Cortex™-M0

Revision: r0p0

Technical Reference Manual

ARM®

Cortex-M0
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
19 March 2009	A	Non-Confidential, Restricted Access	First release for r0p0
27 July 2009	B	Non-Confidential, Restricted Access	Second release for r0p0
30 November 2009	C	Non-Confidential	Third release for r0p0

Proprietary Notice

Confidentiality Status

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

List of Tables
Cortex-M0 Technical Reference Manual

List of Figures
Cortex-M0 Technical Reference Manual

Preface

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

- *About this book* on page xii
- *Feedback* on page xv.

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 manual, 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 to help:

* 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 and its components.

Chapter 3 *Programmers Model*

Read this chapter for an overview of the processor register set, modes of operation, and other information for programming the processor.

Chapter 4 *System Control*

Read this chapter for a summary of the system control registers and programmers model.

Chapter 5 *Nested Vectored Interrupt Controller*

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

Chapter 6 *Debug*

Read this chapter for a summary of the debug registers and programmers model.

Appendix A *Revisions*

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

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

Conventions

Conventions that this manual can use are described in:

• *Typographical*.

Typographical

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<u>mono</u>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.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

This book contains information that is specific to the processor. 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 CoreSight™ Components Technical Reference Manual* (ARM DDI 0314)
- *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)*
  (IHI0036)
- *Cortex-M0 Integration and Implementation Manual* (ARM DII 0238)
- *Cortex-M0 User Guide Reference Material* (ARM DUI 0467A).

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 the processor and its documentation.

Feedback on the processor

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

* The product name.

* The product revision or version.

* An explanation with as much information as you can provide. Include symptoms if appropriate.

Feedback on this manual

If you have any comments on this manual, send an email to errata@arm.com. Give:
* the title
* the number, ARM DDI 0432C
* the page numbers to which your comments apply
* a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

xv

Chapter 1
Introduction

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

- *About the processor* on page 1-2
- *Features* on page 1-3
- *Interfaces* on page 1-4
- *Configurable options* on page 1-5
- *Product documentation, design flow and architecture* on page 1-6
- *Product revisions* on page 1-9.

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 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
- •   power control optimization of system components
- •   integrated sleep modes for low power consumption
- •   fast code execution permits slower processor clock or increases sleep mode time
- •   hardware multiplier
- •   deterministic, high-performance interrupt handling for time-critical applications
- •   Serial Wire Debug reduces the number of pins required for debugging.

For information about Cortex-M0 architectural compliance, see the *Architecture and protocol information* on page 1-8.

1.3   Interfaces

The interfaces included in the processor for external access include:
- external AHB-Lite interface
- *Debug Access Port* (DAP).

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 1, 2, 4, 8, 16, 24 or 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

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 yields the least-significant 32-bits. 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 xiii 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 hard macrocell. This might include integrating RAMs into the design.

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*
- *Debug Access Port architecture*.

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 ARM*.

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.

Chapter 2
Functional Description

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

- *About the functions* on page 2-2
- *Interfaces* on page 2-4.

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 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.
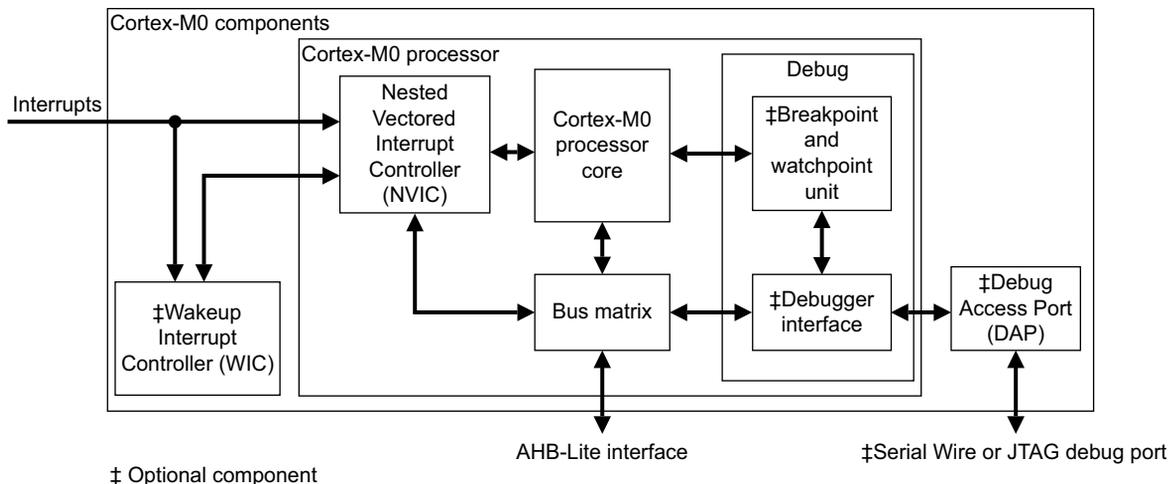
Figure 2-1 Functional block diagram

