continued)

Address	Name	Type	Reset	Description
`0xE0001FD0`	PID4	RO	`0x04`	Peripheral identification registers
`0xE0001FD4`	PID5	RO	`0x00`	
`0xE0001FD8`	PID6	RO	`0x00`	
`0xE0001FDC`	PID7	RO	`0x00`	
`0xE0001FE0`	PID0	RO	`0x02`	
`0xE0001FE4`	PID1	RO	`0xB0`	
`0xE0001FE8`	PID2	RO	`0x3B`	
`0xE0001FEC`	PID3	RO	`0x00`	
`0xE0001FF0`	CID0	RO	`0x0D`	Component identification registers
`0xE0001FF4`	CID1	RO	`0xE0`	
`0xE0001FF8`	CID2	RO	`0x05`	
`0xE0001FFC`	CID3	RO	`0xB1`	

DWT registers are described in the *ARM®v7M Architecture Reference Manual*. Peripheral Identification and Component Identification registers are described in the *ARM® CoreSight™ Components Technical Reference Manual*.

Note

- Cycle matching functionality is only available in comparator 0.
- Data matching functionality is only available in comparator 1.
- Data value is only sampled for accesses that do not produce an MPU or bus fault. The PC is sampled irrespective of any faults. The PC is only sampled for the first address of a burst.
- The FUNCTION field in the DWT_FUNCTION1 register is overridden for comparators given by DATAVADDR0 and DATAVADDR1 if DATAVMATCH is also set in DWT_FUNCTION1. The comparators given by DATAVADDR0 and DATAVADDR1 can then only perform address comparator matches for comparator 1 data matches.
- If the data matching functionality is not included during implementation it is not possible to set DATAVADDR0, DATAVADDR1, or DATAVMATCH in DWT_FUNCTION1. This means that the data matching functionality is not available in the implementation. Test the availability of data matching by writing and reading the DATAVMATCH bit in DWT_FUNCTION1. If this bit cannot be set then data matching is unavailable.
- ARM does not recommend PC match for watchpoints because it stops after the instruction. It mainly guards and triggers the ETM.

Chapter 10
Instrumentation Trace Macrocell Unit

This chapter describes the *Instrumentation Trace Macrocell* (ITM) unit.

It contains the following sections:

- *10.1 ITM functional description* on page 10-89.
- *10.2 ITM programmers' model* on page 10-90.
- *10.3 ITM Trace Privilege Register, ITM_TPR* on page 10-91.

10.1 ITM functional description

The ITM is a an optional application-driven trace source that supports `printf()` style debugging to trace operating system and application events, and generates diagnostic system information. The ITM generates trace information as packets from software traces, hardware traces, time stamping, and global system timestamping sources.

The ITM generates trace information as packets. There are four sources that can generate packets. If multiple sources generate packets at the same time, the ITM arbitrates the order in which packets are output. The four sources in decreasing order of priority are:

- Software trace. Software can write directly to ITM stimulus registers to generate packets.
- Hardware trace. The DWT generates these packets, and the ITM outputs them.
- Time stamping. Timestamps are generated relative to packets. The ITM contains a 21-bit counter to generate the timestamp. The Cortex-M4 clock or the bitclock rate of the *Serial Wire Viewer* (SWV) output clocks the counter.
- Global system timestamping. Timestamps can optionally be generated using a system-wide 48-bit count value. The same count value can be used to insert timestamps in the ETM trace stream, permitting coarse-grain correlation.

10.2 ITM programmers' model

Table showing the ITM registers. Depending on the implementation of your processor, the ITM registers might not be present. Any register that is configured as not present reads as zero.

Note

- You must enable TRCENA of the Debug Exception and Monitor Control Register before you program or use the ITM.
- If the ITM stream requires synchronization packets, you must configure the synchronization packet rate in the DWT.

Table 10-1 ITM register summary

Address	Name	Type	Reset	Description
0xE0000000- 0xE000007C	ITM_STIM0- ITM_STIM31	RW	-	Stimulus Port Registers 0-31
0xE0000E00	ITM_TER	RW	0x00000000	Trace Enable Register
0xE0000E40	ITM_TPR	RW	0x00000000	Refer to the ITM Trace Privilege Register description
0xE0000E80	ITM_TCR	RW	0x00000000	Trace Control Register
0xE0000FD0	PID4	RO	0x00000004	Peripheral Identification registers
0xE0000FD4	PID5	RO	0x00000000	
0xE0000FD8	PID6	RO	0x00000000	
0xE0000FDC	PID7	RO	0x00000000	
0xE0000FE0	PID0	RO	0x00000001	
0xE0000FE4	PID1	RO	0x000000B0	
0xE0000FE8	PID2	RO	0x0000003B	
0xE0000FEC	PID3	RO	0x00000000	
0xE0000FF0	CID0	RO	0x0000000D	Component Identification registers
0xE0000FF4	CID1	RO	0x000000E0	
0xE0000FF8	CID2	RO	0x00000005	
0xE0000FFC	CID3	RO	0x000000B1	

Note

ITM registers are fully accessible in privileged mode. In user mode, all registers can be read, but only the Stimulus Registers and Trace Enable Registers can be written, and only when the corresponding Trace Privilege Register bit is set. Invalid user mode writes to the ITM registers are discarded.

