Cortex™-M0+

Revision: r0p1

Technical Reference Manual

ARM®

ARM DDI 0484C (ID073015)

Cortex-M0+
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
19 January 2012	A	Confidential	First release for r0p0
04 April 2012	B	Non-Confidential	Second release for r0p0
16 December 2012	C	Non-Confidential	First release for r0p1

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

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

Web Address

`http://www.arm.com`

Contents
Cortex-M0+ Technical Reference Manual

Preface

This preface introduces the *Cortex-M0+ Technical Reference Manual*. It contains the following sections:

About this book

This book is for the Cortex-M0+ processor.

Product revision status

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

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

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

Intended audience

This book is written for:

- system designers, system integrators, and verification engineers
- software developers who want to use the processor.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the processor and its features.

Chapter 2 *Functional Description*

Read this chapter for a functional overview of the processor functions.

Chapter 3 *Programmers Model*

Read this chapter for an overview of the application-level programmers model.

Chapter 4 *System Control*

Read this chapter for a summary of the system control registers and their structure.

Chapter 5 *Nested Vectored Interrupt Controller*

Read this chapter for a summary of the *Nested Vectored Interrupt Controller* (NVIC).

Chapter 6 *Memory Protection Unit*

Read this chapter for a description of the *Memory Protection Unit* (MPU).

Chapter 7 *Debug*

Read this chapter for a summary of the debug system.

Appendix A *Revisions*

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

Glossary

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

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

Conventions

This book uses the conventions that are described in:

- *Typographical conventions*.

Typographical conventions

The following table describes the typographical conventions:

Style	Purpose
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *`italic`*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

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

- *ARMv6-M Architecture Reference Manual* (ARM DDI 0419).
- *ARMv6-M Instruction Set Quick Reference Guide* (ARM QRC 0011).
- *ARM AMBA® 3 AHB-Lite Protocol Specification* (ARM IHI 0033).
- *ARM Debug Interface v5, Architecture Specification* (ARM IHI 0031).

  Note

  A Cortex-M0+ implementation can include a *Debug Access Port* (DAP). This DAP is defined in v5.1 of the ARM Debug interface specification, or in the errata document to Issue A of the *ARM Debug Interface v5 Architecture Specification.*

- *Application Binary Interface for the ARM Architecture (The Base Standard)* (IHI 0036).
- *CoreSight™ SoC Technical Reference Manual* (ARM DDI 0480).
- *Cortex-M0+ Integration and Implementation Manual* (ARM DII 0278).
- *CoreSight MTB-M0+ Technical Reference Manual* (ARM DDI 0486).

Other publications

This section lists relevant documents published by third parties:

- IEEE Standard, *Test Access Port and Boundary-Scan Architecture specification* 1149.1-1990 (JTAG).

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

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

- The title.
- The number, ARM DDI 0484C.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Introduction

This chapter introduces the Cortex-M0+ processor and its features. It contains the following sections:

1.1 About the processor

The Cortex-M0+ processor is a very low gate count, highly energy efficient processor that is intended for microcontroller and deeply embedded applications that require an area optimized, low-power processor.

1.2 Features

The processor features and benefits are:

- Tight integration of system peripherals reduces area and development costs.
- Thumb instruction set combines high code density with 32-bit performance.
- Support for single-cycle I/O access.
- Power control optimization of system components.
- Integrated sleep modes for low power consumption.
- Fast code execution enables running the processor with a slower clock or increasing sleep mode time.
- Optimized code fetching for reduced flash and ROM power consumption.
- Hardware multiplier.
- Deterministic, high-performance interrupt handling for time-critical applications.
- Deterministic instruction cycle timing.
- Support for system level debug authentication.
- Serial Wire Debug reduces the number of pins required for debugging.
- Support for optional instruction trace.

For information about Cortex-M0+ architectural compliance, see the *Architecture and protocol information* on page 1-7.

1.3 Interfaces

The interfaces included in the processor for external access include:

- External AHB-Lite interface.
- *Debug Access Port* (DAP).
- Optional single-cycle I/O Port.

1.4 Configurable options

Table 1-1 shows the processor configurable options available at implementation time.

Table 1-1 Processor configurable options

Feature	Configurable option
Interrupts	External interrupts 0-32
Data endianness	Little-endian or big-endian
SysTick timer	Present or absent
Number of watchpoint comparators[a]	0, 1, 2
Number of breakpoint comparators[a]	0, 1, 2, 3, 4
Halting debug support	Present or absent
Multiplier	Fast or small
Single-cycle I/O port	Present or absent
Wake-up interrupt controller	Supported or not supported
Vector Table Offset Register	Present or absent
Unprivileged/Privileged support	Present or absent
Memory Protection Unit	Not present or 8-region
Reset all registers	Present or absent
Instruction fetch width	16-bit only or mostly 32-bit

a. Only when halting debug support is present.

1.4.1 Configurable multiplier

The `MULS` instruction provides a 32-bit x 32-bit multiply that returns the least-significant 32-bits of the result. The processor can implement `MULS` in one of two ways:

- As a fast single-cycle array.
- As a 32-cycle iterative multiplier.

The iterative multiplier has no impact on interrupt response time because the processor abandons multiply operations to take any pending interrupt.

1.5 Product documentation, design flow and architecture

This section describes the processor books, how they relate to the design flow, and the relevant architectural standards and protocols.

See *Additional reading* on page vii for more information about the books described in this section.

1.5.1 Documentation

This section describes the documents for the processor.

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the processor. It is required at all stages of the design flow. The choices made in the design flow can mean that some behavior described in the TRM is not relevant. If you are programming the 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 input configuration of the device 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 describing the pins that the integrator must tie off to configure the macrocell for the required integration.
- 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.

