ARM® Cortex®-M23 Processor

Revision: r1p0

Technical Reference Manual

ARM DDI 0550C (ID112116)

ARM Cortex-M23 Processor
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
25 April 2016	A	Confidential	First release for r0p0
29 July 2016	B	Confidential	First release for r1p0
18 November 2016	C	Non-Confidential	Second release for r1p0

Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

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
ARM Cortex-M23 Processor Technical Reference Manual

Preface

This preface introduces the *ARM® Cortex®-M23 Processor Technical Reference Manual*. It contains the following sections:

- *About this book* on page vii
- *Feedback* on page x.

About this book

This book is for the Cortex-M23 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 *Security Attribution and Memory Protection*

Read this chapter for a description of the security attribution and memory protection facilities that the Cortex-M23 processor provides.

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.
`monospace` (with underlined part "mono")	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: `ADD Rd, SP, #<imm>`
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:

- *ARM® v8-M Architecture Reference Manual* (ARM DDI 0553).
- *ARM® AMBA® 5 AHB Protocol Specification, AHB5, AHB-Lite* (ARM IHI 0033).
- *ARM® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (ARM IHI 0031).

  Note

  A Cortex-M23 implementation can include a *Debug Access Port* (DAP). This DAP is defined in v5.1 of the ARM Debug interface specification.

- *Application Binary Interface for the ARM Architecture (The Base Standard)* (ARM IHI 0036).
- *ARM® CoreSight™ SoC-400 Technical Reference Manual* (ARM DDI 0480).
- *ARM® CoreSight™ ETM-M23 Technical Reference Manual* (ARM DDI 0560).
- *ARM® CoreSight™ MTB-M23 Technical Reference Manual* (ARM DDI 0561).

The following confidential books are only available to licensees:

- *ARM® Cortex®-M23 Processor Integration and Implementation Manual* (ARM DIT 0059).

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 0550C.
- 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-M23 processor and its features. It contains the following sections:

1.1 About the processor

The Cortex-M23 processor is a low gate count, two-stage, and highly energy efficient processor. It is intended for microcontroller and deeply embedded applications that require an area optimized, low-power processor for use in environments where security is an important consideration.

1.2 Compliance

This processor is an implementation of the ARMv8-M baseline architecture. For details on the instructions that you can use with this processor, see *Instruction set summary* on page 3-4.

For complete descriptions of all instruction sets, see the *ARM®v8-M Architecture Reference Manual*.

1.3 Features

The processor features and benefits are:

- Tight integration of system peripherals reduces area and development costs.
- Thumb® instruction set that combines high code density with 32-bit performance.
- Optional support for single-cycle I/O access.
- Power control optimization of system components.
- Integrated sleep modes for low power consumption.
- Optimized code fetching for reduced flash and ROM power consumption.
- Hardware multiplier.
- Hardware divider.
- Deterministic, high-performance interrupt handling for time-critical applications.
- Deterministic instruction cycle timing.
- Support for system level debug authentication.
- Support for JTAG or *Serial Wire Debug* (SWD) that reduces the number of pins that are required for debugging.
- Support for optional instruction trace.
- Separated privileged and unprivileged modes.
- Optional Security Extension supporting a Secure and a Non-secure state.
- *Protected Memory System Architecture* (PMSAv8) *Memory Protection Unit*s (MPUs) for both Secure and Non-secure states.
- Optional *Security Attribution Unit* (SAU).
- Optional SysTick timers for both Secure and Non-secure states.
- A *Nested Vectored Interrupt Controller* (NVIC) closely integrated with the processor with up to 240 interrupts.

For information about Cortex-M23 processor architectural compliance, see the *Architecture and protocol information* on page 1-10.

1.4 Interfaces

The interfaces included in the processor for external access include:

- External AMBA® 5 AHB interface.
- *Debug Access Port* (DAP).
- Optional single-cycle I/O port.

1.5 Test Features

The processor is delivered as fully synthesizable RTL and is a fully static design. Scan chains for production test can be inserted into the design by the synthesis tools during implementation.

1.6 Configurable options

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

Table 1-1 Processor configurable options

Feature	Configurable option
Security Extension[a]	Present or absent.
Non-secure MPU	4, 8, 12, 16 regions, or absent.
Secure MPU[b]	4, 8, 12, 16 regions, or absent.
SAU[b]	Absent, 4-region, or 8-region.
SysTick timers	• If the Security Extension is not implemented, can be present or absent. • If the Security Extension is implemented, 0, 1, or 2 SysTick timers can be present. If only one, it is configurable by software if it is Secure or not.
Vector Table Offset Register (VTOR)[c]	• If the Security Extension is not implemented, can be present or absent. • If the Security Extension is implemented, can be either present or absent for both security states.
Reset all registers	Present or absent.
Multiplier	Fast (one cycle) or slow (32 cycles).
Divider	Fast (17 cycles) or slow (34 cycles).
Interrupts	External interrupts 1-240.
Instruction fetch width	16-bit only or 32-bit.
Single-cycle I/O port	Present or absent.
Architectural clock gating present	When set, architectural clock gating cells are instantiated.
Data endianness[d]	Little-endian or byte-invariant big-endian.
Halting debug support	Present or absent.
Wake-up interrupt controller	Supported or not supported.
Number of breakpoint comparators[e]	0, 1, 2, 3, 4.
Number of watchpoint comparators[e]	0, 1, 2, 3, 4.
Cross Trigger Interface (CTI)[e]	Present or absent.
Micro Trace Buffer (MTB)[e]	Present or absent.
Embedded Trace Macrocell (ETM)[e]	Present or absent.
JTAGnSW debug protocol	Selects between JTAG or Serial-Wire interfaces for the DAP.
Multi-drop support for Serial Wire[f]	Present or absent.
Slave port support for AHB DAP	When set, include slave port support for any AHB DAP implementation. Otherwise, support only the low area DAP.

a. There is also the possibility to add or remove the Security Extension with a fuse (input pin).
b. Requires the Security Extension to be present.

c. If the Security Extension is implemented, there is no option to have only one VTOR register. The options are either zero or two.
   If the Security Extension is not implemented, there is no option to have two VTOR registers. The options are either zero or one.

d. Instruction fetches and PPB accesses are always little-endian.

e. Only when Halting debug support is present.

f. Requires Serial-Wire interfaces for DAP.

Note

ETM and MTB are mutually exclusive. Either ETM, MTB, or none of the two is implemented.

1.6.1 Configurable divider

The `UDIV` and `SDIV` instructions provide a 32-bit x 32-bit divide that returns the least-significant 32 bits of the result. The processor can implement `UDIV` and `SDIV` in one of two ways:

- As a 17-cycle iterative divider.
- As a 34-cycle iterative divider.

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

1.6.2 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.

1.7 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 viii for more information about the books that are described in this section.

1.7.1 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 that are 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.

1.7.2 Design Flow

The processor is delivered as synthesizable RTL that must go through the implementation, integration, and programming processes before you can use it in a product.

The following definitions describe each top-level process in the design flow:

Implementation

The implementer configures and synthesizes the RTL.

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 that is 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 synthesize the processor on its own or partially integrated, to produce a macrocell that is then integrated, possibly by a separate team.

The operation of the final device depends on:

Build configuration

The implementer chooses the options that affect how the RTL source files are pre-processed. These options usually include or exclude logic that affects one or more of the area, maximum frequency, and 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 that the appropriate build and pin configuration options have been selected. References to an enabled feature means one that has also been configured by software.

1.7.3 Architecture and protocol information

The processor complies with, or implements, the specifications that are described in:

- *ARM architecture*.
- *Advanced Microcontroller Bus Architecture*.
- *Debug Access Port architecture* on page 1-11.

