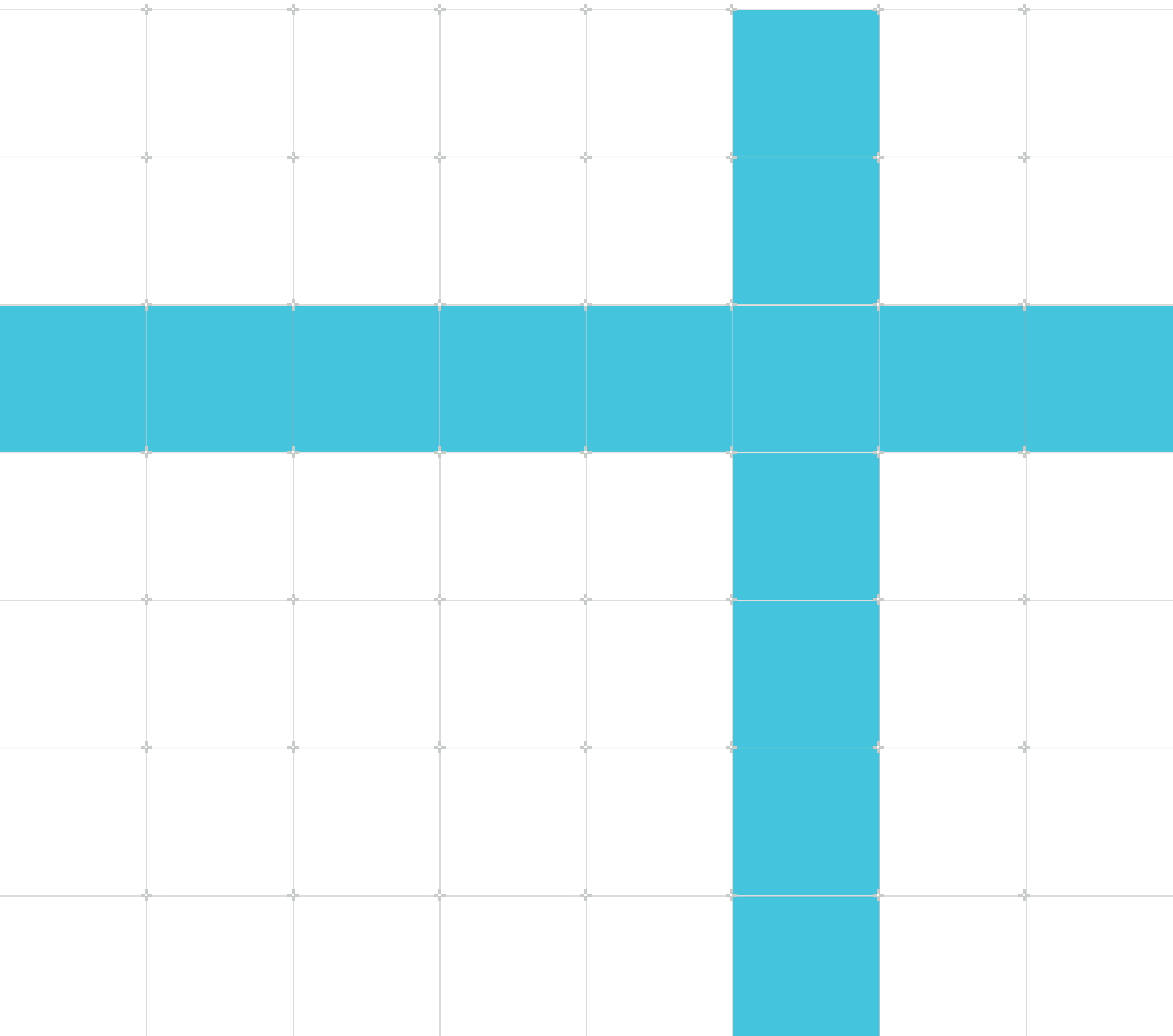
Arm® Cortex®-M23 Processor

Revision: r2p0

Technical Reference Manual

Issue
DDI0550D_0200_en



Arm® Cortex®-M23 Processor

Technical Reference Manual

Release Information

Document history

Issue	Date	Confidentiality	Change
A	25 April 2016	Confidential	First release for r0p0
B	29 July 2016	Confidential	First release for r1p0
C	18 November 2016	Non-Confidential	Second release for r1p0
D	31 January 2023	Non-Confidential	First release for r2p0

Proprietary Notice

REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

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.

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction

1.1 Product revision status

The r*x*p*y* identifier indicates the revision status of the product described in this manual, for example, r1p2, where:

r*x* Identifies the major revision of the product, for example, r1.

p*y* Identifies the minor revision or modification status of the product, for example, p2.

1.2 Intended audience

This book is written for: System designers, system integrators, and verification engineers. Software developers who want to use the processor.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

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

See the Arm Glossary for more information: developer.arm.com/glossary.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`

Convention	Use
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Recommendations. Not following these recommendations might lead to system failure or damage.

Requirements for the system. Not following these requirements might result in system failure or damage.

Requirements for the system. Not following these requirements will result in system failure or damage.

An important piece of information that needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

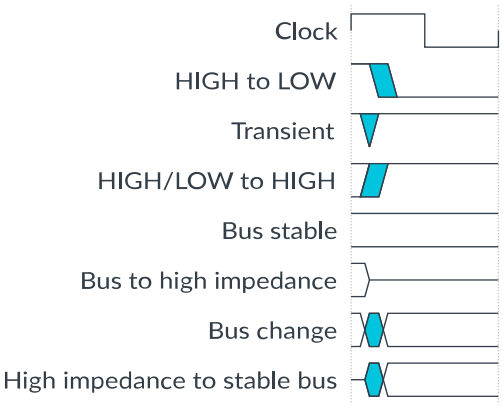
A reminder of something important that relates to the information you are reading.