The following section describes the ITM registers whose implementation is specific to this processor. Other registers are described in the *ARM®v7-M Architectural Reference Manual*.

10.3 ITM Trace Privilege Register, ITM_TPR

Characteristics and bit assignments.of the ITM_TPR register.

Purpose
Enables an operating system to control the stimulus ports that are accessible by user code.

Usage constraints
You can only write to this register in privileged mode.

Configurations
This register is available if the ITM is configured in your implementation.

Attributes
Refer to the ITM register summary table.

The following figure shows the ITM_TPR bit assignments.

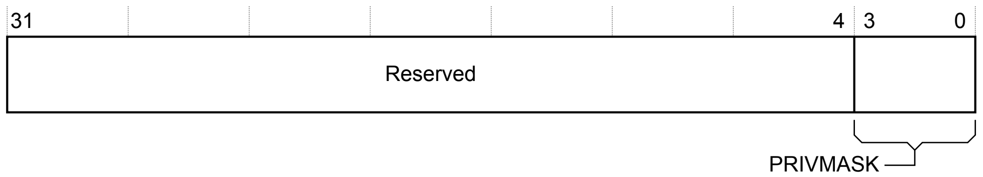


Figure 10-1 ITM_TPR bit assignments

The following table shows the ITM_TPR bit assignments.

Table 10-2 ITM_TPR bit assignments

Bits	Name	Function
[31:4]	-	Reserved.
[3:0]	PRIVMASK	Bit mask to enable tracing on ITM stimulus ports: bit [0] = stimulus ports [7:0] bit [1] = stimulus ports [15:8] bit [2] = stimulus ports [23:16] bit [3] = stimulus ports [31:24].

Chapter 11
Trace Port Interface Unit

This chapter describes the Cortex-M4 TPIU, the Trace Port Interface Unit that is specific to the Cortex-M4 processor.

It contains the following sections:

- *11.1 About the TPIU* on page 11-93.
- *11.2 TPIU functional description* on page 11-94.
- *11.3 TPIU programmers' model* on page 11-96.

11.1 About the TPIU

The Cortex-M4 TPIU is an optional component that acts as a bridge between the on-chip trace data from the Embedded Trace Macrocell (ETM) and the Instrumentation Trace Macrocell (ITM), with separate IDs, to a data stream. The TPIU encapsulates IDs where required, and the data stream is then captured by a Trace Port Analyzer (TPA).

The Cortex-M4 TPIU is specially designed for low-cost debug. It is a special version of the CoreSight TPIU. Your implementation can replace the Cortex-M4 TPIU with other CoreSight components if your implementation requires the additional features of the CoreSight TPIU.

In this chapter, the term TPIU refers to the Cortex-M4 TPIU. For information about the CoreSight TPIU, see the *ARM® CoreSight™ Components Technical Reference Manual*.

11.2 TPIU functional description

The TPIU is available in a configuration that supports ITM debug trace, and a configuration that supports both ITM and ETM debug trace. If your implementation requires no trace support then the TPIU might not be present.

Note

If your Cortex-M4 system uses the optional ETM component, the TPIU configuration supports both ITM and ETM debug trace. See the *ETM-M4 Technical Reference Manual*.

This section contains the following subsections:
- *11.2.1 TPIU block diagram* on page 11-94.
- *11.2.2 TPIU formatter* on page 11-94.
- *11.2.3 Serial Wire Output format* on page 11-95.

11.2.1 TPIU block diagram

Block diagram showing the component layout of the TPIU.