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 ARMv8-M baseline architecture profile. For more information, see the *ARM®v8-M Architecture Reference Manual*.

Advanced Microcontroller Bus Architecture

The system bus of the processor implements AMBA 5 AHB. For more information, see the *ARM® AMBA® 5 Protocol Specification, AHB5, AHB-Lite*.

Debug Access Port architecture

The *Debug Access Port* (DAP) is an optional component, which is defined by v5.1 of the ARM Debug Interface specification. For more information, see the *ARM® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2*.

1.8 Product revisions

This section describes the differences in functionality between product revisions.

r0p0 First release.

r1p0 Changed the ACTLR register bit assignments.

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 2-5.

2.1 About the functions

The Cortex-M23 processor is a configurable, two-stage, 32-bit RISC processor. It has an AMBA 5 AHB interface and includes an NVIC component. It also has optional hardware debug, single-cycle I/O interfacing, and memory-protection functionality. The processor also supports the Security Extension.

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

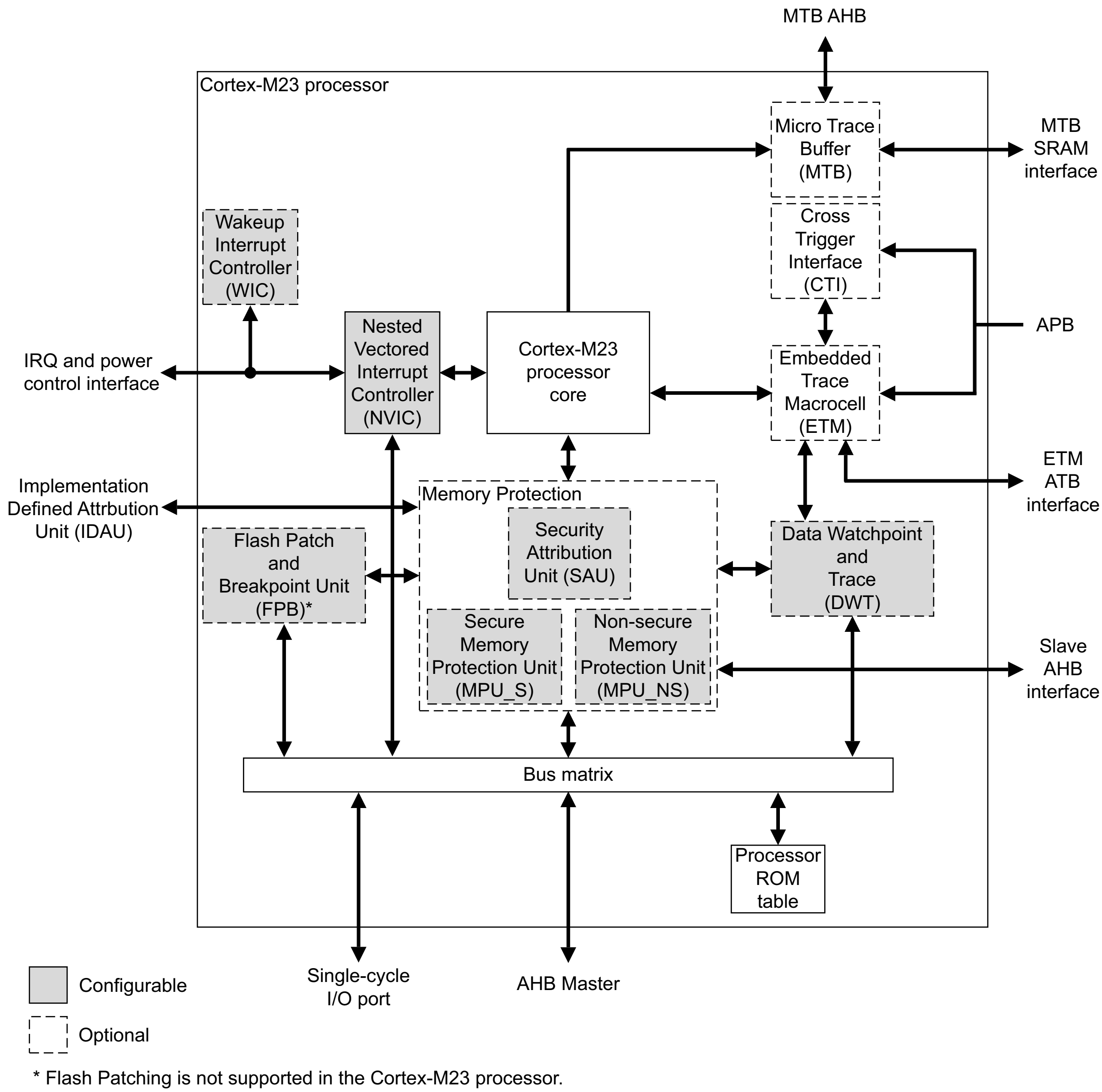

Figure 2-1 Functional block diagram

The implemented device provides:

A low gate count processor that features:

- The ARMv8-M baseline Thumb-2 instruction set.

- Optional Security Extension, including a Secure and a Non-secure state.
- Optionally, an ARMv8-M-compliant 24-bit SysTick timer for each security domain.
- A 32-bit hardware multiplier. This can be the standard single-cycle multiplier, or a 32-cycle multiplier that has a smaller area and lower performance implementation.
- A 32-bit hardware divider. This can be the fast 17-cycle divider, or a slower 34-cycle divider.
- Support for either *little-endian* (LE) or *byte invariant big-endian* (BE8) data accesses.
- The ability to have deterministic and fixed-latency interrupt handling.
- Load/store multiple, multicycle multiply and division instructions that can be abandoned to facilitate rapid interrupt handling.
- Unprivileged/privileged support for improved system integrity.
- C Application Binary Interface compliant exception model.

  This is the ARMv8-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 240 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.

  Note

  There is no internal register to differentiate that the interrupt was pulse or level and no configuration option to support one or the other.
- Optional *Wake-up Interrupt Controller* (WIC), providing ultra-low power sleep mode support.

Optional debug support:

- Halting mode debug only.
- Zero to four hardware breakpoints.
- Zero to four watchpoints.
- *Program Counter Sample Register* (PCSR) for non-intrusive code profiling, if at least one hardware data watchpoint is implemented.
- Single step and vector catch capabilities.
- Breakpoint comparators that allow instruction address matching.
- Support for unlimited software breakpoints using the `BKPT` instruction.
- Non-intrusive access to core peripherals and zero-wait state 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 5 AHB 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 Non-secure *Memory Protection Unit* (MPU):

- Up to 16 user configurable memory regions.
- *eXecute-Never* (XN) support.

Optional Secure *Memory Protection Unit* (MPU):

- Present only if the Security Extension is implemented.
- Up to 16 user configurable memory regions.
- *eXecute-Never* (XN) support.

Optional *Security Attribution Unit* (SAU):

- Present only if the Security Extension is implemented.
- Up to eight user configurable memory regions.
- External *Implementation Defined Attribution Unit* (IDAU). This interface is always present, but is ignored if the Security Extension is not implemented.

Optional *Embedded Trace Macrocell* (ETM):

- Provides a complete instruction trace solution.
- Configurable by an APB slave.
- For more information, see the *ARM® CoreSight™ ETM-M23 Technical Reference Manual.*

Optional *Micro Trace Buffer* (MTB):

- Provides a simple execution trace capability for the processor.
- It offers a lower-cost alternative that has certain limitations compared to ETM.
- For more information, see the *ARM® CoreSight™ MTB-M23 Technical Reference Manual.*

Optional *Cross Trigger Interface* (CTI):

- Enables the debug logic and the ETM to interact with each other and with other CoreSight components.