Timing diagrams

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

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

Figure 1-1: Key to timing diagram conventions



Signals

The signal conventions are:

Signal level

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

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

Lowercase n

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

1.4 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Table 1-2: Arm Publications

Arm product resources	Document ID	Confidentiality
Arm®v8-M Architecture Reference Manual	Arm DDI 0553	Non-Confidential
Arm® AMBA® 5 AHB Protocol Specification, AHB5, AHB-Lite	Arm IHI 0033	Non-Confidential
Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2[1]	Arm IHI 0031	Non-Confidential

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

Arm product resources	Document ID	Confidentiality
Application Binary Interface for the Arm Architecture (The Base Standard)	Arm IHI 0036	Non-Confidential
Cortex®-M23 Processor Integration and Implementation Manual	Arm DIT 0059	Confidential

Table 1-3: Other publication

Non-Arm resources	Document ID	Organization
IEEE Standard, TestAccess Port and Boundary-Scan Architecture specification	1149.1-1990(JTAG)	


Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com.

2. Cortex-M23 Introduction

It contains the following sections:

2.1 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.

2.2 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 4.3 Instruction set summary on page 28.

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

2.3 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 Units* (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.
- Safety features
- Optional support for safety features: transient fault protection with flop parity, interface protection and STL hardware

For information about Cortex-M23 processor architectural compliance, see the 2.7.1 Architecture and protocol information on page 19.

2.4 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.

2.5 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.

2.6 Configurable options

The following table shows the processor configurable options available at implementation time.

Table 2-1: Processor configurable options

Feature	Configurable option
Security Extension[2]	Present or absent
Non-secure MPU	4, 8, 12, 16 regions, or absent
Secure MPU	4, 8, 12, 16 regions, or absent
SAU[3]	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)[4]	• 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 (1 cycle) or slow (32 cycles)
Divider	Fast (21 cycles) or slow (37 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[5]	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[6]	0, 1, 2, 3, 4
Number of watchpoint comparators	0, 1, 2, 3, 4
Cross Trigger Interface (CTI)[6]	Present or absent
Micro Trace Buffer (MTB)[6]	Present or absent
Embedded Trace Macrocell (ETM)[6]	Present or absent

[2] There is also the possibility to add or remove the Security Extension with a fuse (input pin).
[3] Requires the Security Extension to be present.
[4] If the Security Extension is implemented, there is no option to have only one VTOR register. The options are either 0 or 2.

If the Security Extension is not implemented, there is no option to have two VTOR registers. The options are either 0 or 1.
[5] Instruction fetches and PPB accesses are always little-endian.
[6] Only when Halting debug support is present.

Feature	Configurable option
JTAGnSW debug protocol	Selects between JTAG or Serial-Wire interfaces for the DAP
Multi-drop support for Serial Wire[7]	Present or absent
Subordinate port support for AHB DAP	When set, include subordinate port support for any AHB DAP implementation. Otherwise, support only the low area DAP


Note

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

Cortex-M23 is written in Verilog-2001 and uses Verilog parameters for static configuration at instantiation/synthesis time. The following table lists additional parameters for functional safety, what block they are used in {Processor-level (P), DAP (D), Integration-level(I)}, their permitted values and the function of each permitted value.

Table 2-2: Safety configurations

Parameter	Default Value	Configure by Wire Tie-off and input signal	Description
BUSPROT (P,D, I)	0	No	Specifies whether interface protection is supported on the following interfaces. Interface protection provides parity bits to the bus interface to help with fault coverage in functional safety applications. • AHB Manager • I/O port • SBIST APB The options are: **0** Interface protection is excluded. **1** Interface protection is included.

[7] Requires Serial-Wire interfaces for DAP.

Parameter	Default Value	Configure by Wire Tie-off and input signal	Description
RAR (P, I)	0	No	Specifies whether all synchronous states or only architecturally required states are reset: **0** Only architecturally required state is reset. **1** All state is reset. **Note:** • When RAR is YES, all registers in the design can be reset, incurring an area penalty. • When RAR is NO, the registers in the design that do not require a reset have no reset.
FLOPPARITY (P,I)	0	No	Specifies whether the processor is configured with parity generation and checks on all flip-flops in non-debug logic. The options are: **1** Include parity on flip-flops **0** No parity on flip-flops **Note:** If FLOPPARITY is set to YES, then RAR must also be set to YES.

Parameter	Default Value	Configure by Wire Tie-off and input signal	Description
SBISTC (P,I)	0	No	Specifies whether Processor is configured to include STL hardware features - observation registers and whether MCU level is configured to include SBIST Controller. **0** STL harwdware features including observation registers and SBIST controller are not present **1** Present **Note:** The SBIST controller is used with the Software Test Library (STL) which is a separate licensable product from Arm.
SBIST_DL_CYCLES (I)	0xA0	No	Specifies the reset value SBIST deadlock counter. It can be any value from 0 to 0xFFFF. **Note:** Time out clock cycles = SBIST_DL_CYCLES * 32
SBIST_DL_RESET (I)	0	No	Sets FCTLT register at reset in SBIST controller: **0** Idle (Deadlock counter disabled) **1** Init (Deadlock counter counts down to zero) **Note:** • Must be set to NO for correct operation of the Execution Testbench. • The Software Test Library (STL) can set to YES for additional coverage.