1.5.2 Design Flow

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

Implementation

The implementer configures and synthesizes the RTL to produce a gate-level layout. The implementer can do synthesis and layout before integration to produce a hard macrocell, or after integration to produce a chip layout.

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

Programming

This is the last process. The system programmer develops the software required to configure and initialize the processor, and tests the required application software.

Each process can be performed by a different party. The 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 synthesise 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 function 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 can be included by selecting the appropriate build configuration options. Reference to a feature that is included means the appropriate build and pin configuration options have been selected. References 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, the specifications described in:

- *ARM architecture*.
- *Advanced Microcontroller Bus Architecture* on page 1-8.
- *Debug Access Port architecture* on page 1-8.

This TRM 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 ARMv6-M architecture profile. See the *ARMv6-M Architecture Reference Manual*.

Advanced Microcontroller Bus Architecture

The system bus of the processor implements AMBA-3 AHB-Lite. See the *ARM AMBA 3 AHB-Lite Protocol Specification*.

Debug Access Port architecture

The *Debug Access Port* (DAP) is an optional component, defined by v5.1 of the ARM Debug interface specification, see the *ARM Debug Interface v5 Architecture Specification*.

1.6 Product revisions

This section describes the differences in functionality between product revisions.

r0p0 First release.

r0p1 No technical changes.

Chapter 2
Functional Description

This chapter provides an overview of the processor functions. It contains the following sections:

2.1 About the functions

The Cortex-M0+ processor is a configurable, multistage, 32-bit RISC processor. It has an AMBA AHB-Lite interface and includes an NVIC component. It also has optional hardware debug, single-cycle I/O interfacing, and memory-protection functionality. The processor can execute Thumb code and is compatible with other Cortex-M profile processors.

Figure 2-1 shows the functional blocks of the processor.

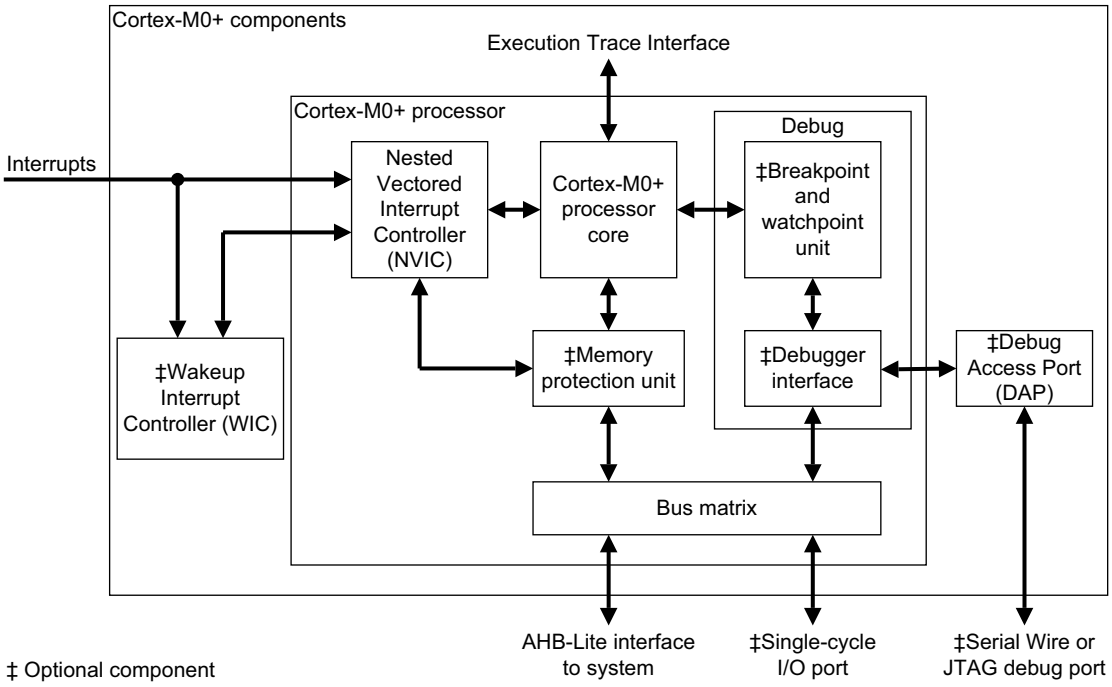

Figure 2-1 Functional block diagram

The implemented device provides:

A low gate count processor that features:

- The ARMv6-M Thumb® instruction set.
- Thumb-2 technology.
- Optionally, an ARMv6-M compliant 24-bit SysTick timer.
- A 32-bit hardware multiplier. This can be the standard single-cycle multiplier, or a 32-cycle multiplier that has a lower area and performance implementation.
- Support for either little-endian or byte invariant big-endian data accesses.
- The ability to have deterministic, fixed-latency, interrupt handling.
- Load/store multiple and multicycle multiply instructions that can be abandoned and restarted to facilitate rapid interrupt handling.
- Optionally, Unprivileged/Privileged support for improved system integrity.
- C Application Binary Interface compliant exception model.

  This is the ARMv6-M, *C Application Binary Interface* (C-ABI) compliant exception model that enables the use of pure C functions as interrupt handlers.
- Low power sleep-mode entry using *Wait For Interrupt* (`WFI`), *Wait For Event* (`WFE`) instructions, or the return from interrupt sleep-on-exit feature.

NVIC that features:

- Up to 32 external interrupt inputs, each with four levels of priority.
- Dedicated *Non-Maskable Interrupt* (NMI) input.
- Support for both level-sensitive and pulse-sensitive interrupt lines.
- Optional *Wake-up Interrupt Controller* (WIC), providing ultra-low power sleep mode support.
- Optional relocation of the vector table.