Optional *Trace Port Interface Unit* (TPIU):

- Present by default when ETM is used.

2.2 Interfaces

This section describes the external interface functions.

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

2.2.1 AMBA 5 AHB interface

Transactions on the AMBA 5 AHB interface are always marked as non-sequential.

Processor accesses and debug accesses share the same 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 AMBA 5 AHB interface. To optimize performance, the 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 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. It 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:

- Serial Wire or JTAG debug port, see the ADI v5.1 version of the *ARM® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2*.
- CoreSight DAP, see the *ARM® CoreSight™ SoC-400 Technical Reference Manual*.

2.2.4 Execution Trace Interface

The Cortex-M23 processor supports two different instruction trace options, ETM and MTB. As a result:

- The processor optionally implements an interface for the ETM execution trace component. See the *ARM® CoreSight™ ETM-M23 Technical Reference Manual* for more information.
- The processor optionally implements an interface for the MTB execution trace component. See the *ARM® CoreSight™ MTB-M23 Technical Reference Manual* for more information.

Note

ETM and MTB options are mutually exclusive.

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 *ARM®v8-M Architecture Reference Manual* provides a complete description of the programmers model. This chapter gives an overview of the Cortex-M23 processor programmers model that describes the implementation-defined options. It also contains the ARMv8-M baseline Thumb-2 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 MPU and SAU features of the programmers model.
- Chapter 7 summarizes the Debug features of the programmers model.

3.2 Modes of operation and execution

If the Security Extension is implemented, the programmers model has Secure and Non-secure views of the operating modes, operating states, and privileged access and unprivileged user access.

If the Security Extension is not implemented, the programmers model has only Non-secure views of the operating modes, operating states, and privileged access and unprivileged user access.

See the *ARM®v8-M Architecture Reference Manual* for information about the modes of operation and execution.

3.2.1 Operating modes

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

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

3.2.2 Operating states

The processor can operate in Thumb or debug state:

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

3.2.3 Privileged access and unprivileged user access

Code can execute as privileged or unprivileged. Unprivileged execution limits or excludes access to some resources. Privileged execution has access to all resources. Handler mode is always privileged. Thread mode can be privileged or unprivileged.

3.3 Instruction set summary

The processor implements the entire ARMv8-M baseline instruction set.

Table 3-1 shows the Cortex-M23processor instructions and their cycle counts. The cycle counts are based on a system with zero wait-states on the AHB bus.

Table 3-1 Cortex-M23 processor instruction summary

Operation	Description	Assembler	Cycles
Move	8-bit immediate	MOVS <Rd>, #<imm>	1
	Lo to Lo[a]	MOVS <Rd>, <Rm>	1
	Any to Any[b]	MOV <Rd>, <Rm>	1
	Any[b] to PC	MOV PC, <Rm>	2
	Top	MOVT <Rd>, #<imm>	3
	Wide	MOVW <Rd>, #<imm>	3
Add	3-bit immediate	ADDS <Rd>, <Rn>, #<imm>	1
	All registers Lo[a]	ADDS <Rd>, <Rn>, <Rm>	1
	Any to Any[b]	ADD <Rd>, <Rd>, <Rm>	1
	Any[b] 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[a]	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[c]
Divide	Unsigned	UDIV {<Rd>,} <Rn>, <Rm>	17 or 34[d]
	Signed	SDIV {<Rd>,} <Rn>, <Rm>	17 or 34[d]

Table 3-1 Cortex-M23 processor instruction summary (continued)

Operation	Description	Assembler	Cycles
Compare	Compare	CMP <Rn>, <Rm>	1
	Negative	CMN <Rn>, <Rm>	1
	Immediate	CMP <Rn>, #<imm>	1
	Compare and Branch on Non-Zero	CBNZ <Rn>, <label>	2 or 1[e]
	Compare and Branch on Zero	CBZ <Rn>, <label>	2 or 1[e]
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
Clear	Exclusive	CLREX	1

Table 3-1 Cortex-M23 processor instruction summary (continued)

Operation	Description	Assembler	Cycles
Load	Word, immediate offset	LDR <Rd>, [<Rn>, #<imm>]	2 or 1[f]
	Halfword, immediate offset	LDRH <Rd>, [<Rn>, #<imm>]	2 or 1[f]
	Byte, immediate offset	LDRB <Rd>, [<Rn>, #<imm>]	2 or 1[f]
	Word, register offset	LDR <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	Halfword, register offset	LDRH <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	Signed halfword, register offset	LDRSH <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	Byte, register offset	LDRB <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	Signed byte, register offset	LDRSB <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	PC-relative	LDR <Rd>, <label>	2 or 1[f]
	SP-relative	LDR <Rd>, [SP, #<imm>]	2 or 1[f]
	Multiple, excluding base	LDM <Rn>!, {<loreglist>}[g]	1+N[h]
	Multiple, including base	LDM <Rn>, {<loreglist>}[g]	3+N[h]
	Exclusive Word	LDREX Rt, [<Rn>{,#<imm>}]	4
	Exclusive Halfword	LDREXH Rt, [<Rn>]	4
	Exclusive Byte	LDREXB Rt, [<Rn>]	4
	Acquire Word	LDA Rt, [<Rn>]	3 or 2[i]
	Acquire Halfword	LDAH Rt, [<Rn>]	3 or 2[i]
	Acquire Byte	LDAB Rt, [<Rn>]	3 or 2[i]
	Acquire Exclusive Word	LDAEX Rt, [<Rn>]	4
	Acquire Exclusive Halfword	LDAEXH Rt, [<Rn>]	4
	Acquire Exclusive Byte	LDAEXB Rt, [<Rn>]	4

Table 3-1 Cortex-M23 processor instruction summary (continued)

Operation	Description	Assembler	Cycles
Store	Word, immediate offset	STR <Rd>, [<Rn>, #<imm>]	2 or 1[f]
	Halfword, immediate offset	STRH <Rd>, [<Rn>, #<imm>]	2 or 1[f]
	Byte, immediate offset	STRB <Rd>, [<Rn>, #<imm>]	2 or 1[f]
	Word, register offset	STR <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	Halfword, register offset	STRH <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	Byte, register offset	STRB <Rd>, [<Rn>, <Rm>]	2 or 1[f]
	SP-relative	STR <Rd>, [SP, #<imm>]	2 or 1[f]
	Multiple	STM <Rn>!, {<loreglist>}[g]	1+N[h]
	Exclusive Word	STREX <Rd>, Rt, [<Rn> {,#<imm>}]	4
	Exclusive Halfword	STREXH <Rd>, Rt, [<Rn>]	4
	Exclusive Byte	STREXB <Rd>, Rt, [<Rn>]	4
	Acquire Word	STL Rt, [<Rn>]	3 or 2[i]
	Acquire Halfword	STLH Rt, [<Rn>]	3 or 2[i]
	Acquire Byte	STLB Rt, [<Rn>]	3 or 2[i]
	Acquire Exclusive Word	STLEX <Rd>, Rt, [<Rn>]	4
	Acquire Exclusive Halfword	STLEXH <Rd>, Rt, [<Rn>]	4
	Acquire Exclusive Byte	STLEXB <Rd>, Rt, [<Rn>]	4
Push	Push	PUSH {<loreglist>}[g]	1+N[h]
	Push with link register	PUSH {<loreglist>, LR}[g]	1+N[j]
Pop	Pop	POP {<loreglist>}[g]	1+N[h]
	Pop and return	POP {<loreglist>, PC}[g]	3+N[j]
	Pop and function return	POP {<loreglist>, PC}[g]	3+N[j]
Branch	Conditional	B<c> <label>	1 or 2[k]
	Unconditional	B <label>	2
	To target	B.W <label>	3
	With link	BL <label>	3
	And exchange	BX <Rm>	2
	And exchange Non-secure	BXNS <Rm>	4
	With function return and exchange	BX{NS} <Rm>	4
	With link and exchange	BLX <Rm>	2
	With link and exchange Non-secure	BLXNS <Rm>	4