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.
  - The system interface supports either little-endian or byte invariant big-endian data accesses.
  - The ability to have deterministic, fixed-latency, interrupt handling.
  - Load/store-multiples and multicycle-multiplies that can be abandoned and restarted to facilitate rapid interrupt handling.
  - 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:
  — 1, 2, 4, 8, 16, 24, or 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 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

  — single 32-bit slave port that supports the DAP.

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. See the *Cortex-M0 Integration Manual* or your implementers documentation for more information.

2.2.1 AHB-Lite interface

Transactions on the AHB-Lite interface are always marked as non-sequential. For information about the supported transactions, see the *Cortex-M0 Integration Manual*.

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.

2.2.2 Debug Access Port

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

You can configure the processor slave port to connect to a full CoreSight DAP system, with the processor providing full multiprocessor debug 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 *ARM CoreSight Components Technical Reference Manual*.

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 ARM* 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 6 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.

——— **Note** ———

Other ARM architectures support the concept of privileged or unprivileged software execution. This processor does not support different privilege levels. Software execution is always privileged, meaning software can access all the features of the processor.

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	3
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	3
	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

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

Operation	Description	Assembler	Cycles
Subtract	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
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
	Halfword, immediate offset	`LDRH Rd, [Rn, #<imm>]`	2
	Byte, immediate offset	`LDRB Rd, [Rn, #<imm>]`	2
	Word, register offset	`LDR Rd, [Rn, Rm]`	2
	Halfword, register offset	`LDRH Rd, [Rn, Rm]`	2
	Signed halfword, register offset	`LDRSH Rd, [Rn, Rm]`	2
	Byte, register offset	`LDRB Rd, [Rn, Rm]`	2

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

Operation	Description	Assembler	Cycles
Load	Signed byte, register offset	LDRSB Rd, [Rn, Rm]	2
	PC-relative	LDR Rd, <label>	2
	SP-relative	LDR Rd, [SP, #<imm>]	2
	Multiple, excluding base	LDM Rn!, {<loreglist>}	$1+N^b$
	Multiple, including base	LDM Rn, {<loreglist>}	$1+N^b$
Store	Word, immediate offset	STR Rd, [Rn, #<imm>]	2
	Halfword, immediate offset	STRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	STRB Rd, [Rn, #<imm>]	2
	Word, register offset	STR Rd, [Rn, Rm]	2
	Halfword, register offset	STRH Rd, [Rn, Rm]	2
	Byte, register offset	STRB Rd, [Rn, Rm]	2
	SP-relative	STR Rd, [SP, #<imm>]	2
	Multiple	STM Rn!, {<loreglist>}	$1+N^b$
Push	Push	PUSH {<loreglist>}	$1+N^b$
	Push with link register	PUSH {<loreglist>, LR}	$1+N^b$
Pop	Pop	POP {<loreglist>}	$1+N^b$
	Pop and return	POP {<loreglist>, PC}	$4+N^c$
Branch	Conditional	B<cc> <label>	1 or 3^d
	Unconditional	B <label>	3
	With link	BL <label>	4
	With exchange	BX Rm	3
	With link and exchange	BLX Rm	3
Extend	Signed halfword to word	SXTH Rd, Rm	1
	Signed byte to word	SXTB Rd, Rm	1
	Unsigned halfword	UXTH Rd, Rm	1