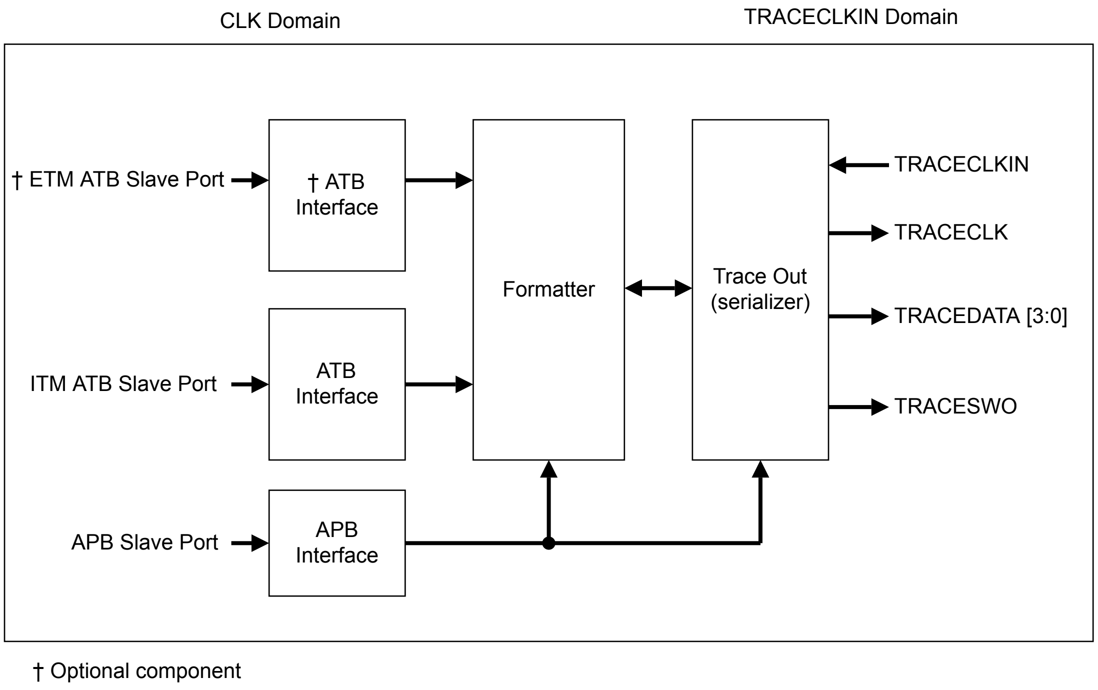


Figure 11-1 TPIU block diagram

11.2.2 TPIU formatter

The TPIU formatter inserts source ID signals into the data packet stream so that trace data can be re-associated with its trace source. The formatter is always active when the Trace Port Mode is active.

The formatting protocol is described in the *CoreSight™ Architecture Specification*. You must enable synchronization packets in the DWT to provide synchronization for the formatter.

When the formatter is enabled, half-sync packets can be inserted if there is no data to output after a frame has been started. Synchronization, caused by the distributed synchronization from the DWT, ensures that any partial frame is completed, and at least one full synchronization packet is generated.

11.2.3 Serial Wire Output format

The TPIU can output trace data in TPIU_DEVID or TPIU_SPPR Serial Wire Output formats and can be configured to bypass the formatter for trace output if either SWO format is selected.

The TPIU can output trace data in a *Serial Wire Output* (SWO) format:

- TPIU_DEVID specifies the formats that are supported.
- TPIU_SPPR specifies the SWO format in use. See the *ARM®v7-M Architecture Reference Manual*.

When one of the two SWO modes is selected, you can enable the TPIU to bypass the formatter for trace output. If the formatter is bypassed, only the ITM and DWT trace source passes through. The TPIU accepts and discards data from the ETM. This function can be used to connect a device containing an ETM to a trace capture device that is only able to capture SWO data.

Related references

11.3.11 TPIU_DEVID on page 11-104.

11.3 TPIU programmers' model

The programmers' model enables you to use the TPIU registers to set up the Trace Port Interface Unit.

The Formatter, Integration Mode Control, and Claim Tag registers are described in the *CoreSight™ Components Technical Reference Manual*. Other registers are described in the *ARM®v7-M Architecture Reference Manual*.

This section contains the following subsections:

- *11.3.1 TPIU registers* on page 11-96.
- *11.3.2 Asynchronous Clock Prescaler Register, TPIU_ACPR* on page 11-97.
- *11.3.3 Formatter and Flush Status Register, TPIU_FFSR* on page 11-98.
- *11.3.4 Formatter and Flush Control Register, TPIU_FFCR* on page 11-99.
- *11.3.5 TRIGGER* on page 11-100.
- *11.3.6 Integration ETM Data* on page 11-100.
- *11.3.7 ITATBCTR2* on page 11-101.
- *11.3.8 Integration ITM Data* on page 11-102.
- *11.3.9 ITATBCTR0* on page 11-102.
- *11.3.10 Integration Mode Control, TPIU_ITCTRL* on page 11-103.
- *11.3.11 TPIU_DEVID* on page 11-104.
- *11.3.12 TPIU_DEVTYPE* on page 11-105.

11.3.1 TPIU registers

The table shows the TPIU registers. Depending on the implementation of your processor, the TPIU registers might not be present, or the CoreSight TPIU might be present instead. Any register that is configured as not present reads as zero.

Table 11-1 TPIU registers

Address	Name	Type	Reset	Description
`0xE0040000`	TPIU_SSPSR	RO	`0x0xx`	Supported Parallel Port Size Register
`0xE0040004`	TPIU_CSPSR	RW	`0x01`	Current Parallel Port Size Register
`0xE0040010`	TPIU_ACPR	RW	`0x0000`	Asynchronous Clock Prescaler Register, TPIU_ACPR
`0xE00400F0`	TPIU_SPPR	RW	`0x01`	Selected Pin Protocol Register
`0xE0040300`	TPIU_FFSR	RO	`0x08`	Formatter and Flush Status Register, TPIU_FFSR
`0xE0040304`	TPIU_FFCR	RW	`0x102`	Formatter and Flush Control Register, TPIU_FFCR
`0xE0040308`	TPIU_FSCR	RO	`0x00`	Formatter Synchronization Counter Register
`0xE0040EE8`	TRIGGER	RO	`0x0`	TRIGGER register
`0xE0040EEC`	FIFO data 0	RO	`0x--000000`	Integration ETM Data
`0xE0040EF0`	ITATBCTR2	RO	`0x0`	ITATBCTR2
`0xE0040EFC`	FIFO data 1	RO	`0x--000000`	Integration ITM Data
`0xE0040EF8`	ITATBCTR0	RO	`0x0`	ITATBCTR0
`0xE0040F00`	ITCTRL	RW	`0x0`	Integration Mode Control, TPIU_ITCTRL
`0xE0040FA0`	CLAIMSET	RW	`0xF`	Claim tag set
`0xE0040FA4`	CLAIMCLR	RW	`0x0`	Claim tag clear
`0xE0040FC8`	DEVID	RO	`0xCA0/0xCA1`	TPIU_DEVID