Table 3-1 Cortex-M23 processor instruction summary (continued)

Operation	Description	Assembler	Cycles
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 halfword	REVSH <Rd>, <Rm>	1
State change	Supervisor Call	SVC #<imm>	-[l]
	Disable interrupts	CPSID i	1
	Enable interrupts	CPSIE i	1
	Read special register	MRS <Rd>, <specreg>[m]	3
	Write special register	MSR <specreg>, <Rn>[m]	3
	Breakpoint	BKPT #<imm>	-[l]
Hint	Send event	SEV	1
	Wait for event	WFE	2[n]
	Wait for interrupt	WFI	2[n]
	Yield	YIELD	1[o]
	No operation	NOP	1
Barriers	Instruction synchronization	ISB	3
	Data memory	DMB	3
	Data synchronization	DSB	3
Gateway	Secure Gateway	SG	3
Test	Test Target	TT{A}{T} <Rd>, <Rn>	3

a. Lo indicates registers R0-R7.
b. Any indicates registers R0-R15.
c. Depends on multiplier implementation.
d. Depends on divider implementation.
e. 2 if the branch is taken, 1 if the branch is not taken.
f. 2 if to AHB interface or SCS, 1 if to single-cycle I/O port.
g. loreglist indicates registers R0-R7.
h. N is the number of elements in the list.
i. 3 if to AHB interface or SCS, 2 if to single-cycle I/O port.
j. N is the number of elements in the list including PC or LR.
k. 2 if taken, 1 if not-taken.
l. Cycle count depends on processor and debug configuration.
m. specreg indicates special-purpose registers
n. Excludes time that is spent waiting for an interrupt or event.
o. Executes as NOP.

See the *ARM®v8-M Architecture Reference Manual* for more information about the instructions.

3.4 Memory model

The processor contains a bus matrix that arbitrates the processor core and optional 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 non-intrusive for typical code execution.

The system memory map is ARMv8-M baseline architecture compliant, and is common both to the debugger and processor accesses.

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

The default memory map provides user and privileged access to all regions except for the *Private Peripheral Bus* (PPB). The PPB space is privileged access only.

Table 3-2 shows the default memory map. This is the memory map that is used by implementations without the optional MPUs, or when the included MPUs are disabled. The attributes and permissions of all regions, except that targeting the Cortex-M23 processor NVIC and debug components, can be modified using an implemented MPU.

Table 3-2 Default memory map

Address region	Region name	Memory type	Instruction accesses	Data accesses	Description
`0x00000000` - `0x1FFFFFFF`	Code	Normal	AMBA 5 AHB port	AMBA 5 AHB port or I/O port[a]	Typically ROM or Flash. Vector table that is required for boot-up resides here by default. Supports code.
`0x20000000` - `0x3FFFFFFF`	SRAM	Normal	AMBA 5 AHB port	AMBA 5 AHB port or I/O port[a]	On chip RAM. Supports code.
`0x40000000` - `0x5FFFFFFF`	Peripheral	Device	-	AMBA 5 AHB port or I/O port[a]	On chip peripherals. XN
`0x60000000` - `0x9FFFFFFF`	RAM	Normal	AMBA 5 AHB port	AMBA 5 AHB port or I/O port[a]	Supports code.
`0xA0000000` - `0xDFFFFFFF`	Device	Device	-	AMBA 5 AHB port or I/O port[a]	XN.
`0xE0000000` - `0xE003FFFF`	PPB	-	-	Internal PPB interface	SCS, NVIC, MPU, and SAU registers.
`0xE0040000` - `0xE004FFFF`	Device	Device	AMBA 5 AHB port	AMBA 5 AHB port or I/O port[a]	MTB, ETM, CTI, TPIU configuration registers when implemented. XN.

Table 3-2 Default memory map (continued)

Address region	Region name	Memory type	Instruction accesses	Data accesses	Description
`0xE0050000` - `0xE00EFFFF`	PPB	-	-	Internal PPB interface	Reserved. XN.
`0xE00F0000` - `0xE00FFFFF`	Device	Device	AMBA 5 AHB port	AMBA 5 AHB port or I/O port[a]	Cortex-M23 MCU ROM when implemented. XN.
`0xE0100000` - `0xFFFFFFFF`	Vendor_SYS	Device	-	AMBA 5 AHB port or I/O port[a]	Suitable for CoreSight ROM tables and other CoreSight components. XN.

a. Only when the I/O port is included.

Note

- Regions that are marked as *eXecute-Never* (XN) generate a HardFault exception if code attempts to execute from such a location.
- If an MPU is included, it can be used to control accesses to all regions other than the PPB. Attempted accesses that do not meet the required privilege level for the respective region are not performed on the bus and cause a HardFault exception.
- The MPU can override all region attributes except the PPB space. See Chapter 6 *Security Attribution and Memory Protection*.
- Some regions can be defined as secure by the SAU and/or the IDAU. In this case a Non-secure access triggers a secure Hardfault. Region `0xEXXXXXXX` cannot be changed by the SAU and/or the IDAU. Region `0xFXXXXXXX` is always Secure and not Non-secure callable for instructions. For more information, see Chapter 6 *Security Attribution and Memory Protection*.

See the *ARM®v8-M Architecture Reference Manual* for more information about the memory model.

3.5 Registers summary

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

Table 3-3 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 registers indicate the Stack Pointers to use, *Main Stack Pointer* (MSP) or *Process Stack Pointer* (PSP). **If the Security Extension is implemented:** There are two MSP registers in the Cortex-M23 processor: • MSP_NS for the Non-secure state. • MSP_S for the Secure state. There are two PSP registers in the Cortex-M23 processor. • PSP_NS for the Non-secure state. • PSP_S for the Secure state. **If the Security Extension is not implemented:** There is one MSP register and one PSP register in the Cortex-M23 processor.
MSPLIM_S[a] PSPLIM_S[a]	The stack limit registers limit the extent to which the MSP_S and PSP_S registers can descend respectively.
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 3-14. **If the Security Extension is implemented:** There are two PRIMASK registers in the Cortex-M23 processor: • PRIMASK_NS for the Non-secure state. • PRIMASK_S for the Secure state. **If the Security Extension is not implemented:** There is one PRIMASK register in the Cortex-M23 processor.
CONTROL	The CONTROL registers control the stack that is used, and optionally the code privilege level, when the processor is in Thread mode. **If the Security Extension is implemented:** There are two CONTROL registers in the Cortex-M23 processor: • CONTROL_NS for the Non-secure state. • CONTROL_S for the Secure state. **If the Security Extension is not implemented:** There is one CONTROL register in the Cortex-M23 processor.

a. If Security Extension is not implemented, the Cortex-M23 processor does not support stack limit registers.

See the *ARM®v8-M Architecture Reference Manual* for information about the processor registers and their addresses, access types, and reset values.

Note

`PUSH` instructions starting below the stack limit are not performed. Exception stacking going below the limit may be partially written on the memory for address equal to or above the stack limit.

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 *ARM®v8-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.
- The processor abandons both 17-cycle and 34-cycle divide instructions in the same way.
- When a `BX` or `POP PC` causes a function return, the instruction becomes non-interruptible during unstacking phase.

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 that is supplied by the processor implementer for more information.

To reduce interrupt latency and jitter, the Cortex-M23 processor implements both interrupt late-arrival and interrupt tail-chaining mechanisms, as defined by the ARMv8-M architecture.