Parameter	Default Value	Configure by Wire Tie-off and input signal	Description
SBIST_PSI (I)	0xF	No	Indicates the PS feature implemented. A bit set to 1 indicates the presence of the feature: **PSI[0]** Interrupt Generation **PSI[1]** IWICSENSE Mux **PSI[2]** EVENTBUS Mux **PSI[3]** AHB Subordinate **Note:** It is recommended to set this parameter to its default value, 0xF to get higher coverage when using STLs.

2.6.1 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 21-cycle iterative divider.
- As a 37-cycle iterative divider.

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

2.6.2 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.

2.7 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 1.4 Useful resources on page 9 for more information about the books that are described in 2.7 Product documentation, design flow, and architecture on page 17.

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.

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.

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.

2.7.1 Architecture and protocol information

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

- 2.7.2 Arm architecture on page 20.
- 2.7.3 Advanced Microcontroller Bus Architecture on page 20.
- 2.7.4 Debug Access Port architecture on page 20.

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

2.7.2 Arm architecture

The processor implements the Armv8-M baseline architecture profile.

For more information, see the *Arm®v8-M Architecture Reference Manual*.

2.7.3 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*.

2.7.4 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*.

2.8 Product revisions

This section describes the differences in functionality between product revisions.

r0p0
: First release.

r1p0
: Changed the ACTLR register bit assignments.

r2p0
: Addition of safety features- flop parity, interface protection and hardware features for STL.

3. Functional Description

Functional Description

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

3.1 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 3-1: Functional block diagram on page 22 shows the basic structure of the Grebe processor. An AHB subordinate interface is implemented at the Processor level which allows integration with a DAP at the next level up in the hierarchy.

Subcomponents with dashed outlines are optional, shaded components are configurable.

The MCU level is a reference design and can be modified by customers to suit their requirements. For more details on the interfaces present at the Processor and Integration levels.

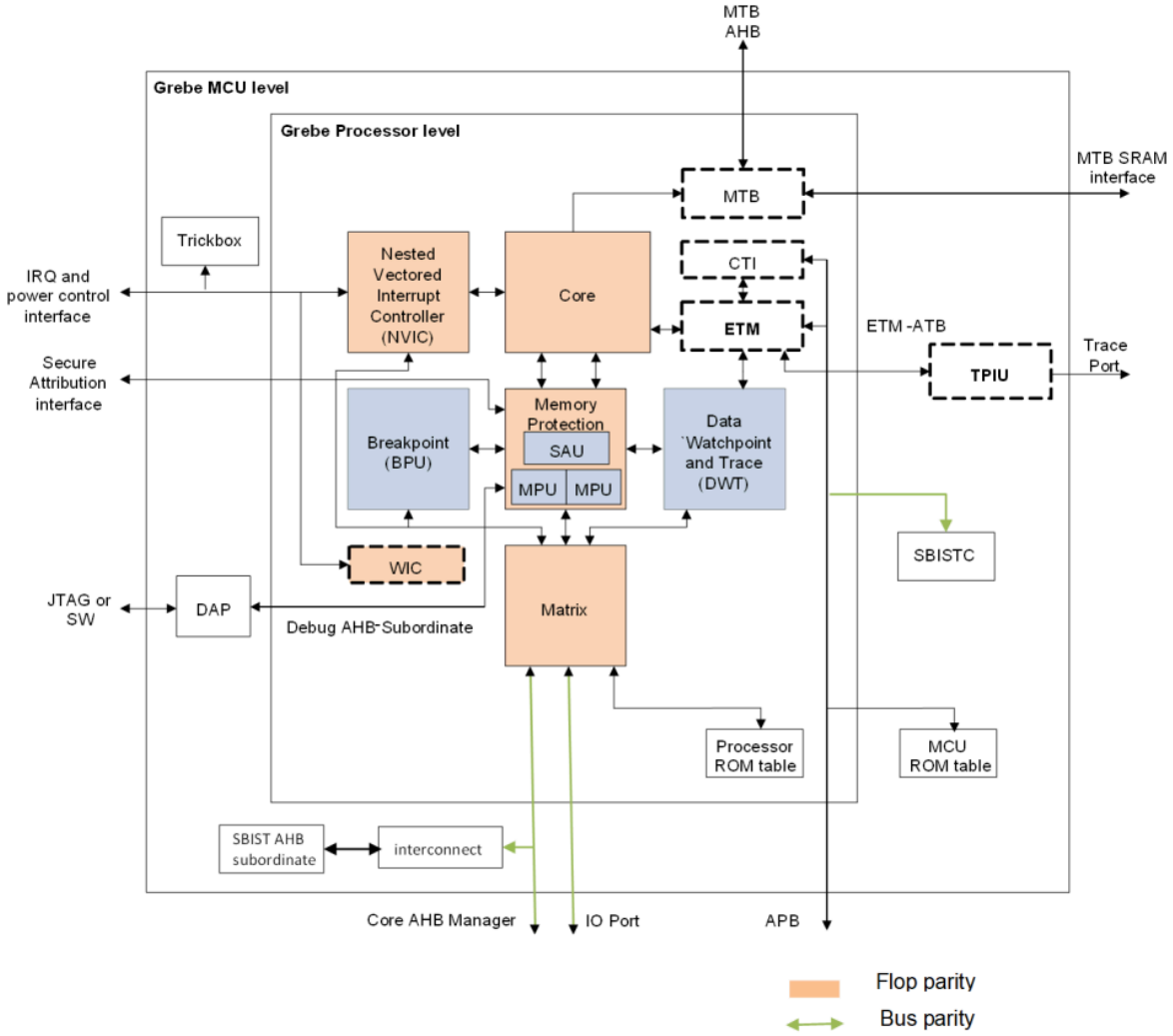
The following modules (in orange) include flop parity addition to the functional flops when processor is configured with FLOPPARITY:

- Core
- NVIC
- Memory Protection
- WIC
- bus Matrix
- Q-channel for power control

The interfaces in green include bus protection added to control and data signals:

- AHB Manager
- IOP
- SBISTC APB

Figure 3-1: Functional block diagram



The following table shows which blocks are safety related and which implement debug functionality and are therefore not safety related.