Table 11-1 TPIU registers (continued)

Address	Name	Type	Reset	Description
`0xE0040FCC`	DEVTYPE	RO	`0x11`	TPIU_DEVTYPE
`0xE0040FD0`	PID4	RO	`0x04`	Peripheral identification registers
`0xE0040FD4`	PID5	RO	`0x00`	
`0xE0040FD8`	PID6	RO	`0x00`	
`0xE0040FDC`	PID7	RO	`0x00`	
`0xE0040FE0`	PID0	RO	`0xA1`	
`0xE0040FE4`	PID1	RO	`0xB9`	
`0xE0040FE8`	PID2	RO	`0x0B`	
`0xE0040FEC`	PID3	RO	`0x00`	
`0xE0040FF0`	CID0	RO	`0x0D`	Component identification registers
`0xE0040FF4`	CID1	RO	`0x90`	
`0xE0040FF8`	CID2	RO	`0x05`	
`0xE0040FFC`	CID3	RO	`0xB1`	

Related references

11.3.2 Asynchronous Clock Prescaler Register, TPIU_ACPR on page 11-97.
11.3.3 Formatter and Flush Status Register, TPIU_FFSR on page 11-98.
11.3.4 Formatter and Flush Control Register, TPIU_FFCR on page 11-99.
11.3.5 TRIGGER on page 11-100.
11.3.6 Integration ETM Data on page 11-100.
11.3.8 Integration ITM Data on page 11-102.
11.3.7 ITATBCTR2 on page 11-101.
11.3.10 Integration Mode Control, TPIU_ITCTRL on page 11-103.
11.3.9 ITATBCTR0 on page 11-102.
11.3.11 TPIU_DEVID on page 11-104.
11.3.12 TPIU_DEVTYPE on page 11-105.

11.3.2 Asynchronous Clock Prescaler Register, TPIU_ACPR

Characteristics and bit assignments of the TPIU_ACPR register.

Purpose
Scales the baud rate of the asynchronous output.

Usage constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the TPIU_ACPR bit assignments.

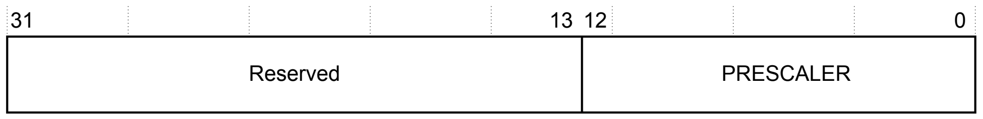


Figure 11-2 TPIU_ACPR bit assignments

The following table shows the TPIU_ACPR bit assignments.

Table 11-2 TPIU_ACPR bit assignments

Bits	Name	Function
[31:13]	-	Reserved. RAZ/SBZP.
[12:0]	PRESCALER	Divisor for TRACECLKIN is Prescaler + 1.

11.3.3 Formatter and Flush Status Register, TPIU_FFSR

Characteristics and bit assignments.of the TPIU_FFSR register.

Purpose
Indicates the status of the TPIU formatter.

Usage constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the TPIU_FFSR bit assignments.

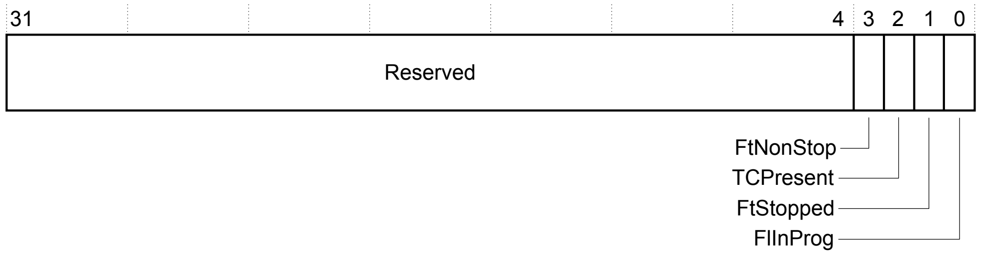


Figure 11-3 TPIU_FFSR bit assignments

The following table shows the TPIU_FFSR bit assignments.