In a zero wait-state system, excluding late arriving interrupts:

- If the Security Extension is not implemented or when a short stack is used, the interrupt latency is 15 cycles.
- If the Security Extension is implemented and a full stack is used (Non-secure exception interrupts code executing in Secure state), the interrupt latency is 27 cycles.

Note

- The number of cycles are cycles after the interrupt is set, up to the data phase of the first instruction in the handler.
- If the Security Extension is implemented, this number assumes no fault happening during interrupt return and stacking. Latency cannot be guaranteed in case of fault.

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*, see the *ARM®v8 -M Architecture Register Manual*. **If the Security Extension is implemented:** There are two SYST_CSR registers in the Cortex-M23 processor: • SYST_CSR_NS for the Non-secure state (optional). • SYST_CSR_S for the Secure state.[a] **If the Security Extension is not implemented:** There is one SYST_CSR register in the Cortex-M23 processor.
SYST_RVR	*SysTick Reload Value Register*, see the *ARM®v8 -M Architecture Register Manual*. **If the Security Extension is implemented:** There are two SYST_RVR registers in the Cortex-M23 processor: • SYST_RVR_NS for the Non-secure state (optional). • SYST_RVR_S for the Secure state.[a] **If the Security Extension is not implemented:** There is one SYST_RVR register in the Cortex-M23 processor.
SYST_CVR	*SysTick Current Value Register*, see the *ARM®v8 -M Architecture Register Manual*. **If the Security Extension is implemented:** There are two SYST_CVR registers in the Cortex-M23 processor: • SYST_CVR_NS for the Non-secure state (optional). • SYST_CVR_S for the Secure state.[a] **If the Security Extension is not implemented:** There is one SYST_CVR register in the Cortex-M23 processor.
SYST_CALIB[b]	*SysTick Calibration value Register*, see the *ARM®v8 -M Architecture Register Manual*. **If the Security Extension is implemented:** There are two SYST_CALIB registers in the Cortex-M23 processor: • SYST_CALIB_NS for the Non-secure state (optional). • SYST_CALIB_S for the Secure state.[a] **If the Security Extension is not implemented:** There is one SYST_CALIB register in the Cortex-M23 processor.
CPUID	See *CPUID Register* on page 4-5.
ICSR	*Interrupt Control and State Register*, see the *ARM®v8-M Architecture Reference Manual*.
AIRCR[b]	*Application Interrupt and Reset Control Register*, see the *ARM®v8-M Architecture Reference Manual*.
CCR	*Configuration and Control Register, see* the *ARM®v8-M Architecture Reference Manual*.
SHPR2	*System Handler Priority Register 2*, see the *ARM®v8-M Architecture Reference Manual*.
SHPR3	*System Handler Priority Register 3*, see the *ARM®v8-M Architecture Reference Manual*.

Table 4-1 System control registers (continued)

Name	Description
SHCSR	*System Handler Control and State Register*, see the *ARM®v8-M Architecture Reference Manual*.
VTOR[c]	*Vector table Offset Register*, see the *ARM®v8 -M Architecture Register Manual*. **If the Security Extension is implemented:** There are two VTOR registers in the Cortex-M23 processor:[d] • VTOR_NS for the Non-secure state (optional). • VTOR_S for the Secure state (optional). **If the Security Extension is not implemented:** There is one VTOR register in the Cortex-M23 processor (optional).
ACTLR	See *ACTLR Register*.

a. If there is only one SysTick timer present, it is configurable by software if this register is Secure or not.

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

c. The initial value of VTOR is determined by external input pins on the processor's top level. This determines the vector table that is used for boot up sequence.

If implemented, the VTOR enables bits[31:8] of the vector table address to be specified. The reset value can be configured by external pins. Bits[9:8] can be RAZ/WI depending on the number of interrupts.

If not implemented, the registers are RO/WI and report the value of the external pins.

d. There is no option to have only one VTOR register if the Security Extension is implemented.

Note

- All system control registers are only accessible using word transfers. Any attempt to read or write a halfword or byte generates a Hardfault.
- If the processor is implemented without the SysTick timers, the SYST_CSR, SYST_RVR, SYST_CVR, and SYST_CALIB registers are RAZ/WI.
- If the processor is implemented with a single SysTick timer, the SYST_CSR_NS, SYST_RVR_NS, SYST_CVR_NS, and SYST_CALIB_NS registers are RAZ/WI.
- See the *ARM®v8-M Architecture Reference Manual* for more information about the system control registers, and their addresses and access types, and reset values that are not shown in Table 4-1 on page 4-3.

4.2.1 ACTLR Register

The ACTLR characteristics are:

Purpose Provides configuration and control options regarding the use of the Global Exclusive Monitor by the LDREX and STREX instructions.

Usage constraints Privileged access permitted only. Unprivileged accesses generate a BusFault.

This register is word accessible only. Halfword and byte accesses are UNPREDICTABLE.

Attributes See Table 4-2 on page 4-5.

Figure 4-1 on page 4-5 shows the ACTLR bit register assignments.

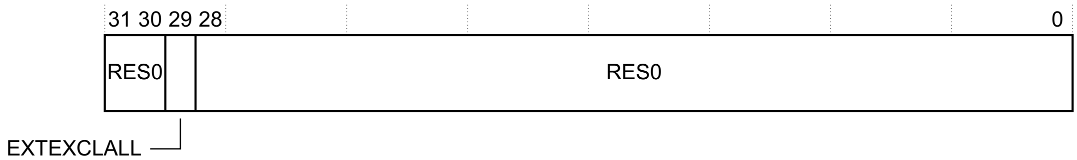

Figure 4-1 ACTLR bit register assignments

Table 4-3 on page 4-6 shows the ACTLR register bit assignments.

Table 4-2 ACTLR bit register assignments

Bits	Field	Function	
[31:30]	-	RES0	
[29]	EXTEXCLALL	`0x0`	`LDREX` and `STREX` instructions only use the Global Exclusive Monitor on Shared regions, either in the default memory map, or when hitting in a Shared MPU region.
		`0x1`	`LDREX` and `STREX` instructions always use the Global Exclusive Monitor, even if the memory region is not Shared. **Note** • If the Security Extension is implemented, this bit is banked between Security states. • Accesses to Device regions in the ranges `0x40000000-0x5FFFFFFF` and `0xC0000000-0xFFFFFFFF` do not use the Global Exclusive Monitor when ACTLR.EXTEXCLALL is 0 and the default memory map is used.
[28:0]	-	RES0	

4.2.2 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 on page 4-3.

Figure 4-2 shows the CPUID bit register assignments.

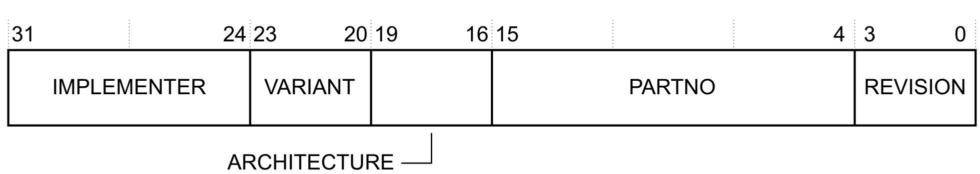

Figure 4-2 CPUID bit register assignments

Table 4-3 shows the CPUID register bit assignments.

Table 4-3 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 vii: `0x1`.
[19:16]	ARCHITECTURE	Indicates the architecture, ARMv8-M baseline: `0xC`.
[15:4]	PARTNO	Cortex-M23 processor part number: `0xD20`.
[3:0]	REVISION	Minor revision number *m* in the r*n*p*m* revision status. See *Product revision status* on page vii. `0x0`.

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 supports four programmable levels of priority while AIRCR.PRIS increases the levels to eight, as it splits Secure and Non-secure priorities. The NVIC and the Cortex-M23 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 generates a Hardfault.