Optional debug support:

- Zero to four hardware breakpoints.
- Zero to two watchpoints.
- *Program Counter Sampling Register* (PCSR) for non-intrusive code profiling, if at least one hardware data watchpoint is implemented.
- Single step and vector catch capabilities.
- Support for unlimited software breakpoints using `BKPT` instruction.
- Non-intrusive access to core peripherals and zero-waitstate system slaves through a compact bus matrix. A debugger can access these devices, including memory, even when the processor is running.
- Full access to core registers when the processor is halted.
- Optional, low gate-count CoreSight compliant debug access through a *Debug Access Port* (DAP) supporting either Serial Wire or JTAG debug connections.

Bus interfaces:

- Single 32-bit AMBA-3 AHB-Lite system interface that provides simple. integration to all system peripherals and memory.
- Optional single 32-bit single-cycle I/O port.
- Optional single 32-bit slave port that supports the DAP.

Optional *Memory Protection Unit* (MPU):

- Eight user configurable memory regions.
- Eight sub-region disables per region.
- Execute never (XN) support.
- Default memory map support.

2.2 Interfaces

This section describes the external interface functions.

This manual does not include pinout and signal naming because each device implementation can be different.

2.2.1 AHB-Lite interface

Transactions on the AHB-Lite interface are always marked as non-sequential.

Processor accesses and debug accesses share the external interface to external AHB peripherals. The processor accesses take priority over debug accesses.

Any vendor specific components can populate this bus.

Note

Instructions are only fetched using the AHB-Lite interface. To optimize performance, the Cortex-M0+ processor fetches ahead of the instruction it is executing. To minimize power consumption, the fetch ahead is limited to a maximum of 32-bits.

2.2.2 Single-cycle I/O port

The processor optionally implements a single-cycle I/O port that provides very high speed access to tightly-coupled peripherals, such as general-purpose-I/O (GPIO). The port is accessible both by loads and stores, from the processor and from the debugger. You cannot execute code from the I/O port.

2.2.3 Debug Access Port

The processor is implemented with either a low gate count *Debug Access Port* (DAP) or a full CoreSight DAP.

The low gate count *Debug Access Port* (DAP) provides a Serial Wire or JTAG debug-port, and connects to the processor slave port to provide full system-level debug access.

The full CoreSight DAP system enables the processor to provide full multiprocessor debug with simultaneous halt and release cross-triggering capabilities.

For more information on:

- DAP, see the ADI v5.1 version of the *ARM Debug Interface v5, Architecture Specification*
- CoreSight DAP, see the *CoreSight SoC Technical Reference Manual*.

2.2.4 Execution Trace Interface

The processor optionally implements an interface for the Micro Trace Buffer execution trace component. See the *CoreSight MTB-M0+ Technical Reference Manual* for more information.

Chapter 3
Programmers Model

This chapter provides an overview of the application-level programmers model. It contains the following sections:

3.1 About the programmers model

The *ARMv6-M Architecture Reference Manual* provides a complete description of the programmers model. This chapter gives an overview of the Cortex-M0+ programmers model that describes the implementation-defined options. It also contains the ARMv6-M Thumb instructions it uses and their cycle counts for the processor. In addition:

- Chapter 4 summarizes the system control features of the programmers model.
- Chapter 5 summarizes the NVIC features of the programmers model.
- Chapter 7 summarizes the Debug features of the programmers model.

3.2 Modes of operation and execution

See the *ARMv6-M Architecture Reference Manual* for information about the modes of operation and execution.

3.3 Instruction set summary

The processor implements the ARMv6-M Thumb instruction set, including a number of 32-bit instructions that use Thumb-2 technology. The ARMv6-M instruction set comprises:

- All of the 16-bit Thumb instructions from ARMv7-M excluding `CBZ`, `CBNZ` and `IT`.
- The 32-bit Thumb instructions `BL`, `DMB`, `DSB`, `ISB`, `MRS` and `MSR`.

Table 3-1 shows the Cortex-M0+ instructions and their cycle counts. The cycle counts are based on a system with zero wait-states.

Table 3-1 Cortex-M0+ instruction summary

Operation	Description	Assembler	Cycles
Move	8-bit immediate	`MOVS Rd, #<imm>`	1
	Lo to Lo	`MOVS Rd, Rm`	1
	Any to Any	`MOV Rd, Rm`	1
	Any to PC	`MOV PC, Rm`	2
Add	3-bit immediate	`ADDS Rd, Rn, #<imm>`	1
	All registers Lo	`ADDS Rd, Rn, Rm`	1
	Any to Any	`ADD Rd, Rd, Rm`	1
	Any to PC	`ADD PC, PC, Rm`	2
	8-bit immediate	`ADDS Rd, Rd, #<imm>`	1
	With carry	`ADCS Rd, Rd, Rm`	1
	Immediate to SP	`ADD SP, SP, #<imm>`	1
	Form address from SP	`ADD Rd, SP, #<imm>`	1
	Form address from PC	`ADR Rd, <label>`	1
Subtract	Lo and Lo	`SUBS Rd, Rn, Rm`	1
	3-bit immediate	`SUBS Rd, Rn, #<imm>`	1
	8-bit immediate	`SUBS Rd, Rd, #<imm>`	1
	With carry	`SBCS Rd, Rd, Rm`	1
	Immediate from SP	`SUB SP, SP, #<imm>`	1
	Negate	`RSBS Rd, Rn, #0`	1
Multiply	Multiply	`MULS Rd, Rm, Rd`	1 or 32[a]
Compare	Compare	`CMP Rn, Rm`	1
	Negative	`CMN Rn, Rm`	1
	Immediate	`CMP Rn, #<imm>`	1

Table 3-1 Cortex-M0+ instruction summary (continued)