Table 3-1: Safety classification

Processor functionality	Debug functionality - Not safety related	MCU functionality	MCU debug functionality - Not safety related
Core (Fetch and DPU)	BPU	SBISTC	DAP
SAU	DWT	Trickbox	TPIU
MPU	ROM Table	Interconnect	MCU ROM Table
Matrix	CTI	AHB Subordinate	
NVIC	ETM		
WIC	MTB		

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 21-cycle divider, or a slower 37-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 subordinates 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 subordinate 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 completer.
- 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.

Functional Safety features

- Optional support for flop parity protection in the processor logic
- Optional interface protection included on the M-AHB, Debug-AHB and SBISTC-APB interfaces
- Hardware updates to support development of Software Test Libraries (STL)
- *FUSAEN* I/O for debug and trace logic protection

3.2 Interfaces

This section describes the external interface functions.

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

3.2.1 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.

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.

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 subordinate 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*.

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.

4. Programmers Model

Programmers Model

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

4.1 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:

- 5. System Control on page 37 summarizes the system control features of the programmers model.
- 6. Nested Vectored Interrupt Controller on page 44 summarizes the NVIC features of the programmers model.
- 7. Security Attribution and Memory Protection on page 46 summarizes the MPU and SAU features of the programmers model.
- 8. Debug on page 49 summarizes the Debug features of the programmers model.

4.2 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.

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.

Operating states

The processor can operate in Thumb or debug state:

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

Privileged access and 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.

4.3 Instruction set summary

The processor implements the entire Armv8-M baseline instruction set.

The following table 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 4-1: Cortex-M23 processor 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[8]	`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[9]	`ADDS <Rd>, <Rn>, <Rm>`	1
	Any to Any[8]	`ADD <Rd>, <Rd>, <Rm>`	1
	Any to PC[8]	`ADD PC, PC, <Rm>`	2
	8-bit immediate	`ADDS <Rd>, <Rd>, #<imm>`	1
	With carry	`ADCS <Rd>, <Rd>, <Rm>`	1

[8] Any indicates registers R0-R15.
[9] Lo indicates registers R0-R7.

Operation	Description	Assembler	Cycles
	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[9]	`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[10]
Divide	Unsigned	`UDIV {<Rd>,} <Rn>, <Rm>`	17 or 34
	Signed	`SDIV {<Rd>,} <Rn>, <Rm>`	17 or 34
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
	Compare and Branch on Zero	`CBZ <Rn>, <label>`	2 or 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
Clear	Exclusive	`CLREX`	1
Load	Word, immediate offset	`LDR <Rd>, [<Rn>, #<imm>]`	2 or 1
	Halfword, immediate offset	`LDRH <Rd>, [<Rn>, #<imm>]`	2 or 1[13]
	Byte, immediate offset	`LDRB <Rd>, [<Rn>, #<imm>]`	2 or 1[13]
	Word, register offset	`LDR <Rd>, [<Rn>, <Rm>]`	2 or 1[13]
	Halfword, register offset	`LDRH <Rd>, [<Rn>, <Rm>]`	2 or 1[13]

[10] Depends on multiplier implementation.
[11] Depends on divider implementation.
[12] 2 if the branch is taken, 1 if the branch is not taken.
[13] 2 if to AHB interface or SCS, 1 if to single-cycle I/O port.

Operation	Description	Assembler	Cycles
	Signed halfword, register offset	`LDRSH <Rd>, [<Rn>, <Rm>]`	2 or 1[13]
	Byte, register offset	`LDRB <Rd>, [<Rn>, <Rm>]`	2 or 1[13]
	Signed byte, register offset	`LDRSB <Rd>, [<Rn>, <Rm>]`	2 or 1[13]
	PC-relative	`LDR <Rd>, <label>`	2 or 1[13]
	SP-relative	`LDR <Rd>, [SP, #<imm>]`	2 or 1[13]
	Multiple, excluding base	`LDM <Rn>!, {<loreglist>}`	1+N
	Multiple, including base	`LDM <Rn>, {<loreglist>}`	3+N[15]
	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
	Acquire Halfword	`LDAH Rt, [<Rn>]`	3 or 2[16]
	Acquire Byte	`LDAB Rt, [<Rn>]`	3 or 2[16]
	Acquire Exclusive Word	`LDAEX Rt, [<Rn>]`	4
	Acquire Exclusive Halfword	`LDAEXH Rt, [<Rn>]`	4
	Acquire Exclusive Byte	`LDAEXB Rt, [<Rn>]`	4
Store	Word, immediate offset	`STR <Rd>, [<Rn>, #<imm>]`	2 or 1[13]
	Halfword, immediate offset	`STRH <Rd>, [<Rn>, #<imm>]`	2 or 1[13]
	Byte, immediate offset	`STRB <Rd>, [<Rn>, #<imm>]`	2 or 1[13]
	Word, register offset	`STR <Rd>, [<Rn>, <Rm>]`	2 or 1[13]
	Halfword, register offset	`STRH <Rd>, [<Rn>, <Rm>]`	2 or 1[13]
	Byte, register offset	`STRB <Rd>, [<Rn>, <Rm>]`	2 or 1[13]
	SP-relative	`STR <Rd>, [SP, #<imm>]`	2 or 1[13]
	Multiple	`STM <Rn>!, {<loreglist>}`	1+N
	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[16]
	Acquire Halfword	`STLH Rt, [<Rn>]`	3 or 2[16]
	Acquire Byte	`STLB Rt, [<Rn>]`	3 or 2[16]
	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>}`	1+N[15]

[14] loreglist indicates registers R0-R7.
[15] N is the number of elements in the list.
[16] 3 if to AHB interface or SCS, 2 if to single-cycle I/O port.