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 3-14.

5.1.1 Low-power modes

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. The SLEEPONEXIT is banked depending on the security states.

The implementation can include a WIC. This enables the processor and NVIC to be put into a low-power sleep mode leaving the WIC to identify and prioritize interrupts.

See the *ARM®v8-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_ISERn	*Interrupt Set-Enable Register n*.
NVIC_ICERn	*Interrupt Clear-Enable Register n*.
NVIC_ISPRn	*Interrupt Set-Pending Register n*.
NVIC_ICPRn	*Interrupt Clear-Pending Register n.*
NVIC_IABRn	*Interrupt Active Bit Register n*.
NVIC_ITNSn[a]	*Interrupt Target Non-Secure Register n*.
NVIC_IPRn	*Interrupt Priority Registers n*.

a. Present only when the Security Extension is implemented.

Note

See the *ARM®v8-M Architecture Reference Manual* for more information about the NVIC registers and their addresses, access types, and reset values.

Chapter 6
Security Attribution and Memory Protection

This chapter describes the security attribution and memory protection facilities that the Cortex-M23 processor provides. It contains the following sections:

6.1 About Security Attribution and Memory Protection

Security attribution and memory protection in the processor is provided by the optional SAU and the optional MPUs.

The SAU is an optional component that determines the security of an address. If the Security Extension is not implemented, the SAU is also not implemented. Otherwise, it can optionally be implemented and supports zero, four, or eight regions.

For instructions, the SAU returns the security attribute (Secure or Non-secure) and identifies whether the instruction address is in a Non-secure callable region.

For data, the SAU returns the security attribute and checks whether the core is running in Non-secure state and tries to access a Secure region. If this happens, a HardFault is generated.

The security level returned by the SAU is a combination of the region type defined in the internal SAU, if configured, and the type that is returned on the associated *Implementation Defined Attribution Unit* (IDAU). If an address maps to regions defined by both internal and external attribution units, the region of the highest security level is selected.

Table 6-1 Examples of Highest Security Level Region

IDAU	SAU Region	Final Security
S	X	S
X	S	S
S-NSC or NS	S-NSC	S-NSC
NS	NS	NS

At reset, before any SAU regions are programmed, the SAU_CTRL.ALLNS register bit selects the default internal security level. On reset the SAU_CTRL.ALLNS register is always reset to zero, setting all memory, apart from some specific regions in the PPB space, to Secure state. Setting SAU_CTRL.ALLNS bit to zero prevents an external SAU overriding any security level.

The MPU is an optional component for memory protection. When implemented, the processor supports the ARMv8-M *Protected Memory System Architecture* (PMSA) model. The MPU provides full support for:

- Protection regions.
- Access permissions.
- Exporting memory attributes to the system.

MPU mismatches and permission violations invoke the HardFault handler. See the *ARM®v8-M Architecture Reference Manual* for more information.

You can use the MPU to:

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

The Cortex-M23 processor includes up to two MPUs depending on whether it implements the Security Extension or not:

- If the Cortex-M23 processor does not implement the Security Extension, it can optionally contain one MPU that supports 0, 4, 8, 12, or 16 memory regions.

- If the Cortex-M23 processor implements the Security Extension, it contains:
  - One optional Secure MPU (MPU_S).
  - One optional Non-secure MPU (MPU_NS).
  - One optional SAU.

See the *ARM®v8-M Architecture Reference Manual* for more information on SAU and MPUs.

6.2 SAU register summary

Table 6-2 shows the SAU registers. Each of these registers is 32 bits wide.

Table 6-2 SAU registers

Name	Reset value	Description
SAU_CTRL	`0x00000000`[a]	*Security Attribution Unit Control Register*.
SAU_TYPE[b]	UNKNOWN	*Security Attribution Unit Type Register*.
SAU_RNR	UNKNOWN	*Security Attribution Unit Region Number Register*.
SAU_RBAR	UNKNOWN	*Security Attribution Unit Region Base Address Register.*
SAU_RLAR	`0x00000000`[c]	*Security Attribution Unit Region Limit Address Register*.

a. The reset value is `0x00000002` if the Security Extension is not implemented (write ignored).

b. Bits[7:0] depend on the number of SAU regions included. This value can be 0, 4, or 8.

c. Read-only.

Note

See the *ARM®v8-M Architecture Reference Manual* for more information about the SAU registers and their addresses, access types, and reset values.

6.3 MPU register summary

Table 6-3 shows the MPU registers. Each of these registers is 32 bits wide. If the MPU is not present in the implementation, then all these registers read as zero.

Table 6-3 MPU registers

Name	Reset value	Description
MPU_TYPE[a]	`0x0000XX00`	MPU Type Register.
MPU_CTRL	`0x00000000`	MPU Control Register.
MPU_RNR	UNKNOWN	MPU Region Number Register.
MPU_RBAR	UNKNOWN	MPU Region Base Address Register.
MPU_RLAR	UNKNOWN	MPU Region Limit Address Register
MPU_MAIR0	UNKNOWN	MPU Memory Attribute Indirection Register 0
MPU_MAIR1	UNKNOWN	MPU Memory Attribute Indirection Register 1

a. Bits[15:8] depend on the number of MPU regions included. This value can be 0, 4, 8, 12, or 16.

Note

- See the *ARM®v8-M Architecture Reference Manual* for more information about the MPU registers and their addresses, access types, and reset values.
- If the Security Extension is implemented, the registers are aliased.
- The MPU supports region sizes from 32 bytes to 4GB.

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 *ARM®v8-M Architecture Reference Manual*.

The debug option might include either or both:

- A breakpoint unit supporting one, two, three, or four hardware breakpoints.
- A watchpoint unit supporting one, two, three, or four 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.

All debug registers are accessible by the DAP interface.

To discover the components in the CoreSight debug infrastructure, ARM recommends that a debugger follows the flow that is 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.