Operation	Description	Assembler	Cycles
Logical	AND	`ANDS Rd, Rd, Rm`	1
	Exclusive OR	`EORS Rd, Rd, Rm`	1
	OR	`ORRS Rd, Rd, Rm`	1
	Bit clear	`BICS Rd, Rd, Rm`	1
	Move NOT	`MVNS Rd, Rm`	1
	AND test	`TST Rn, Rm`	1
Shift	Logical shift left by immediate	`LSLS Rd, Rm, #<shift>`	1
	Logical shift left by register	`LSLS Rd, Rd, Rs`	1
	Logical shift right by immediate	`LSRS Rd, Rm, #<shift>`	1
	Logical shift right by register	`LSRS Rd, Rd, Rs`	1
	Arithmetic shift right	`ASRS Rd, Rm, #<shift>`	1
	Arithmetic shift right by register	`ASRS Rd, Rd, Rs`	1
Rotate	Rotate right by register	`RORS Rd, Rd, Rs`	1
Load	Word, immediate offset	`LDR Rd, [Rn, #<imm>]`	2 or 1[b]
	Halfword, immediate offset	`LDRH Rd, [Rn, #<imm>]`	2 or 1[b]
	Byte, immediate offset	`LDRB Rd, [Rn, #<imm>]`	2 or 1[b]
	Word, register offset	`LDR Rd, [Rn, Rm]`	2 or 1[b]
	Halfword, register offset	`LDRH Rd, [Rn, Rm]`	2 or 1[b]
	Signed halfword, register offset	`LDRSH Rd, [Rn, Rm]`	2 or 1[b]
	Byte, register offset	`LDRB Rd, [Rn, Rm]`	2 or 1[b]
	Signed byte, register offset	`LDRSB Rd, [Rn, Rm]`	2 or 1[b]
	PC-relative	`LDR Rd, <label>`	2 or 1[b]
	SP-relative	`LDR Rd, [SP, #<imm>]`	2 or 1[b]
	Multiple, excluding base	`LDM Rn!, {<loreglist>}`	1+N[c]
	Multiple, including base	`LDM Rn, {<loreglist>}`	1+N[c]

Table 3-1 Cortex-M0+ instruction summary (continued)

Operation	Description	Assembler	Cycles
Store	Word, immediate offset	`STR Rd, [Rn, #<imm>]`	2 or 1[b]
	Halfword, immediate offset	`STRH Rd, [Rn, #<imm>]`	2 or 1[b]
	Byte, immediate offset	`STRB Rd, [Rn, #<imm>]`	2 or 1[b]
	Word, register offset	`STR Rd, [Rn, Rm]`	2 or 1[b]
	Halfword, register offset	`STRH Rd, [Rn, Rm]`	2 or 1[b]
	Byte, register offset	`STRB Rd, [Rn, Rm]`	2 or 1[b]
	SP-relative	`STR Rd, [SP, #<imm>]`	2 or 1[b]
	Multiple	`STM Rn!, {<loreglist>}`	1+N[c]
Push	Push	`PUSH {<loreglist>}`	1+N[c]
	Push with link register	`PUSH {<loreglist>, LR}`	1+N[d]
Pop	Pop	`POP {<loreglist>}`	1+N[c]
	Pop and return	`POP {<loreglist>, PC}`	3+N[d]
Branch	Conditional	`B<cc> <label>`	1 or 2[e]
	Unconditional	`B <label>`	2
	With link	`BL <label>`	3
	With exchange	`BX Rm`	2
	With link and exchange	`BLX Rm`	2
Extend	Signed halfword to word	`SXTH Rd, Rm`	1
	Signed byte to word	`SXTB Rd, Rm`	1
	Unsigned halfword	`UXTH Rd, Rm`	1
	Unsigned byte	`UXTB Rd, Rm`	1
Reverse	Bytes in word	`REV Rd, Rm`	1
	Bytes in both halfwords	`REV16 Rd, Rm`	1
	Signed bottom half word	`REVSH Rd, Rm`	1
State change	Supervisor Call	`SVC #<imm>`	-[f]
	Disable interrupts	`CPSID i`	1
	Enable interrupts	`CPSIE i`	1
	Read special register	`MRS Rd, <specreg>`	3
	Write special register	`MSR <specreg>, Rn`	3
	Breakpoint	`BKPT #<imm>`	-[f]

Table 3-1 Cortex-M0+ instruction summary (continued)

Operation	Description	Assembler	Cycles
Hint	Send event	SEV	1
	Wait for event	WFE	2[g]
	Wait for interrupt	WFI	2[g]
	Yield	YIELD	1[h]
	No operation	NOP	1
Barriers	Instruction synchronization	ISB	3
	Data memory	DMB	3
	Data synchronization	DSB	3

a. Depends on multiplier implementation.
b. 2 if to AHB interface or SCS, 1 if to single-cycle I/O port.
c. N is the number of elements in the list.
d. N is the number of elements in the list including PC or LR.
e. 2 if taken, 1 if not-taken.
f. Cycle count depends on processor and debug configuration.
g. Excludes time spent waiting for an interrupt or event.
h. Executes as NOP.

See the *ARMv6-M Architecture Reference Manual* for more information about the ARMv6-M Thumb instructions.

3.3.1 Binary compatibility with other Cortex processors

The processor implements a binary compatible subset of the instruction set and features provided by other Cortex-M profile processors. You can move software, including system level software, from the Cortex-M0+ 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 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.
- If you use an ARMv7-M processor, 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 Memory model

The processor contains a bus matrix that arbitrates the processor core and optional *Debug Access Port* (DAP) memory accesses to both the external memory system and to the internal NVIC and debug components.

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, NVIC, and debug resources are completely non-intrusive for typical code execution.