Operation	Description	Assembler	Cycles
	Push with link register	`PUSH {<loreglist>, LR}`	1+N[17]
Pop	Pop	`POP {<loreglist>}`	1+N[15]
	Pop and return	`POP {<loreglist>, PC}`	3+N
	Pop and function return	`POP {<loreglist>, PC}`	3+N[17]
Branch	Conditional	`B<c> <label>`	1 or 2[18]
	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
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>`	-
	Disable interrupts	`CPSID i`	1
	Enable interrupts	`CPSIE i`	1
	Read special register	`MRS <Rd>, <specreg>`	3
	Write special register	`MSR <specreg>, <Rn>`	3
	Breakpoint	`BKPT #<imm>`	-
Hint	Send event	`SEV`	1
	Wait for event	`WFE`	2
	Wait for interrupt	`WFI`	2
	Yield	`YIELD`	1[22]
	No operation	`NOP`	1
Barriers	Instruction synchronization	`ISB`	3
	Data memory	`DMB`	3
	Data synchronization	`DSB`	3

[17] N is the number of elements in the list including PC or LR.
[18] 2 if taken, 1 if not-taken.
[19] specreg indicates special-purpose registers
[20] Cycle count depends on processor and debug configuration.
[21] Excludes time that is spent waiting for an interrupt or event.
[22] Executes as NOP.

Operation	Description	Assembler	Cycles
Gateway	Secure Gateway	`SG`	3
Test	Test Target	`TT{A}{T} <Rd>, <Rn>`	3

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

4.4 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 4-2: Default memory map on page 32 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 4-2: 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	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[23]	On chip RAM. Supports code.
`0x40000000 - 0x5FFFFFFF`	Peripheral	Device	-	AMBA 5 AHB port or I/O port[23]	On chip peripherals. XN
`0x60000000 - 0x9FFFFFFF`	RAM	Normal	AMBA 5 AHB port	AMBA 5 AHB port or I/O port[23]	Supports code.
`0xA0000000 - 0xDFFFFFFF`	Device	Device	-	AMBA 5 AHB port or I/O port[23]	XN.

[23] Only when the I/O port is included.

Address region	Region name	Memory type	Instruction accesses	Data accesses	Description
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[23]	MTB, ETM, CTI, TPIU configuration registers when implemented. XN.
0xE0050000 - 0xE00EFFFF	PPB	-	-	Internal PPB interface	Reserved. XN.
0xE00F0000 - 0xE00FFFFF	Device	Device	AMBA 5 AHB port	AMBA 5 AHB port or I/O port[23]	Cortex-M23 MCU ROM when implemented. XN.
0xE0100000 - 0xFFFFFFFF	Vendor_SYS	Device	-	AMBA 5 AHB port or I/O port[23]	Suitable for CoreSight ROM tables and other CoreSight components. XN.

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 7. Security Attribution and Memory Protection on page 46.
- 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 7. Security Attribution and Memory Protection on page 46.

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

4.5 Registers summary

The following table shows the processor register set summary. Each of these registers is 32 bits wide.

Table 4-3: Core register set summary

Name	Description
R0-R12	R0-R12 are general-purpose registers for data operations.

Name	Description
MSP (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.
PSP (R13)	
MSPLIM_S	The stack limit registers limit the extent to which the MSP_S and PSP_S registers can descend respectively.
PSPLIM_S	
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 4.6 Exceptions on page 35. **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.

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

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


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.

4.6 Exceptions

This section describes the exception model of the processor.

4.6.1 Exception handling

The processor implements advanced exception and interrupt handling.

This process is 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 21-cycle and 37-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 25 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.

5. System Control

System Control

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

5.1 About system control

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

5.2 System control register summary

Table Table 5-1: System control registers on page 37 gives the system control registers. Each of these registers is 32 bits wide.

Table 5-1: 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. **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. [25] **If the Security Extension is not implemented:** There is one SYST_RVR register in the Cortex-M23 processor.

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

Name	Description
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. [25] **If the Security Extension is not implemented:** There is one SYST_CVR register in the Cortex-M23 processor.
SYST_CALIB	*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. [25] **If the Security Extension is not implemented:** There is one SYST_CALIB register in the Cortex-M23 processor.
CPUID	See 5.2.2 CPUID Register on page 40.
ICSR	*Interrupt Control and State Register*, see the *Arm®v8-M Architecture Reference Manual*.
AIRCR[27]	*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*.
SHCSR	*System Handler Control and State Register*, see the *Arm®v8-M Architecture Reference Manual*.
VTOR[28]	*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:[29] • 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 5.2.1 ACTLR Register on page 39.

[26] This value is configured by the implementer during implementation. See the documentation that is supplied by your vendor for more information.
[27] This value is configured by the implementer during implementation. See the documentation that is supplied by your vendor for more information.
[28] 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.
[29] 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 5-1: System control registers on page 37.

5.2.1 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 5-2: ACTLR bit register assignments on page 39.