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

Operation	Description	Assembler	Cycles
Extend	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>`	- [e]
	Disable interrupts	`CPSID i`	1
	Enable interrupts	`CPSIE i`	1
	Read special register	`MRS Rd, <specreg>`	4
	Write special register	`MSR <specreg>, Rn`	4
	Breakpoint	`BKPT #<imm>`	- [e]
Hint	Send event	`SEV`	1
	Wait for event	`WFE`	2 [f]
	Wait for interrupt	`WFI`	2 [f]
	Yield	`YIELD`[g]	1
	No operation	`NOP`	1
Barriers	Instruction synchronization	`ISB`	4
	Data memory	`DMB`	4
	Data synchronization	`DSB`	4

a. Depends on multiplier implementation.
b. N is the number of elements.
c. N is the number of elements in the stack-pop list including PC and assumes load or store does not generate a HardFault exception.
d. 3 if taken, 1 if not-taken.
e. Cycle count depends on core and debug configuration.
f. Excludes time spent waiting for an interrupt or event.
g. Executes as NOP.

See the *ARMv6-M ARM* 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 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-waitstate system, all debug accesses to system memory, NVIC, and debug resources are completely non-intrusive.

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

- All accesses below 0xE0000000 or above 0xF0000000 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.

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 memory map.

Table 3-2 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 ARM* 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)	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).
PSP (R13)	
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 3-12.
CONTROL	The CONTROL register controls the stack used when the processor is in Thread mode.

——— **Note** ———

See the *ARMv6-M ARM* 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 ARM*.

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 multiplies in the same way.

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

* *About system control* on page 4-2
* *System control register summary* on page 4-3.

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	Reset value	Description
SYST_CSR	-	*SysTick Control and Status Register* in the *ARMv6-M ARM*
SYST_RVR	Unknown	*SysTick Reload Value Register* in the *ARMv6-M ARM*
SYST_CVR	Unknown	*SysTick Current Value Register* in the *ARMv6-M ARM*
SYST_CALIB	Implementation-defined[a]	*SysTick Calibration value Register* in the *ARMv6-M ARM*
CPUID	0x410CC200	See *CPUID Register* on page 4-4
ICSR	-	*Interrupt Control State Register* in the *ARMv6-M ARM*
AIRCR	0x00000000[b] 0x00008000[c]	*Application Interrupt and Reset Control Register* in the *ARMv6-M ARM*
CCR	-	*Configuration and Control Register* in the *ARMv6-M ARM*
SHPR2	-	*System Handler Priority Register 2* in the *ARMv6-M ARM*
SHPR3	-	*System Handler Priority Register 3* in the *ARMv6-M ARM*
SHCSR	-	*System Handler Control and State Register* in the *ARMv6-M ARM*

a. This value is configured by the implementer during implementation. See the documentation supplied by your implementer for more information.
b. Little-endian implementation.
c. Big-endian implementation.

——— **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 register reads as zero, writes ignored RAZ/WI.

- See the *ARMv6-M ARM* 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 the *System control registers* on page 4-3.

Figure 4-1 shows the CPUID bit register assignments.

31	24	23	20	19	16	15	4	3	0
Implementer		Variant		Constant		Partno		Revision	
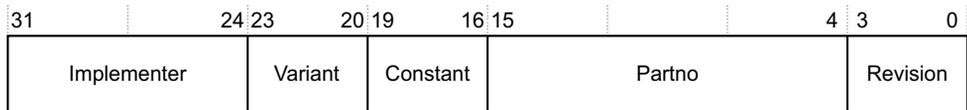
Figure 4-1 CPUID bit register assignments

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	Implementation defined. In ARM implementations this is the major revision number *n* in the r*n* part of the r*n*p*n* revision status, *Product revision status* on page xii: 0x0.
[19:16]	Constant	Indicates the architecture, ARMv6-M: 0xC.
[15:4]	Partno	Indicates part number, Cortex-M0: 0xC20.
[3:0]	Revision	Indicates revision. In ARM implementations this is the minor revision number *n* in the p*n* part of the r*n*p*n* revision status, see *Product revision status* on page xii. For example, for release r0p0: 0x0.