The system memory map is ARMv6-M architecture compliant, and is common both to the debugger and processor accesses. Transactions are routed as follows:

- All accesses below `0xE0000000` or above `0xEFFFFFFF` appear as AHB-Lite transactions on the AHB-Lite master port of the processor.
- Accesses in the range `0xE0000000` to `0xEFFFFFFF` are handled within the processor and do not appear on the AHB-Lite master port of the processor.
- Data accesses to the AHB-Lite interface from both the debugger and the processor can be hardware configured to appear instead on the single-cycle I/O port.

The processor supports only word size accesses in the range `0xE0000000` - `0xEFFFFFFF`.

Table 3-2 shows the code, data, and device suitability for each region of the default memory map. This is the memory map used by implementations without the optional *Memory Protection Unit* (MPU), or when an included MPU is disabled. The attributes and permissions of all regions, except that targeting the Cortex-M0+ NVIC and debug components, can be modified using an implemented MPU.

Table 3-2 Default memory map usage

Address range	Code	Data	Device
`0xF0000000` - `0xFFFFFFFF`	No	No	Yes
`0xE0000000` - `0xEFFFFFFF`	No	No	No[a]
`0xA0000000` - `0xDFFFFFFF`	No	No	Yes
`0x60000000` - `0x9FFFFFFF`	Yes	Yes	No
`0x40000000` - `0x5FFFFFFF`	No	No	Yes
`0x20000000` - `0x3FFFFFFF`	Yes[b]	Yes	No
`0x00000000` - `0x1FFFFFFF`	Yes	Yes	No

a. Space reserved for Cortex-M0+ NVIC and debug components.

b. Cortex-M1 devices implementing data *Tightly-Coupled Memories* (TCMs) in this region do not support code execution from the data TCM.

Note

Regions not marked as suitable for code behave as *eXecute-Never* (XN) and generate a HardFault exception if code attempts to execute from this location.

See the *ARMv6-M Architecture Reference Manual* for more information about the memory model.

3.5 Processor core registers summary

Table 3-3 shows the processor core register set summary. Each of these registers is 32 bits wide.

Table 3-3 Processor core register set summary

Name	Description
R0-R12	R0-R12 are general-purpose registers for data operations.
MSP (R13) PSP (R13)	The *Stack Pointer* (SP) is register R13. In Thread mode, the CONTROL register indicates the stack pointer to use, *Main Stack Pointer* (MSP) or *Process Stack Pointer* (PSP).
LR (R14)	The *Link Register* (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	The *Program Counter* (PC) is register R15. It contains the current program address.
PSR	The *Program Status Register* (PSR) combines: • *Application Program Status Register* (APSR). • *Interrupt Program Status Register* (IPSR). • *Execution Program Status Register* (EPSR). These registers provide different views of the PSR.
PRIMASK	The PRIMASK register prevents activation of all exceptions with configurable priority. For information about the exception model the processor supports, see *Exceptions* on page 3-10.
CONTROL	The CONTROL register controls the stack used, and optionally the code privilege level, when the processor is in Thread mode.

Note

See the *ARMv6-M Architecture Reference Manual* for information about the processor core registers and their addresses, access types, and reset values.

3.6 Exceptions

This section describes the exception model of the processor.

3.6.1 Exception handling

The processor implements advanced exception and interrupt handling, as described in the *ARMv6-M Architecture Reference Manual*.

To minimize interrupt latency, the processor abandons any load-multiple or store-multiple instruction to take any pending interrupt. On return from the interrupt handler, the processor restarts the load-multiple or store-multiple instruction from the beginning.

Note

A processor that implements the 32-cycle multiplier abandons multiply instructions in the same way.

This means that software must not use load-multiple or store-multiple instructions when a device is accessed in 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.

The processor implementation can ensure that a fixed number of cycles are required for the NVIC to detect an interrupt signal and the processor fetch the first instruction of the associated interrupt handler. If this is done, the highest priority interrupt is jitter-free. See the documentation supplied by the processor implementer for more information.

To reduce interrupt latency and jitter, the Cortex-M0+ processor implements both interrupt late-arrival and interrupt tail-chaining mechanisms, as defined by the ARMv6-M architecture. The worst case interrupt latency, for the highest priority active interrupt in a zero wait-state system not using jitter suppression, is 15 cycles.

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

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

Chapter 4
System Control

This chapter summarizes the system control registers and their structure. It contains the following sections:

4.1 About system control

This section describes the system control registers that control and configure various system control functions.

4.2 System control register summary

Table 4-1 gives the system control registers. Each of these registers is 32 bits wide.

Table 4-1 System control registers

Name	Description
SYST_CSR	*SysTick Control and Status Register* in the *ARMv6-M Architecture Reference Manual.*
SYST_RVR	*SysTick Reload Value Register* in the *ARMv6-M Architecture Reference Manual*.
SYST_CVR	*SysTick Current Value Register* in the *ARMv6-M Architecture Reference Manual*.
SYST_CALIB[a]	*SysTick Calibration value Register* in the *ARMv6-M Architecture Reference Manual*.
CPUID	See *CPUID Register*.
ICSR	*Interrupt Control State Register* in the *ARMv6-M Architecture Reference Manual*.
AIRCR[a]	*Application Interrupt and Reset Control Register* in the *ARMv6-M Architecture Reference Manual.*
CCR	*Configuration and Control Register* in the *ARMv6-M Architecture Reference Manual.*
SHPR2	*System Handler Priority Register 2* in the *ARMv6-M Architecture Reference Manual*.
SHPR3	*System Handler Priority Register 3* in the *ARMv6-M Architecture Reference Manual*.
SHCSR	*System Handler Control and State Register* in the *ARMv6-M Architecture Reference Manual.*
VTOR[b]	*Vector table Offset Register* in the *ARMv6-M Architecture Reference Manual*.
ACTLR[c]	*Auxiliary Control Register* in the *ARMv6-M Architecture Reference Manual*.

a. This value is configured by the implementer during implementation. See the documentation supplied by your implementer for more information.

b. If implemented, the VTOR enables bits[31:8] of the vector table address to be specified.

c. Implemented as RAZ/WI

Note

- All system control registers are only accessible using word transfers. Any attempt to read or write a halfword or byte is Unpredictable.
- If the processor is implemented without the SysTick timer, the SYST_CSR, SYST_RVR, SYST_CVR, and SYST_CALIB registers are RAZ/WI.
- See the *ARMv6-M Architecture Reference Manual* for more information about the system control registers, and their addresses and access types, and reset values not shown in Table 4-1.