Figure 5-1: ACTLR bit register assignments on page 39 shows the ACTLR bit register assignments.

Figure 5-1: ACTLR bit register assignments

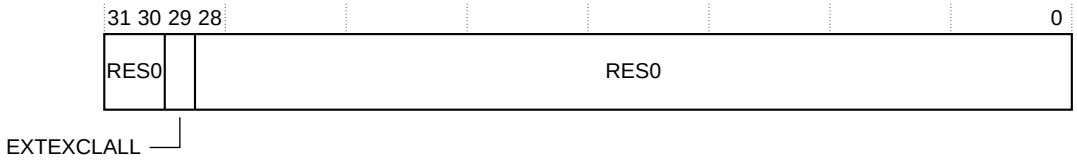


Table 5-2: ACTLR bit register assignments on page 39 shows the ACTLR register bit assignments.

Table 5-2: ACTLR bit register assignments

Bits	Field	Function
[31:30]	-	**RES0**

Bits	Field	Function
[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**

5.2.2 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 5.2 System control register summary on page 37.

Figure 5-2: CPUID bit register assignments on page 40 shows the CPUID bit register assignments.

Figure 5-2: CPUID bit register assignments

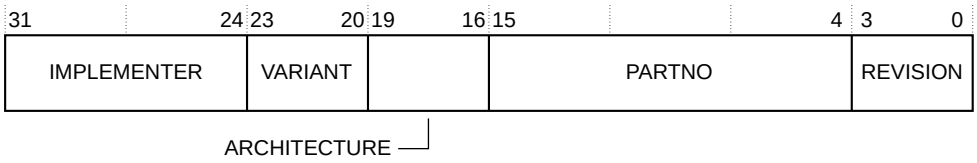


Table 5-3: CPUID bit register assignments on page 41 shows the CPUID register bit assignments.

Table 5-3: CPUID bit register assignments

Bits	Field	Function
[31:24]	IMPLEMENTER	Implementer code: **0x41** Arm.
[23:20]	VARIANT	Major revision number *n* in the *rnpm* revision status. See 1.1 Product revision status on page 7: 0x2.
[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 *rnpm* revision status. See 1.1 Product revision status on page 7. 0x0.

5.2.3 STL registers

The processor includes a number of registers which can be used by the Software Test Library, STL, to observe the internal state of the NVIC priority tree outputs and to sample the MPU region hit and associated attributes when a MemManage fault occurs on an instruction fetch or data access based on a programmable address.

Usage constraints and attributes

- The STL observation registers are only available when processor is configured with SBISTC = 1
- The STL observation registers are not banked between security states
- If the Security Extension is implemented these registers are RAZ/WI from Non-secure state
- Unprivileged access will result in a BusFault exception

NVIC Observation Registers: STLNVICPENDOR and STLNVICACTVOR

The STLNVICPEN and STLNVICACTV registers can be used to observe the current output state of the NVIC pending and active priority tree which represents the highest priority pended or active interrupt at the point that the register is read. Both registers are read-only and reset to 0x00000000. The format of the registers is specified in Table 25.

Table 5-4: STLNVICPENDOR and STLNVICACTVOR observation registers

Bits	Field	Function
[31:19]	-	RES0
[18]	VALID	Priority tree output is valid

Bits	Field	Function
[17]	TARGET	Exception Security target • 0 = Secure • 1 = Non-Secure
[16:15]	PRIORITY	Exception priority **Note:** PRIORITY = 0 for exceptions with fixed priority in INTNUM
[14:8]	-	RES0
[7:0]	INTNUM	Exception number as defined in [1]: **16 > INTNUM ≥ 0** Armv8-M exceptions **INTNUM ≥ 16** IRQ

MPU Observation Registers: STLIDMPUSR, STLAMPUOR, STLBMPUOR

The STLAMPUOR and STLBMPUOR registers can be used to observe the MPU region hit, and memory attributes associated with a MemManage fault on an instruction fetch or data access based on the address specified in MPU sample register STLIDMPUSR.

The processor includes two MPU ports (A and B). Port A is for data while port B is for instructions.

- All the registers are reset to `0x00000000`.
- STLAMPUOR and STLBMPUOR are reset to `0x00000000` when the STLIDMPUSR register is updated.
- STLAMPUOR and STLBMPUOR will be updated independently if a fault is detected on the associated MPU if the associated selection fields in the STLIDMPUSR register is set, e.g. If the sample register is configured to select the data MPU, DATA = 0b1, then an access will be captured in the appropriate observation register STLAMPUOR.

STLIDMPUSR is specified in Table 5-5: STLIDMPUSR observation sample register on page 42 and STLAMPUOR and STLBMPUOR is specified in Table 5-6: STLAMPUOR and STLBMPUOR observation registers on page 43.

Table 5-5: STLIDMPUSR observation sample register

Bits	Field	Function
[31:5]	ADDR	Sample Address
[4:3]	-	RES0
[2]	INSTR	Select Instruction MPU
[1]	DATA	Select Data MPU
[0]	-	RES0

Table 5-6: STLAMPUOR and STLBMPUOR observation registers

Bits	Field	Function
[31:17]	-	RES0
[16:9]	HITREGION	MPU region hit for data STLAMPUOR **Note:** • HITREGION range depends on the processor security state and MPU configuration in the Verilog parameters MPU_S and MPU_NS • HITREGION[7:4] is RAZ This field is RAZ for STLBMPUOR
[8:6]	-	RES0
[5:0]	ATTR	Memory attributes: ATTR[5]: Shareability ATTR[4:0] : Attributes

6. Nested Vectored Interrupt Controller

Nested Vectored Interrupt Controller

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

6.1 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 4.6 Exceptions on page 35.