Chapter 5
Nested Vectored Interrupt Controller

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

- *About the NVIC* on page 5-2
- *NVIC register summary* on page 5-3.

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 3-12.

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 ARM* 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, which causes the processor core to enter sleep mode when it returns from an exception handler to Thread mode. See the *ARMv6-M ARM* 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
ISER	*Interrupt Set-Enable Register* in the *ARMv6-M ARM*
ICER	*Interrupt Clear-Enable Register* in the *ARMv6-M ARM*
ISPR	*Interrupt Set-Pending Register* in the *ARMv6-M ARM*
ICPR	*Interrupt Clear-Pending Register* in the *ARMv6-M ARM*
IPR0-IPR7	*Interrupt Priority Registers* in the *ARMv6-M ARM*

——— **Note** ———

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

Chapter 6
Debug

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

- *About debug* on page 6-2
- *Debug register summary* on page 6-9.

6.1   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 ARM* for more information.

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 recommend that a debugger identifies and connects to the debug components using the CoreSight debug infrastructure.

Figure 6-1 on page 6-3 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.

CoreSight debug port

CoreSight access port
Base pointer

Redirection from the
‡ System ROM table, if implemented

Cortex-M0 ROM table
CoreSight ID
Pointers

System control space
CoreSight ID
Cortex-M0 CPUID
Debug control

‡Data watchpoint unit
CoreSight ID
Watchpoint control

‡Breakpoint unit
CoreSight ID
Breakpoint control

‡ Optional component
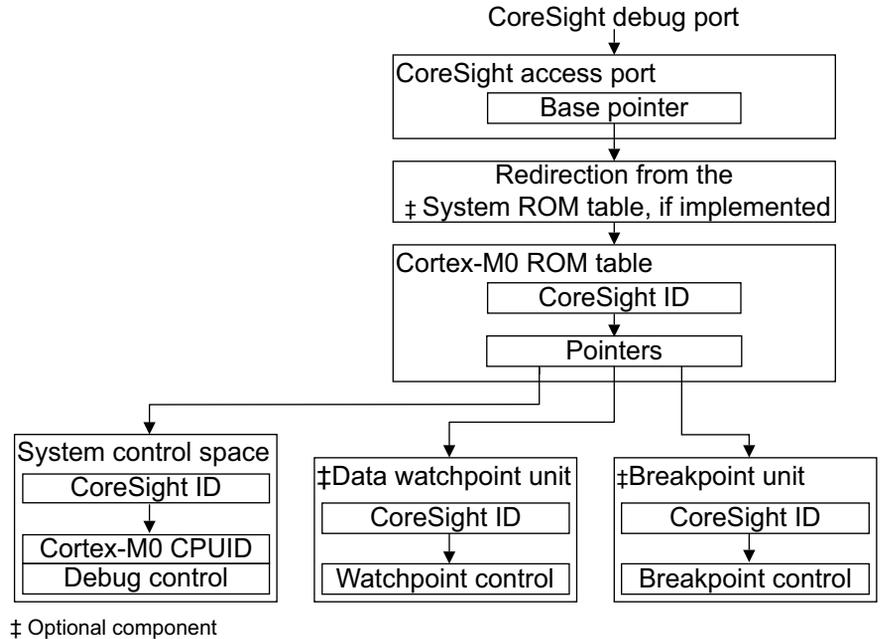
Figure 6-1 CoreSight discovery

To identify the Cortex-M0 processor within the CoreSight system, ARM Limited recommend 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 6-4 for more information.

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 6-4 for more information.

    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 uniquely identifiable.

6.1.1 Cortex-M0 ROM table identification and entries

Table 6-1 shows the ROM table identification registers and values for debugger detection. This permits debuggers to identify the processor and its debug capabilities.

——— **Note** ———
The Cortex-M0 ROM table only supports word size transactions.