4.2.1 CPUID Register

The **CPUID** characteristics are:

Purpose Contains the part number, version, and implementation information that is specific to this processor.

Usage constraints There are no usage constraints.

Attributes See Table 4-1.

Figure 4-1 shows the CPUID bit register assignments.

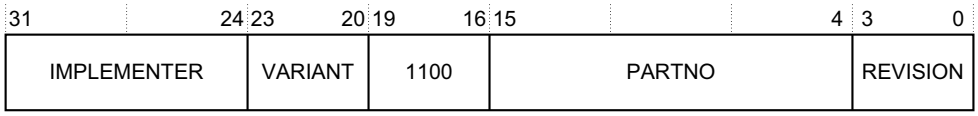

Figure 4-1 CPUID bit register assignments

Table 4-2 shows the **CPUID** register bit assignments.

Table 4-2 CPUID bit register assignments

Bits	Field	Function
[31:24]	IMPLEMENTER	Implementer code: `0x41` ARM.
[23:20]	VARIANT	Major revision number *n* in the r*n*p*m* revision status. See *Product revision status* on page vi: `0x0`.
[19:16]	ARCHITECTURE	Indicates the architecture, ARMv6-M: `0xC`.
[15:4]	PARTNO	Indicates part number, Cortex-M0+: `0xC60`.
[3:0]	REVISION	Minor revision number *m* in the r*n*p*m* revision status. See *Product revision status* on page vi. `0x1`.

Chapter 5
Nested Vectored Interrupt Controller

This chapter summarizes the *Nested Vectored Interrupt Controller* (NVIC). It contains the following sections:

5.1 About the NVIC

External interrupt signals connect to the NVIC, and the NVIC prioritizes the interrupts. Software can set the priority of each interrupt. The NVIC and the Cortex-M0+ processor core are closely coupled, providing low latency interrupt processing and efficient processing of late arriving interrupts.

All NVIC registers are only accessible using word transfers. Any attempt to read or write a halfword or byte individually is Unpredictable.

NVIC registers are always little-endian. Processor accesses are correctly handled regardless of the endian configuration of the processor.

Processor exception handling is described in *Exceptions* on page 3-10.

5.1.1 SysTick timer option

The implementation can include a 24-bit SysTick system timer, that extends the functionality of both the processor and the NVIC.

When present, the NVIC part of the extension provides:

- A 24-bit system timer (SysTick).
- Additional configurable priority SysTick interrupt.

See the *ARMv6-M Architecture Reference Manual* for more information.

5.1.2 Low power modes

The implementation can include a 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 *ARMv6-M Architecture Reference Manual* for more information.

5.2 NVIC register summary

Table 5-1 shows the NVIC registers. Each of these registers is 32 bits wide.

Table 5-1 NVIC registers

Name	Description
NVIC_ISER	*Interrupt Set-Enable Register*.
NVIC_ICER	*Interrupt Clear-Enable Register*.
NVIC_ISPR	*Interrupt Set-Pending Register*.
NVIC_ICPR	*Interrupt Clear-Pending Register.*
NVIC_IPR0-NVIC_IPR7	*Interrupt Priority Registers*.

Note

See the *ARMv6-M Architecture Reference Manual* for more information about the NVIC registers and their addresses, access types, and reset values.

Chapter 6
Memory Protection Unit

This chapter describes the processor *Memory Protection Unit* (MPU). It contains the following sections:

6.1 About the MPU

The MPU is an optional component for memory protection. When implemented, the processor supports the ARMv6 *Protected Memory System Architecture* model. The MPU provides full support for:

- Eight unified 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 HardFault handler. See the *ARMv6-M Architecture Reference Manual* for more information.

You can use the MPU to:

- Enforce privilege rules.
- Separate processes.
- Manage memory attributes.

6.2 MPU register summary

Table 6-1 shows the MPU registers. Each of these registers is 32 bits wide. If the MPU is not present in the implementation, then all of these registers read as zero.

Table 6-1 MPU registers

Name	Description
MPU_TYPE	MPU Type Register.
MPU_CTRL	MPU Control Register.
MPU_RNR	MPU Region Number Register.
MPU_RBAR	MPU Region Base Address Register.
MPU_RASR	MPU Region Attribute and Size Register.

Note

- See the *ARMv6-M Architecture Reference Manual* for more information about the MPU registers and their addresses, access types, and reset values.
- The MPU supports region sizes from 256-bytes to 4Gb, with 8-sub regions per region.

Chapter 7
Debug

This chapter summarizes the debug system. It contains the following sections:

7.1 About debug

The processor implementation determines the debug configuration, including whether debug is implemented. If debug is not implemented, no ROM table is present and the halt, breakpoint, and watchpoint functionality is not present.

Basic debug functionality includes processor halt, single-step, processor core register access, Reset and HardFault Vector Catch, unlimited software breakpoints, and full system memory access. See the *ARMv6-M Architecture Reference Manual*.

The debug option might include either or both:

- A breakpoint unit supporting 1, 2, 3, or 4 hardware breakpoints.
- A watchpoint unit supporting 1 or 2 watchpoints.