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.

6.2 NVIC register summary

The following table shows the NVIC registers. Each of these registers is 32 bits wide.

Table 6-1: NVIC registers

Name	Description
NVIC_ISERn	*Interrupt Set-Enable Register n.*
NVIC_ICTR	*Interrupt Controller Type Register*
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[30]	*Interrupt Target Non-Secure Register n.*
NVIC_IPRn	*Interrupt Priority Registers n.*


Note

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

[30] Present only when the Security Extension is implemented.

7. Security Attribution and Memory Protection

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:

7.1 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 7-1: 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.

7.2 SAU register summary

The following table shows the SAU registers. Each of these registers is 32 bits wide.

Table 7-2: SAU registers

Name	Reset value	Description
SAU_CTRL	`0x00000000`[31]	*Security Attribution Unit Control Register*.
SAU_TYPE[32]	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*.

[31] The reset value is `0x00000002` if the Security Extension is not implemented (write ignored).
[32] Bits[7:0] depend on the number of SAU regions included. This value can be 0, 4, or 8.

Name	Reset value	Description
SAU_RLAR	0x00000000[33]	*Security Attribution Unit Region Limit Address Register*.

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

7.3 MPU register summary

The following table 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 7-3: MPU registers

Name	Reset value	Description
MPU_TYPE[34]	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

- 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.

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

8. Debug

Debug

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

8.1 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.

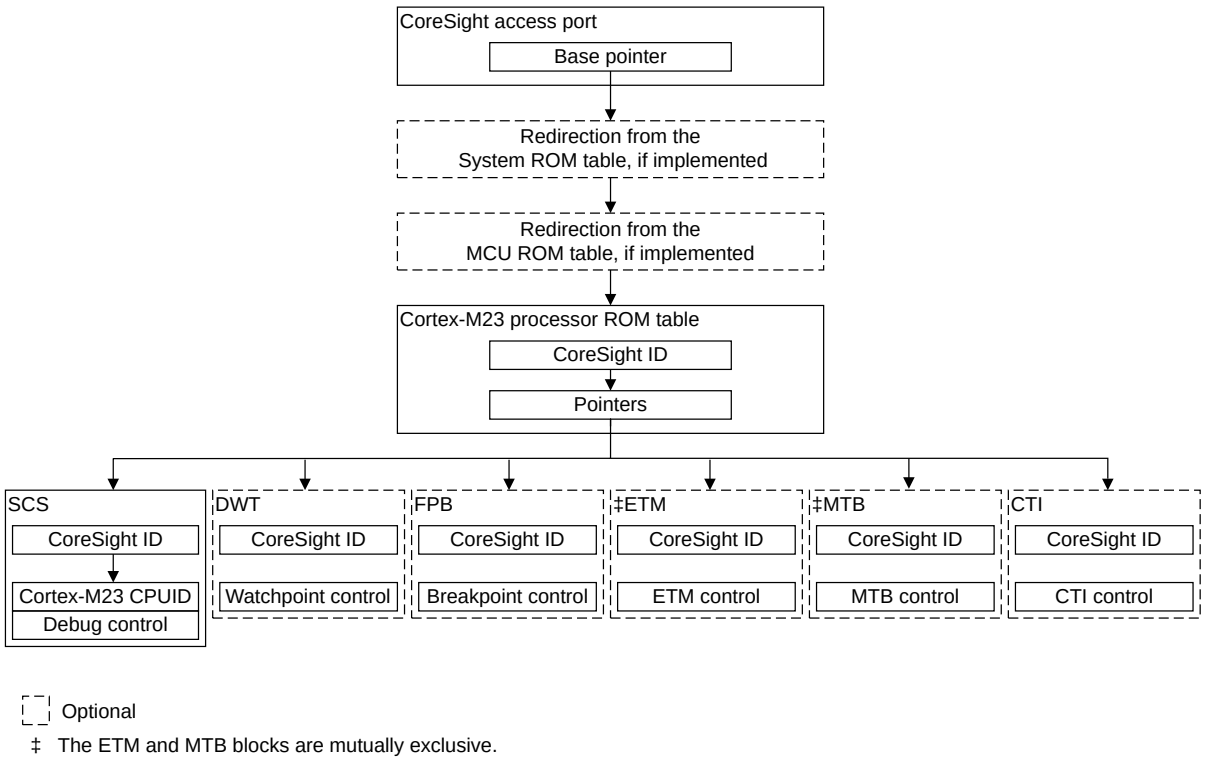
The processor implementation can be partitioned to place the debug components in a separate power domain from the processor core and NVIC.

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 8-1: CoreSight discovery on page 50. In this example, a debugger reads the peripheral and component ID registers for each CoreSight component in the CoreSight system.

Figure 8-1: 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 Table 8-2: Cortex-M23 processor ROM table identification values on page 52.
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 Table 8-1: Cortex-M23 processor ROM table components on page 51.

   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.

8.1.1 Cortex-M23 processor ROM table identification and entries

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

Table 8-1: Cortex-M23 processor ROM table components

Address	Component	Value	Description
`0xE00FF000`	SCS	`0xFFF0F003`	See 8.1.2 System Control Space on page 52.
`0xE00FF004`	DWT	`0xFFF02003`[35]	See 8.1.4 Data watchpoint unit on page 53.
`0xE00FF008`	FPB	`0xFFF03003`[36]	See 8.1.6 Flash Patch and Breakpoint unit on page 54.
`0xE00FF00C`	Reserved	`0xFFF01002`	-
`0xE00FF010`	Reserved	`0xFFF41002`	-
`0xE00FF014`	ETM	`0xFFF42003`[37]	See 8.2 Embedded Trace Macrocell on page 54.
0xE00FF018	CTI	`0xFFF43003`[38]	See 8.4 Cross Trigger Interface on page 55.
0xE00FF01C	MTB	`0xFFF44003`[39]	See 8.3 Micro Trace Buffer on page 55.
`0xE00FF020`	End marker	`0x00000000`	See the *Arm®v8-M Architecture Reference Manual*.