Table 6-1 Cortex-M0 ROM table identification values

Register	Value	Description
Peripheral ID4	0x00000004	*Component and peripheral ID register formats* in the *ARMv6-M ARM*
Peripheral ID0	0x00000071	
Peripheral ID1	0x000000B4	
Peripheral ID2	0x0000000B	
Peripheral ID3	0x00000000	
Component ID0	0x0000000D	
Component ID1	0x00000010	
Component ID2	0x00000005	
Component ID3	0x000000B1	

Table 6-2 shows the CoreSight components that the Cortex-M0 ROM table points to. The values depend on the implemented debug configuration.

Table 6-2 Cortex-M0 ROM table components

Component	Value	Description
SCS	0xFFF0F003	See *System Control Space* on page 6-5
DWT	0xFFF02003[a]	See *Data watchpoint unit* on page 6-6
BPU	0xFFF03003[b]	See *Breakpoint unit* on page 6-7
End marker	0x00000000	See *DAP accessible ROM table* in the *ARMv6-M ARM*
MemType	0x00000001	See *CoreSight management registers* in the *ARMv6-M ARM*

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 ARM* 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.

6.1.2   System Control Space

If debug is implemented, the processor provides debug through registers in the SCS, see *Debug register summary* on page 6-9.

SCS CoreSight identification

Table 6-3 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 4-4.

Table 6-3 SCS identification values

Register	Value	Description
Peripheral ID4	0x00000004	*Component and Peripheral ID register formats* in the *ARMv6-M ARM*
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 ARM* and the *ARM CoreSight Components Technical Reference Manual* for more information about the SCS CoreSight identification registers, and their addresses and access types.

6.1.3 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 ARM*.

DWT CoreSight identification

Table 6-4 shows the DWT identification registers and values for debugger detection.

Table 6-4 DWT identification values

Register	Value	Description
Peripheral ID4	0x00000004	*Component and Peripheral ID register formats* in the *ARMv6-M ARM*
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 ARM* and the *ARM CoreSight Components 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 permits a debugger to periodically sample the PC without halting the processor. This provides coarse grained profiling. See the *ARMv6-M ARM* for more information.

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

6.1.4 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 ARM*.

BPU CoreSight identification

Table 6-5 shows the BPU identification registers and their values for debugger detection.

Table 6-5 BPU identification registers

Register	Value	Description
Peripheral ID4	0x00000004	*Component and Peripheral ID register formats* in the *ARMv6-M ARM*
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 ARM* and the *ARM CoreSight Components Technical Reference Manual* for more information about the BPU CoreSight identification registers, and their addresses and access types.

6.2   Debug register summary

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

Table 6-6 Debug registers summary

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

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

Table 6-7 BPU register summary

Name	Description
BP_CTRL	*Breakpoint Control Register* in the *ARMv6-M ARM*
BP_COMP0	*Breakpoint Comparator Registers* in the *ARMv6-M ARM*
BP_COMP1	
BP_COMP2	
BP_COMP3	

Table 6-8 shows the DWT registers. Each of these registers is 32 bits wide.

Table 6-8 DWT register summary

Name	Description
DWT_CTRL	*Control Register* in the *ARMv6-M ARM*
DWT_PCSR	*Program Counter Sample Register* in the *ARMv6-M ARM*
DWT_COMP0	*Comparator Register* in the *ARMv6-M ARM*
DWT_MASK0	*Mask Register* in the *ARMv6-M ARM*
DWT_FUNCTION0	*Function Register* in the *ARMv6-M ARM*

Table 6-8 DWT register summary (continued)

Name	Description
DWT_COMP1	*Comparator Register* in the *ARMv6-M ARM*
DWT_MASK1	*Mask Register* in the *ARMv6-M ARM*
DWT_FUNCTION1	*Function Register* in the *ARMv6-M ARM*

——— **Note** ———

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

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
Update to the product documentation information	*Product documentation, design flow and architecture* on page 1-6	All revisions
Update to the instruction set summary	*Instruction set summary* on page 3-4	All revisions
Clarification of the processor core register set summary	*Processor core registers summary* on page 3-11	All revisions

Table A-3 Differences between issue B and issue C

Change	Location	Affects
No technical changes	-	-

ARM DDI 0432C
ID113009

Glossary

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

Advanced High-performance Bus (AHB)

A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM Limited recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)

A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules.