Table 11-3 TPIU_FFSR bit assignments

Bits	Name	Function
[31:4]	-	Reserved
[3]	FtNonStop	Formatter cannot be stopped
[2]	TCPresent	This bit always reads zero
[1]	FtStopped	This bit always reads zero
[0]	FlInProg	This bit always reads zero

11.3.4 Formatter and Flush Control Register, TPIU_FFCR

Characteristics and bit assignments of the TPIU_FFSR register.

Purpose
Controls the TPIU formatter.

Usage constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the TPIU_FFCR bit assignments.

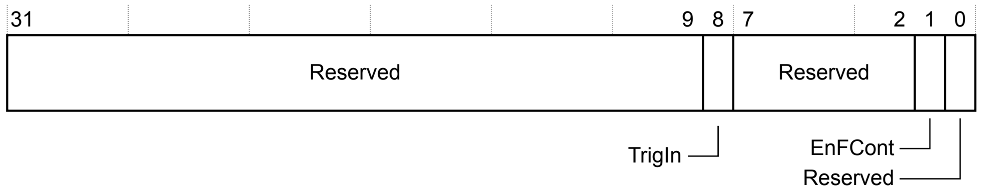


Figure 11-4 TPIU_FFCR bit assignments

The following table shows the TPIU_FFCR bit assignments.

Table 11-4 TPIU_FFCR bit assignments

Bits	Name	Function
[31:9]	-	Reserved.
[8]	TrigIn	This bit Reads-As-One (RAO), specifying that triggers are inserted when a trigger pin is asserted.
[7:2]	-	Reserved.
[1]	EnFCont	Enable continuous formatting. Value can be: 0 = Continuous formatting disabled. 1 = Continuous formatting enabled.
[0]	-	Reserved.

The TPIU can output trace data in a *Serial Wire Output* (SWO) format.

When one of the two SWO modes is selected, bit [1] of TPIU_FFCR enables the formatter to be bypassed. If the formatter is bypassed, only the ITM and DWT trace source passes through. The TPIU accepts and discards data from the ETM. This function can be used to connect a device containing an ETM to a trace capture device that is only able to capture SWO data. Enabling or disabling the formatter causes momentary data corruption.

Note

If TPIU_SPPR is set to select Trace Port Mode, the formatter is automatically enabled. If you then select one of the SWO modes, TPIU_FFCR reverts to its previously programmed value.

Related concepts

11.2.3 Serial Wire Output format on page 11-95.

11.3.5 TRIGGER

Characteristics and bit assignments.of the TRIGGER.

Purpose
Integration test of the TRIGGER input.

Usage constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the TRIGGER bit assignments.

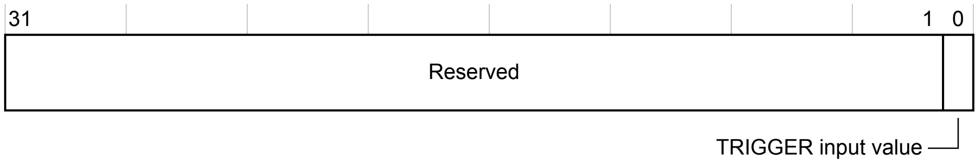


Figure 11-5 TRIGGER bit assignments

The following table shows the TRIGGER bit assignments.

Table 11-5 TRIGGER bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	TRIGGER input value	When read, this bit returns the TRIGGER input.

11.3.6 Integration ETM Data

Characteristics and bit assignments of the Integration ETM Data register.

Purpose
Trace data integration testing.

Usage constraints
You must set bit [1] of TPIU_ITCTRL to use this register.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the Integration ETM Data bit assignments.

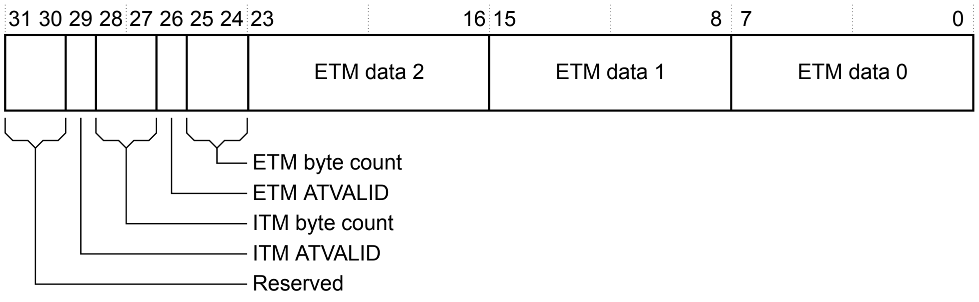


Figure 11-6 Integration ETM Data bit assignments

The following table shows the Integration ETM Data bit assignments.