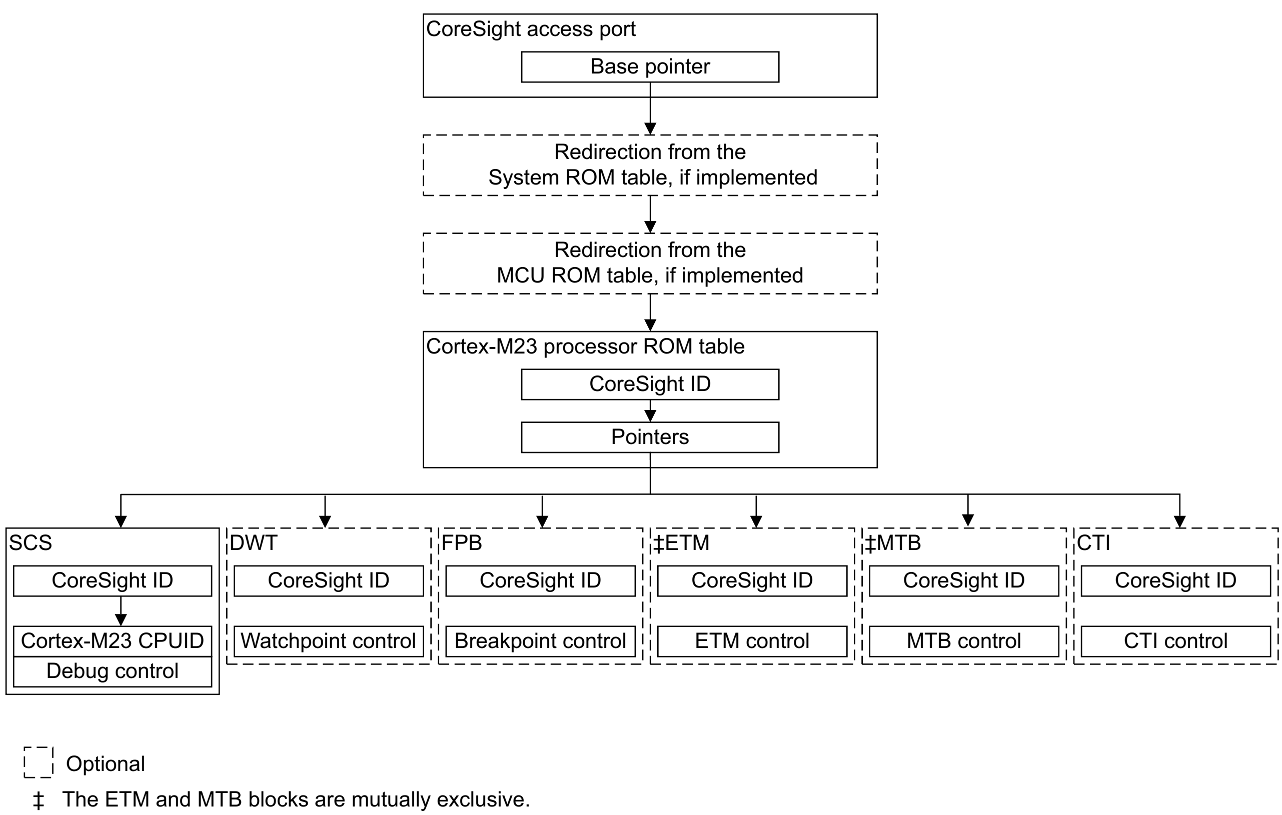

Figure 7-1 CoreSight discovery

To identify the Cortex-M23 processor within the CoreSight system, ARM recommends that a debugger:

1. Locates and identifies the Cortex-M23 processor ROM table using its CoreSight identification. See *Cortex-M23 processor ROM table identification values* on page 7-4.
2. Follows the pointers in that Cortex-M23 processor ROM table:
   a. *System Control Space* (SCS).
   b. *Flash Patch and Breakpoint Unit* (FPB).
   c. *Data watchpoint unit* (DWT).
   d. *Embedded Trace Macrocell* (ETM).
   e. *Micro Trace Buffer* (MTB).
   f. *Cross Trigger Interface* (CTI).

   See *Cortex-M23 processor ROM table components*.

   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-M23 processor ROM table being the first ROM table encountered. One or more system ROM tables are required between the access port and the Cortex-M23 processor ROM table if other CoreSight components are in the system, or if the implementation is to be uniquely identifiable.

7.1.1 Cortex-M23 processor ROM table identification and entries

Table 7-1 shows the CoreSight components that the Cortex-M23 processor ROM table points to. The values depend on the implemented debug configuration.

Table 7-1 Cortex-M23 processor ROM table components

Address	Component	Value	Description
`0xE00FF000`	SCS	`0xFFF0F003`	See *System Control Space* on page 7-4.
`0xE00FF004`	DWT	`0xFFF02003`[a]	See *Data watchpoint unit* on page 7-5.
`0xE00FF008`	FPB	`0xFFF03003`[b]	See *Flash Patch and Breakpoint unit* on page 7-6.
`0xE00FF00C`	Reserved	`0xFFF01002`	-
`0xE00FF010`	Reserved	`0xFFF41002`	-
`0xE00FF014`	ETM	`0xFFF42003`[c]	See *Embedded Trace Macrocell* on page 7-7.
`0xE00FF018`	CTI	`0xFFF43003`[d]	See *Cross Trigger Interface* on page 7-7.
`0xE00FF01C`	MTB	`0xFFF44003`[e]	See *Micro Trace Buffer* on page 7-7.
`0xE00FF020`	End marker	`0x00000000`	See the *ARM®v8-M Architecture Reference Manual*.

a. Reads as `0xFFF02002` if no watchpoints are implemented.
b. Reads as `0xFFF03002` if no breakpoints are implemented.
c. Reads as `0xFFF42002` if ETM is not implemented.
d. Reads as `0xFFF43002` if CTI is not implemented.
e. Reads as `0xFFF44002` if MTB is not implemented.

The SCS, DWT, FPB, ETM, MTB, and CTI ROM table entries point to the debug components at addresses `0xE000E000`, `0xE0001000`, `0xE0002000`, `0xE0041000`, `0xE0043000`, and `0xE0042000` respectively. The value for each entry is the offset of that component from the ROM table base address, `0xE00FF000`.

Table 7-2 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-M23 processor ROM table only supports word size transactions.

Table 7-2 Cortex-M23 processor ROM table identification values

Address	Register	Value	Description
`0xE00FFFD0`	Peripheral ID4	`0x00000004`	*Component and peripheral ID register formats* in the *ARM®v8-M Architecture Reference Manual.*
`0xE00FFFE0`	Peripheral ID0	`0x000000CB`	
`0xE00FFFE4`	Peripheral ID1	`0x000000B4`	
`0xE00FFFE8`	Peripheral ID2	`0x0000000B`	
`0xE00FFFEC`	Peripheral ID3	`0x00000000`	
`0xE00FFFF0`	Component ID0	`0x0000000D`	
`0xE00FFFF4`	Component ID1	`0x00000010`	
`0xE00FFFF8`	Component ID2	`0x00000005`	
`0xE00FFFFC`	Component ID3	`0x000000B1`	

See the *ARM®v8-M Architecture Reference Manual* and the *CoreSight SoC-400 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 7-9.

SCS CoreSight identification

Table 7-3 shows the SCS CoreSight identification registers and values for debugger detection. Final debugger identification of the Cortex-M23 processor is through the CPUID register in the SCS, see *CPUID Register* on page 4-5.

Table 7-3 SCS identification values

Address	Register	Value	Description
0xE000EFD0	Peripheral ID4	0x00000004	See the *ARM®v8-M Architecture Reference Manual.*
0xE000EFE0	Peripheral ID0	0x00000020	
0xE000EFE4	Peripheral ID1	0x000000BD	
0xE000EFE8	Peripheral ID2	0x0000000B	
0xE000EFEC	Peripheral ID3	0x00000000	
0xE000EFF0	Component ID0	0x0000000D	
0xE000EFF4	Component ID1	0x00000090	
0xE000EFF8	Component ID2	0x00000005	
0xE000EFFC	Component ID3	0x000000B1	

See the *ARM®v8-M Architecture Reference Manual* and the *CoreSight SoC-400 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-M23 processor DWT implementation provides between zero and four watchpoint register sets. A processor that is 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 *ARM®v8-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

Address	Register	Value	Description
0xE0001FD0	Peripheral ID4	0x00000004	See the *ARM®v8-M Architecture Reference Manual.*
0xE0001FE0	Peripheral ID0	0x00000020	
0xE0001FE4	Peripheral ID1	0x000000BD	
0xE0001FE8	Peripheral ID2	0x0000000B	
0xE0001FEC	Peripheral ID3	0x00000000	
0xE0001FF0	Component ID0	0x0000000D	
0xE0001FF4	Component ID1	0x00000090	
0xE0001FF8	Component ID2	0x00000005	
0xE0001FFC	Component ID3	0x000000B1	

See the *ARM®v8-M Architecture Reference Manual* and the *ARM® CoreSight™ SoC-400 Technical Reference Manual* for more information about the DWT CoreSight identification registers, and their addresses and access types.

DWT Program Counter Sample Register

The Cortex-M23 processor implements the ARMv8-M optional *DWT Program Counter Sample Register* (DWT_PCSR) when there is at least one DWT. This register enables a debugger to periodically sample the PC without halting the processor. This provides coarse grained profiling. See the *ARM®v8-M Architecture Reference Manual* for more information.