AHB   *See* Advanced High-performance Bus.

AHB Access Port (AHB-AP)

An optional component of the DAP that provides an AHB interface to a SoC.

AHB-AP   *See* AHB Access Port.

AHB-Lite   A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently by using an AMBA AXI protocol interface.

Aligned   A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.

AMBA   *See* Advanced Microcontroller Bus Architecture.

Architecture   The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6-M architecture.

Big-endian   Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory.

See also Little-endian and Endianness.

Big-endian memory   Memory in which:

- a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address

- a byte at a halfword-aligned address is the most significant byte within the halfword at that address.

See also Little-endian memory.

Breakpoint   A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints can be removed after the program is successfully tested.

See also Watchpoint.

Byte   An 8-bit data item.

Byte-invariant   In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. The ARM architecture supports byte-invariant systems in ARMv6 and later versions.

Core   A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.

Debug Access Port (DAP)

A TAP block that acts as an AMBA, AHB or AHB-Lite, master for access to a system bus. The DAP is the term used to encompass a set of modular blocks that support system wide debug. The DAP is a modular component, intended to be extendable to support optional access to multiple systems such as memory mapped AHB and APB through a single debug interface.

Debugger   A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

Endianness   Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping.

See also Little-endian and Big-endian.

Exception   An error or event which can cause the processor to suspend the currently executing instruction stream and execute a specific exception handler or interrupt service routine. The exception could be an external interrupt or NMI, or it could be a fault or error event that is considered serious enough to require that program execution is interrupted. Examples include attempting to perform an invalid memory access, external interrupts, and undefined instructions. When an exception occurs, normal program flow is interrupted and execution is resumed at the corresponding exception vector. This contains the first instruction of the interrupt service routine to deal with the exception.

Halfword   A 16-bit data item.

Implementation-defined

The behavior is not architecturally defined, but is defined and documented by individual implementations.

Joint Test Action Group (JTAG)

The name of the organization that developed standard IEEE 1149.1. This standard defines a boundary-scan architecture used for in-circuit testing of integrated circuit devices. It is commonly known by the initials JTAG.

JTAG   *See* Joint Test Action Group.

JTAG Debug Port (JTAG-DP)

An optional external interface for the DAP that provides a standard JTAG interface for debug access.

JTAG-DP   *See* JTAG Debug Port.

LE   Little endian view of memory in both byte-invariant and word-invariant systems. See also Byte-invariant, Word-invariant.

Little-endian   Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.

See also Big-endian and Endianness.

Little-endian memory

Memory in which:

- a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address

- a byte at a halfword-aligned address is the least significant byte within the halfword at that address.

See also Big-endian memory.

Load/store architecture

A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.

Multi-layer   An interconnect scheme similar to a cross-bar switch. Each master on the interconnect has a direct link to each slave, The link is not shared with other masters. This enables each master to process transfers in parallel with other masters. Contention only occurs in a multi-layer interconnect at a payload destination, typically the slave.

Processor   A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

Reserved   A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

Serial Wire Debug Port (SW-DP)

The interface for Serial Wire Debug.

Serial Wire Debug (SWD)

A debug implementation that uses a serial connection between the SoC and a debugger. This connection normally requires a bi-directional data signal and a separate clock signal, rather than the four to six signals required for a JTAG connection.

SWD   See Serial Wire Debug.

SW-DP   *See* Serial Wire Debug Port.

TAP   *See* Test Access Port.

Test Access Port (TAP)

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

Thumb instruction   A halfword that specifies an operation for an ARM processor in Thumb state to perform. Thumb instructions must be halfword-aligned.

Thumb state   A processor that is executing Thumb (16-bit) halfword aligned instructions is operating in Thumb state.

Unaligned   A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

Unpredictable   For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.

Watchpoint   A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to enable inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.

Word   A 32-bit data item.