Table 11-6 Integration ETM Data bit assignments

Bits	Name	Function
[31:30]	-	Reserved
[29]	ITM ATVALID input	Returns the value of the ITM ATVALID signal.
[28:27]	ITM byte count	Number of bytes of ITM trace data since last read of Integration ITM Data Register.
[26]	ETM ATVALID input	Returns the value of the ETM ATVALID signal.
[25:24]	ETM byte count	Number of bytes of ETM trace data since last read of Integration ETM Data Register.
[23:16]	ETM data 2	ETM trace data. The TPIU discards this data when the register is read.
[15:8]	ETM data 1	
[7:0]	ETM data 0	

Related references

11.3.10 Integration Mode Control, TPIU_ITCTRL on page 11-103.

11.3.7 ITATBCTR2

Characteristics and bit assignments of the ITATBCTR2 register.

Purpose
Integration test.

Usage constraints
You must set bit [0] of TPIU_ITCTRL to use this register.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the ITATBCTR2 bit assignments.

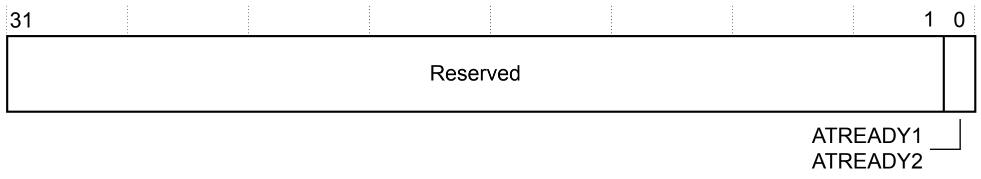


Figure 11-7 ITATBCTR2 bit assignments

The following table shows the ITATBCTR2 bit assignments.

Table 11-7 ITATBCTR2 bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	ATREADY1, ATREADY2	This bit sets the value of both the ETM and ITM ATREADY outputs, if the TPIU is in integration test mode.

Related references

11.3.10 Integration Mode Control, TPIU_ITCTRL on page 11-103.

11.3.8 Integration ITM Data

Characteristics and bit assignments of the Integration ITM Data register.

Purpose
Trace data integration testing.

Usage constraints
You must set bit [1] of TPIU_ITCTRL to use this register.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the Integration ITM Data bit assignments.

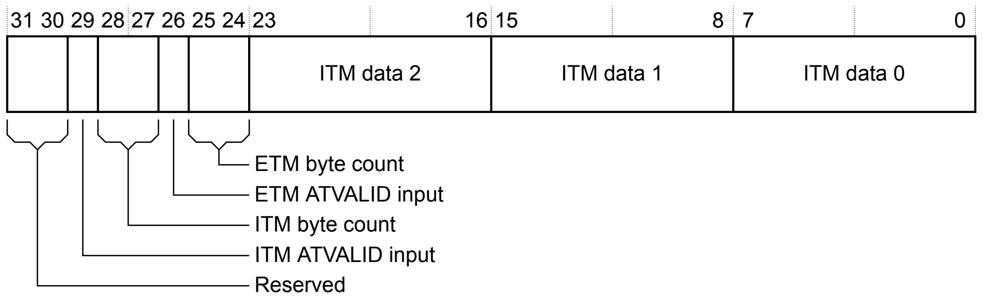


Figure 11-8 Integration ITM Data bit assignments

The following table shows the Integration ITM Data bit assignments.

Table 11-8 Integration ITM Data bit assignments

Bits	Name	Function
[31:30]	-	Reserved
[29]	ITM ATVALID input	Returns the value of the ITM ATVALID signal.
[28:27]	ITM byte count	Number of bytes of ITM trace data since last read of Integration ITM Data Register.
[26]	ETM ATVALID input	Returns the value of the ETM ATVALID signal.
[25:24]	ETM byte count	Number of bytes of ETM trace data since last read of Integration ETM Data Register.
[23:16]	ITM data 2	ITM trace data. The TPIU discards this data when the register is read.
[15:8]	ITM data 1	
[7:0]	ITM data 0	

Related references

11.3.10 Integration Mode Control, TPIU_ITCTRL on page 11-103.

11.3.9 ITATBCTR0

Characteristics and bit assignments of the ITATBCTR0 register.

Purpose
Integration test.

Usage constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes

Refer to the TPIU register table.

The following figure shows the ITATBCTR0 bit assignments.

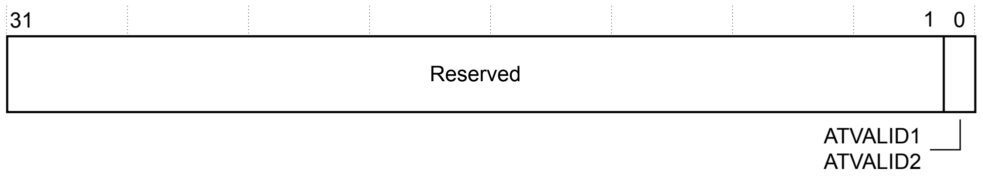


Figure 11-9 ITATBCTR0 bit assignments

The following table shows the ITATBCTR0 bit assignments.

Table 11-9 ITATBCTR0 bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	ATVALID1, ATVALID2	A read of this bit returns the value of ATVALIDS1 OR-ed with ATVALIDS2.

11.3.10 Integration Mode Control, TPIU_ITCTRL

Characteristics and bit assignments of the TPIU_ITCTRL register.