The processor implementation can be partitioned to place the debug components in a separate power domain from the processor core and NVIC.

When debug is implemented, ARM recommends that a debugger identifies and connects to the debug components using the CoreSight debug infrastructure.

To discover the components in the CoreSight debug infrastructure, ARM recommends that a debugger follows the flow shown in Figure 7-1. In this example, a debugger reads the peripheral and component ID registers for each CoreSight component in the CoreSight system.

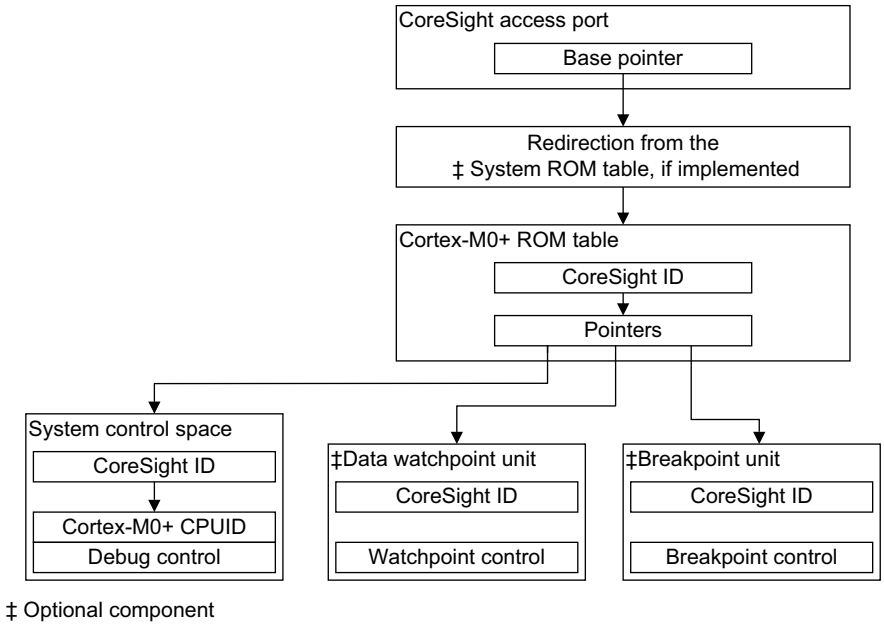

Figure 7-1 CoreSight discovery

To identify the Cortex-M0+ processor within the CoreSight system, ARM recommends that a debugger:

1. Locates and identifies the Cortex-M0+ ROM table using its CoreSight identification. See *Cortex-M0+ ROM table identification values* on page 7-3.
2. Follows the pointers in that Cortex-M0+ ROM table:
   a. *System Control Space* (SCS).
   b. *Breakpoint unit* (BPU).
   c. *Data watchpoint unit* (DWT).

   See *Cortex-M0+ ROM table components* on page 7-3.

When a debugger identifies the SCS from its CoreSight identification, it can identify the processor and its revision number from the CPUID register offset at `0xD00` in the SCS, `0xE000ED00`.

A debugger cannot rely on the Cortex-M0+ ROM table being the first ROM table encountered. One or more system ROM tables are required between the access port and the Cortex-M0+ ROM table if other CoreSight components are in the system, or if the implementation is to be uniquely identifiable.

7.1.1 Cortex-M0+ ROM table identification and entries

Table 7-1 shows the ROM table identification registers and values for debugger detection. This enables debuggers to identify the processor and its debug capabilities.

Note

The Cortex-M0+ ROM table only supports word size transactions.

Table 7-1 Cortex-M0+ ROM table identification values

Register	Value	Description
Peripheral ID4	`0x00000004`	*Component and peripheral ID register formats* in the *ARMv6-M Architecture Reference Manual.*
Peripheral ID0	`0x000000C0`	
Peripheral ID1	`0x000000B4`	
Peripheral ID2	`0x0000000B`	
Peripheral ID3	`0x00000000`	
Component ID0	`0x0000000D`	
Component ID1	`0x00000010`	
Component ID2	`0x00000005`	
Component ID3	`0x000000B1`	

Table 7-2 shows the CoreSight components that the Cortex-M0+ ROM table points to. The values depend on the implemented debug configuration.

Table 7-2 Cortex-M0+ ROM table components

Component	Value	Description
SCS	`0xFFF0F003`	See *System Control Space* on page 7-4.
DWT	`0xFFF02003`[a]	See *Data watchpoint unit* on page 7-4.
BPU	`0xFFF03003`[b]	See *Breakpoint unit* on page 7-5.
End marker	`0x00000000`	See *DAP accessible ROM table* in the *ARMv6-M Architecture Reference Manual.*
MemType	`0x00000001`	See *CoreSight management registers* in the *ARMv6-M Architecture Reference Manual.*

a. Reads as `0xFFF02002` if no watchpoints are implemented.

b. Reads as `0xFFF03002` if no breakpoints are implemented.

The SCS, DWT, and BPU ROM table entries point to the debug components at addresses `0xE000E000`, `0xE0001000` and `0xE0002000` respectively. The value for each entry is the offset of that component from the ROM table base address, `0xE00FF000`.

See the *ARMv6-M Architecture Reference Manual* and the *CoreSight SoC Technical Reference Manual* for more information about the ROM table ID and component registers, and their addresses and access types.

7.1.2 System Control Space

If debug is implemented, the processor provides debug through registers in the SCS, see *Debug register summary* on page 7-7.

SCS CoreSight identification

Table 7-3 shows the SCS CoreSight identification registers and values for debugger detection. Final debugger identification of the Cortex-M0+ processor is through the CPUID register in the SCS, see *CPUID Register* on page 4-3.