The Cortex-M23 processor DWT_PCSR records both instructions that pass their condition codes and those instructions that fail.

7.1.4 Flash Patch and Breakpoint unit

The Cortex-M23 processor FPB implementation provides between zero and four breakpoint registers. A processor that is configured with zero breakpoints implements no breakpoint functionality and the ROM table shows that no FPB is implemented.

FPB functionality

The processor breakpoints implement PC-based breakpoint functionality, as described in the *ARM®v8-M Architecture Reference Manual*.

FPB CoreSight identification

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

Table 7-5 FPB identification registers

Address	Register	Value	Description
`0xE0002FD0`	Peripheral ID4	`0x00000004`	See the *ARM®v8-M Architecture Reference Manual*.
`0xE0002FE0`	Peripheral ID0	`0x00000020`	
`0xE0002FE4`	Peripheral ID1	`0x000000BD`	
`0xE0002FE8`	Peripheral ID2	`0x0000000B`	
`0xE0002FEC`	Peripheral ID3	`0x00000000`	
`0xE0002FF0`	Component ID0	`0x0000000D`	
`0xE0002FF4`	Component ID1	`0x00000090`	
`0xE0002FF8`	Component ID2	`0x00000005`	
`0xE0002FFC`	Component ID3	`0x000000B1`	

See the *ARM®v8-M Architecture Reference Manual* and the *CoreSight SoC-400 Technical Reference Manual* for more information about the FPB CoreSight identification registers, and their addresses and access types.

7.1.5 Embedded Trace Macrocell

The Cortex-M23 processor optionally implements an ETM module that provides a complete trace solution.

Note

You can either implement an ETM, an MTB, or none of the two in your system.

See the *ARM® CoreSight™ ETM-M23 Technical Reference Manual* for information about the ETM CoreSight identification registers.

7.1.6 Micro Trace Buffer

The Cortex-M23 processor optionally implements an MTB module that provides a simple execution trace capability.

Note

You can either implement an ETM, an MTB, or none of the two in your system.

See the *ARM® CoreSight™ MTB-M23 Technical Reference Manual* for information about the MTB CoreSight identification registers.

7.1.7 Cross Trigger Interface

The Cortex-M23 processor implements a CTI that enables the debug logic and the ETM to interact with each other and with other CoreSight components.

CTI CoreSight identification

Table 7-6 shows the CTI identification registers and values for debugger detection.

Table 7-6 CTI identification values

Address	Register	Value	Description
0xE0042FD0	Peripheral ID4	0x00000004	See the *ARM® CoreSight™ SoC-400 Technical Reference Manual*.
0xE0042FE0	Peripheral ID0	0x00000020	
0xE0042FE4	Peripheral ID1	0x000000BD	
0xE0042FE8	Peripheral ID2	0x0000000B	
0xE0042FEC	Peripheral ID3	0x00000000	
0xE0042FF0	Component ID0	0x0000000D	
0xE0042FF4	Component ID1	0x00000090	
0xE0042FF8	Component ID2	0x00000005	
0xE0042FFC	Component ID3	0x000000B1	

See the *ARM® CoreSight™ SoC-400 Technical Reference Manual* for more information about the CTI CoreSight identification registers, and their addresses and access types.

7.2 Debug register summary

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

Table 7-7 Debug registers summary

Name	Description
DAUTHCTRL	*Debug Authentication Control Register* in the *ARM®v8-M Architecture Reference Manual*. **Note** This register is accessible by software from the Cortex-M23 processor and is RAZ/WI when accessed from the debugger.
DAUTHSTATUS	*Debug Authentication Status Register* in the *ARM®v8-M Architecture Reference Manual*.
DDEVARCH	*Device Architecture Register* in the *ARM®v8-M Architecture Reference Manual.*
DFSR	*Debug Fault Status Register* in the *ARM®v8-M Architecture Reference Manual*.
DHCSR	*Debug Halting Control* and Status Register in the *ARM®v8-M Architecture Reference Manual.*
DCRSR	*Debug Core Register Select Register* in the *ARM®v8-M Architecture Reference Manual*.
DCRDR	*Debug Core Register Data Register* in the *ARM®v8-M Architecture Reference Manual.*
DEMCR	*Debug Exception and Monitor Control Register* in the *ARM®v8-M Architecture Reference Manual*.
DSCSR	*Debug Security Control and Status Register* in the *ARM®v8-M Architecture Reference Manual*.

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

Table 7-8 FPB register summary

Name	Description
FP_CTRL	*Flash Patch Control Register* in the *ARM®v8-M Architecture Reference Manual.*
FP_DEVARCH	*FPB Device Architecture Register* in the *ARM®v8-M Architecture Reference Manual.*
FP_COMP0	*Flash Patch Comparator Registers* in the *ARM®v8-M Architecture Reference Manual*.
FP_COMP1	
FP_COMP2	
FP_COMP3	

Table 7-9 shows the DWT registers. Each of these registers is 32 bits wide.

Table 7-9 DWT register summary

Name	Description
DWT_CTRL	*DWT Control Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_DEVARCH	*DWT Device Architecture Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_CYCCNT	*DWT Cycle Count Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_CPICNT	*DWT CPI Count Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_EXCCNT	*DWT Exception Overhead Count Register* in the *ARM®v8-M Architecture Reference Manual*.

Table 7-9 DWT register summary (continued)

Name	Description
DWT_SLEEPCNT	*DWT Sleep Count Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_LSUCNT	*DWT LSU Count Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_FOLDCNT	*DWT Folded Instruction Count Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_PCSR	*DWT Program Counter Sample Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_COMPn	*DWT Comparator Register* in the *ARM®v8-M Architecture Reference Manual*.
DWT_FUNCTIONn	*DWT Function Register* in the *ARM®v8-M Architecture Reference Manual*.

See the *ARM® CoreSight™ ETM-M23 Technical Reference Manual* for information about the ETM registers.

See the *ARM® CoreSight™ MTB-M23 Technical Reference Manual* for information about the MTB registers.

Table 7-10 shows the CTI registers. Each of these registers is 32 bits wide.

Table 7-10 CTI register summary

Name	Description
CTICONTROL	See the *ARM® CoreSight™ SoC-400 Technical Reference Manual*.
CTIINTACK	
CTIAPPSET	
CTIAPPCLEAR	
CTIAPPPULSE	
CTIINEN[7:0]	
CTIINEN1	
CTIOUTEN2[7:0]	
CTITRIGINSTATUS	
CTITRIGOUTSTATUS	
CTICHINSTATUS	
CTICHOUTSTATUS	
CTIGATE	
ASICCTL	
ITCHINACK	
ITTRIGINACK	
ITCHOUT	
ITTRIGOUT	
ITCHOUTACK	
ITTRIGOUTACK	
ITCHIN	
ITTRIGIN	
ITCTRL	
CLAIMSET	
CLAIMCLR	
LAR	
LSR	
AUTHSTATUS	
DEVID	
DEVARCH	

See the *ARM®v8-M Architecture Reference Manual* 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
Updated the exception handling section	*Exception handling* on page 3-14	r1p0
Added the description of the ACTLR Register	*ACTLR Register* on page 4-4	r1p0
Updated the major revision number in the CPUID register	*CPUID Register* on page 4-5	r1p0
Updated the Security Attribution and Memory Protection section	*About Security Attribution and Memory Protection* on page 6-2	r1p0

Table A-3 Differences between issue B and issue C

Change	Location	Affects
Changed the product name to Cortex-M23	-	r1p0
Added a note in the Registers summary	*Registers summary* on page 3-12	r1p0
Updated the ACTLR.EXTEXCLALL description	*ACTLR Register* on page 4-4	r1p0