Purpose

Specifies normal or integration mode for the TPIU.

Usage constraints

There are no usage constraints.

Configurations

This register is available in all processor configurations.

Attributes

Refer to the TPIU register table.

The following figure shows the TPIU_ITCTRL bit assignments.

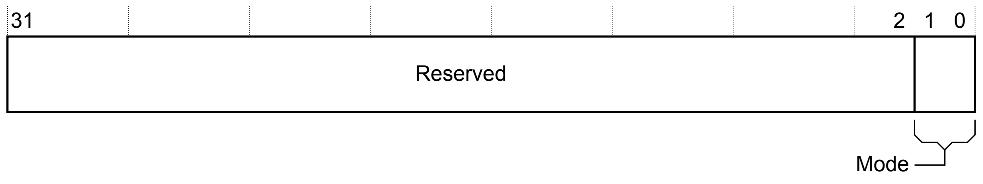


Figure 11-10 TPIU_ITCTRL bit assignments

The following table shows the TPIU_ITCTRL bit assignments.

Table 11-10 TPIU_ITCTRL bit assignments

Bits	Name	Function
[31:2]	-	Reserved.
[1:0]	Mode	Specifies the current mode for the TPIU: `b00` Normal mode. `b01` Integration test mode. `b10` Integration data test mode. `b11` Reserved. In integration data test mode, the trace output is disabled, and data can be read directly from each input port using the integration data registers.

11.3.11 TPIU_DEVID

Characteristics and bit assignments of the TPIU_DEVID register.

Purpose
Indicates the functions provided by the TPIU for use in topology detection.

Usage constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
Refer to the TPIU register table.

The following figure shows the TPIU_DEVID bit assignments.

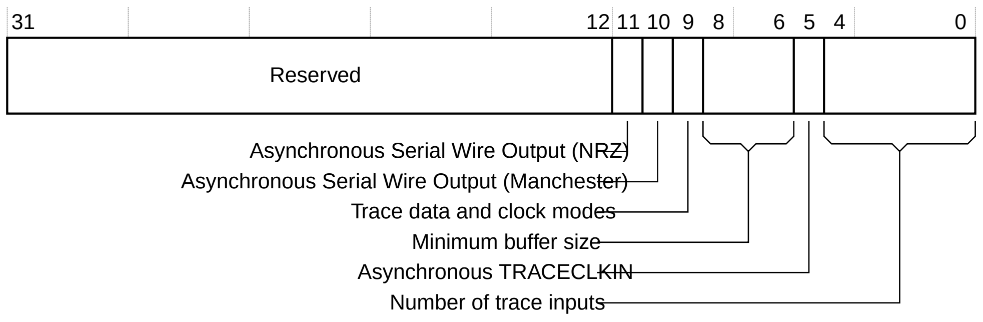

31
12 11 10 9 8 6 5 4 0
Reserved
Asynchronous Serial Wire Output (NRZ)
Asynchronous Serial Wire Output (Manchester)
Trace data and clock modes
Minimum buffer size
Asynchronous TRACECLKIN
Number of trace inputs

Figure 11-11 TPIU_DEVID bit assignments

The following table shows the TPIU_DEVID bit assignments.

Table 11-11 TPIU_DEVID bit assignments

Bits	Name	Function
[31:12]	-	Reserved
[11]	Asynchronous Serial Wire Output (NRZ)	This bit Reads-As-One (RAO), indicating that the output is supported.
[10]	Asynchronous Serial Wire Output (Manchester)	This bit Reads-As-One (RAO), indicating that the output is supported.

Table 11-11 TPIU_DEVID bit assignments (continued)

Bits	Name	Function
[9]	Trace data and clock modes	This bit Reads-As-Zero (RAZ), indicating that trace data and clock modes are supported
[8:6]	Minimum buffer size	Specifies the minimum TPIU buffer size: `b010` = 4 bytes
[5]	Asynchronous **TRACECLKIN**	Specifies whether **TRACECLKIN** can be asynchronous to **CLK** `b0` = **TRACECLKIN** must be synchronous to **CLK** `b1` = **TRACECLKIN** can be asynchronous to **CLK**
[4:0]	Number of trace inputs	Specifies the number of trace inputs: `b00000` = 1 input `b00001` = 2 inputs If your implementation includes an ETM, the value of this field is `b00001`.

11.3.12 TPIU_DEVTYPE

The Device Type Identifier Register is read-only. It provides a debugger with information about the component when the Part Number field is not recognized. The debugger can then report this information.

The TPIU_DEVTYPE characteristics are:

Purpose
Indicates the type of functionality the component supports.

Usage Constraints
There are no usage constraints.

Configurations
This register is available in all processor configurations.

Attributes
The Device Type reads as `0x11` and indicates this device is a trace sink and specifically a TPIU.