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`.

The following table 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.

[35] Reads as `0xFFF02002` if no watchpoints are implemented.
[36] Reads as `0xFFF03002` if no breakpoints are implemented.
[37] Reads as `0xFFF42002` if ETM is not implemented.
[38] Reads as `0xFFF43002` if CTI is not implemented.
[39] Reads as `0xFFF44002` if MTB is not implemented.

Table 8-2: 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.

8.1.2 System Control Space

If debug is implemented, the processor provides debug through registers in the SCS, see 8.6 Debug register summary on page 56.

8.1.3 SCS CoreSight identification

Table 8-3: SCS identification values on page 52 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 5.2.2 CPUID Register on page 40.

Table 8-3: 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	

Address	Register	Value	Description
0xE000EFFC	Component ID3	0x000000B1	See the *Arm®v8-M Architecture Reference Manual*.

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.

8.1.4 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*.

8.1.5 DWT CoreSight identification

The following table shows the DWT identification registers and values for debugger detection.

Table 8-4: 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.

8.1.6 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*.

8.1.6.1 FPB CoreSight identification

The following table shows the FPB identification registers and their values for debugger detection.

Table 8-5: 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.

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

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

8.4 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.

8.5 CTI register summary

The following table shows the CTI identification registers and values for debugger detection.

Table 8-6: CTI identification values

Address	Register	Value	Description
0xE0042FD0	PIDR4	0x00000004	Peripheral ID4
0xE0042FE0	PIDR0	0x00000020	Peripheral ID0
0xE0042FE4	PIDR1	0x000000BD	Peripheral ID1

Address	Register	Value	Description
0xE0042FE8	PIDR2	0x0000000B	Peripheral ID2
0xE0042FEC	PIDR3	0x00000000	Peripheral ID3
0xE0042FD4	PIDR5	0x00000000	Peripheral ID5
0xE0042FD8	PIDR6	0x00000000	Peripheral ID6
0xE0042FDC	PIDR7	0x00000000	Peripheral ID7
0xE0042FF0	CIDR0	0x0000000D	Component ID0
0xE0042FF4	CIDR1	0x00000090	Component ID1
0xE0042FF8	CIDR2	0x00000005	Component ID2
0xE0042FFC	CIDR3	0x000000B1	Component ID3
0xE0042000	CTICONTROL	0x00000000	CTI Control Register
0xE0042010	CTIINTACK	UNKNOWN	CTI Interrupt Acknowledge Register
0xE0042014	CTIAPPSET	0x00000000	CTI Application Trigger Set Register
0xE0042018	CTIAPPCLEAR	0x00000000	CTI Application Trigger Clear Register
0xE004201C	CTIAPPPULSE	UNKNOWN	CTI Application Pulse Register
0xE0042130	CTITRIGINSTATUS	0x00000000	CTI Trigger In Status Register
0xE0042134	CTITRIGOUTSTATUS	0x00000000	CTI Trigger Out Status Register
0xE0042138	CTICHINSTATUS	0x00000000	CTI Channel In Status Register
0xE0042140	CTIGATE	0x0000000F	Enable CTI Channel Gate register
0xE0042144	ASICCTL	0x00000000	External Multiplexer Control register
0xE0042F00	ITCTRL	0x00000000	Integration Mode Control register
0xE0042FC8	DEVID	0x00040800	Device Configuration register
0xE0042FBC	DEVARCH	0x47701A14	Device Architecture register
0xE0042FCC	DEVTYPE	0x00000014	Device Type Identifier register

Note

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

8.6 Debug register summary

The following table shows the debug registers. Each of these registers is 32 bits wide.

Table 8-7: 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*.

The following table shows the FPB registers. Each of these registers is 32 bits wide.

Table 8-8: 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	

The following table shows the DWT registers. Each of these registers is 32 bits wide.

Table 8-9: 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*.
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 *CoreSight™ ETM-M23 Technical Reference Manual* for information about the ETM registers.

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

The following table shows the CTI registers. Each of these registers is 32 bits wide.

Table 8-10: CTI register summary

Name	Description
CTICONTROL	See the *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	4.6.1 Exception handling on page 35	r1p0
Added the description of the ACTLR Register	5.2.1 ACTLR Register on page 39	r1p0
Updated the major revision number in the CPUID register	5.2.2 CPUID Register on page 40	r1p0
Updated the Security Attribution and Memory Protection section	7.1 About Security Attribution and Memory Protection on page 46	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	4.5 Registers summary on page 33	r1p0
Updated the ACTLR.EXTEXCLALL description	5.2.1 ACTLR Register on page 39	r1p0

Table A-4: Differences between issue C and D

Change	Location	Affects
First release for r2p0	-	-
Added Parity checking conditions.	2.6 Configurable options on page 12	r2p0
Updated the divider cycle numbers.	2.6.1 Configurable divider on page 174.6.1 Exception handling on page 35	r2p0
Updated the Functional block diagram and updated the section with the additional functions.	3.1 About the functions on page 21	r2p0
Updated the interrupt latency cycle number.	4.6.1 Exception handling on page 35	r2p0
Added STL registers section	5.2.3 STL registers on page 41	r2p0