Table 7-3 SCS identification values

Register	Value	Description
Peripheral ID4	`0x00000004`	*Component and Peripheral ID register formats* in the *ARMv6-M Architecture Reference Manual.*
Peripheral ID0	`0x00000008`	
Peripheral ID1	`0x000000B0`	
Peripheral ID2	`0x0000000B`	
Peripheral ID3	`0x00000000`	
Component ID0	`0x0000000D`	
Component ID1	`0x000000E0`	
Component ID2	`0x00000005`	
Component ID3	`0x000000B1`	

See the *ARMv6-M Architecture Reference Manual* and the *CoreSight SoC Technical Reference Manual* for more information about the SCS CoreSight identification registers, and their addresses and access types.

7.1.3 Data watchpoint unit

The Cortex-M0+ DWT implementation provides zero, one or two watchpoint register sets. A processor configured with zero watchpoint implements no watchpoint functionality and the ROM table shows that no DWT is implemented.

DWT functionality

The processor watchpoints implement both data address and PC based watchpoint functionality, a PC sampling register, and support comparator address masking, as described in the *ARMv6-M Architecture Reference Manual*.

DWT CoreSight identification

Table 7-4 shows the DWT identification registers and values for debugger detection.

Table 7-4 DWT identification values

Register	Value	Description
Peripheral ID4	0x00000004	*Component and Peripheral ID register formats* in the *ARMv6-M Architecture Reference Manual.*
Peripheral ID0	0x0000000A	
Peripheral ID1	0x000000B0	
Peripheral ID2	0x0000000B	
Peripheral ID3	0x00000000	
Component ID0	0x0000000D	
Component ID1	0x000000E0	
Component ID2	0x00000005	
Component ID3	0x000000B1	

See the *ARMv6-M Architecture Reference Manual* and the *CoreSight SoC Technical Reference Manual* for more information about the DWT CoreSight identification registers, and their addresses and access types.

DWT Program Counter Sample Register

A processor that implements the data watchpoint unit also implements the ARMv6-M optional *DWT Program Counter Sample Register* (DWT_PCSR). This register enables a debugger to periodically sample the PC without halting the processor. This provides coarse grained profiling. See the *ARMv6-M Architecture Reference Manual* for more information.

The Cortex-M0+ DWT_PCSR records both instructions that pass their condition codes and those that fail.

7.1.4 Breakpoint unit

The Cortex-M0+ BPU implementation provides between zero and four breakpoint registers. A processor configured with zero breakpoints implements no breakpoint functionality and the ROM table shows that no BPU is implemented.

BPU functionality

The processor breakpoints implement PC based breakpoint functionality, as described in the *ARMv6-M Architecture Reference Manual*.

BPU CoreSight identification

Table 7-5 shows the BPU identification registers and their values for debugger detection.

Table 7-5 BPU identification registers

Register	Value	Description
Peripheral ID4	0x00000004	*Component and Peripheral ID register formats* in the *ARMv6-M Architecture Reference Manual.*
Peripheral ID0	0x0000000B	
Peripheral ID1	0x000000B0	
Peripheral ID2	0x0000000B	
Peripheral ID3	0x00000000	
Component ID0	0x0000000D	
Component ID1	0x000000E0	
Component ID2	0x00000005	
Component ID3	0x000000B1	

See the *ARMv6-M Architecture Reference Manual* and the *CoreSight SoC Technical Reference Manual* for more information about the BPU CoreSight identification registers, and their addresses and access types.

7.2 Debug register summary

Table 7-6 shows the debug registers. Each of these registers is 32 bits wide.

Table 7-6 Debug registers summary

Name	Description
DFSR	*Debug Fault Status Register* in the *ARMv6-M Architecture Reference Manual*.
DHCSR	*Debug Halting Control* and Status Register in the *ARMv6-M Architecture Reference Manual.*
DCRSR	*Debug Core Register Selector Register* in the *ARMv6-M Architecture Reference Manual*.
DCRDR	*Debug Core Register Data Register* in the *ARMv6-M Architecture Reference Manual.*
DEMCR	*Debug Exception and Monitor Control Register* in the *ARMv6-M Architecture Reference Manual*.

Table 7-7 shows the BPU registers. Each of these registers is 32 bits wide.

Table 7-7 BPU register summary

Name	Description
BP_CTRL	*Breakpoint Control Register* in the *ARMv6-M Architecture Reference Manual.*
BP_COMP0	*Breakpoint Comparator Registers* in the *ARMv6-M Architecture Reference Manual*.
BP_COMP1	
BP_COMP2	
BP_COMP3	

Table 7-8 shows the DWT registers. Each of these registers is 32 bits wide.

Table 7-8 DWT register summary

Name	Description
DWT_CTRL	*Control Register* in the *ARMv6-M Architecture Reference Manual*.
DWT_PCSR	*Program Counter Sample Register* in the *ARMv6-M Architecture Reference Manual*.
DWT_COMP0	*Comparator Register* in the *ARMv6-M Architecture Reference Manual.*
DWT_MASK0[a]	*Mask Register* in the *ARMv6-M Architecture Reference Manual.*
DWT_FUNCTION0	*Function Register* in the *ARMv6-M Architecture Reference Manual.*
DWT_COMP1	*Comparator Register* in the *ARMv6-M Architecture Reference Manual.*
DWT_MASK1[a]	*Mask Register* in the *ARMv6-M Architecture Reference Manual.*
DWT_FUNCTION1	*Function Register* in the *ARMv6-M Architecture Reference Manual.*

a. Supports masking up to 2GB.

See the *ARMv6-M Architecture Reference Manual* for more information about the debug registers and their addresses, access types, and reset values.

Note

Software cannot access the debug registers.

Appendix A
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
Updated CPUID for r0p1	*CPUID Register* on page 4-3	r0p1
Corrected memory map addresses	*Default memory map usage* on page 3-8.	All