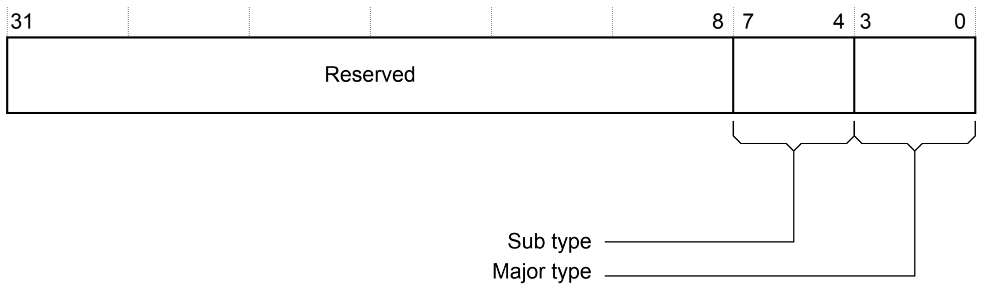


Figure 11-12 TPIU_DEVTYPE bit assignments

Appendix A
Revisions

The technical changes between released issues of this manual.

It contains the following sections:

- *A.1 Revisions* on page Appx-A-107.

A.1 Revisions

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

Table A-1 Issue A

Change	Location	Affects
First release	-	-

Table A-2 Differences between issue A and issue B

Change	Location	Affects
No technical changes	-	-

Table A-3 Differences between issue B and issue C

Change	Location	Affects
Additional information on bus interfaces.	*2.3.1 Bus interfaces* on page 2-24	All
Additional information on Private Peripheral Bus.	*Private Peripheral Bus (PPB)* on page 2-25	All
Updated the Cortex-M4 instruction set cycle times.	*3.3.1 Table of processor instructions* on page 3-30	All
Updated assembler of the signed multiply instructions for DSP instructions.	*3.3.2 Table of processor DSP instructions* on page 3-35	All
Updated information on Load/store timings.	*3.3.3 Load/store timings* on page 3-37	All
Added information on local exclusive monitor.	*3.6 Exclusive monitor* on page 3-44	All
Reset values updated.	*5.3 MPU programmers model table* on page 5-60	All
Updated bit order for Auxiliary Control Register.	*4.2 Auxiliary Control Register, ACTLR* on page 4-54	All
Updated bit order for Auxiliary Control Register.	*4.2 Auxiliary Control Register, ACTLR* on page 4-54	All
Updated information for Auxiliary Fault Status Register	*4.4 Auxiliary Fault Status Register, AFSR* on page 4-56	All
Changed address range for NVIC_IPR registers.	*6.2 NVIC programmers' model* on page 6-63	All
Added Peripheral IDs 5-7.	*8.1.3 ROM table identification and entries* on page 8-75	All
Updated reset value.	*8.3.3 FPB programmers' model* on page 8-83	All
Added names of TPIU registers. Reset values updated and added `TPIU_DEVTYPE`.	*11.3.1 TPIU registers* on page 11-96	All
Added `TPIU_DEVTYPE` bit assignments.	*11.3.12 TPIU_DEVTYPE* on page 11-105	All

Table A-4 Differences between issue C and issue D

Change	Location	Affects
Removed references to Cortex-M4F	*Chapter 7 Floating-Point Unit* on page 7-65 *Chapter 8 Debug* on page 8-73	All
Updated information for DCode memory interface	*DCode memory interface* on page 2-24	All
Updated footnotes a. and b. about division operations and Neighboring load and store single instructions respectively.	*3.3.1 Table of processor instructions* on page 3-30	All

Table A-4 Differences between issue C and issue D (continued)

Change	Location	Affects
Changed description for `SMULTT` signed multiply operation	*3.3.2 Table of processor DSP instructions* on page 3-35	All
Changed description of Cortex-M4 compatibility	*3.3.4 Binary compatibility with other Cortex processors* on page 3-38	All
Updated information for 16-bit instruction access to registers R8-R12	*3.8 Processor core register summary* on page 3-47	All
Specified access permission of CCR.STKALIGN bit	*4.1 System control registers* on page 4-52	All
Defined the ACTL. DISFPCA bit as SBZP	*4.2 Auxiliary Control Register, ACTLR* on page 4-54	All
Clarified address information for NVIC_ICER0-NVIC_ICER7 registers	*6.2.1 Table of NVIC registers* on page 6-63	All
Clarified lazy stacking information	*7.2.7 Exceptions* on page 7-71	All
Clarified that the FPU in the Cortex-M4 is implementation defined	*7.3 FPU programmers' model* on page 7-72	All
Clarified the latency issues when the FPU option is implemented and: • A floating point context is active and the lazy stacking is not enabled. • When an active floating point context is included in the stack frame.	*3.9.2 Interrupt latency* on page 3-49	All
Clarified Peripheral ID0 SCS identification value for implementations with and without FPU	*8.1.5 System Control Space registers* on page 8-77	All
Updated function information for CSW register bit[7]	*AHB-AP Control and Status Word Register, CSW* on page 8-80	All
Added footnote that clarifies mask size of DWT_MASK0, DWT_MASK1, DWT_MASK2, DWT_MASK3 registers.	*9.2 DWT Programmers' model* on page 9-86	All