Arm® Cortex®-M33 Processor

Revision: r1p0

Technical Reference Manual

Non-Confidential

Issue 07

100230_0100_07_en

Arm® Cortex®-M33 Processor

Technical Reference Manual

Release Information

Document history

Issue	Date	Confidentiality	Change
0000-00	28 September 2016	Confidential	First release for r0p0
0001-00	3 February 2017	Confidential	First release for r0p1
0002-00	10 May 2017	Non-Confidential	First release for r0p2
0003-00	6 December 2017	Non-Confidential	First release for r0p3
0004-00	18 April 2018	Non-Confidential	First release for r0p4
0100-03	19 June 2020	Non-Confidential	First release for r1p0
0100-07	15 January 2023	Non-Confidential	Second release for r1p0

Proprietary Notice

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. Preface

1.1 Product revision status

The `rxpy` identifier indicates the revision status of the product described in this manual, for example, `r1p2`, where:

rx Identifies the major revision of the product, for example, r1.

py Identifies the minor revision or modification status of the product, for example, p2.

1.2 Intended audience

This manual is written to help system designers, system integrators, verification engineers, and software programmers who are implementing a *System on Chip* (SoC) device based on the Cortex®-M33 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.

Convention	Use
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Caution

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

Warning

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

Danger

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

Note

An important piece of information that needs your attention.

Tip

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

Remember

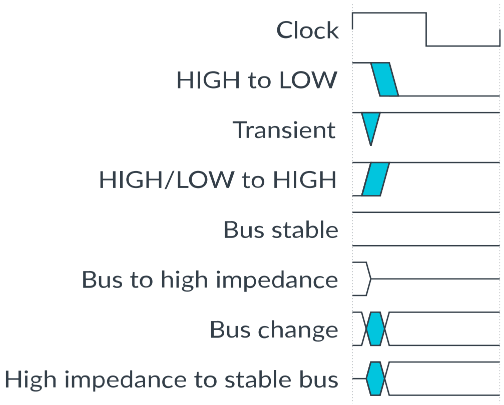
A reminder of something important that relates to the information you are reading.

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.

Arm product resources	Document ID	Confidentiality
Arm® Cortex®-M33 Processor STL User Guide	101170	Confidential
Arm® Cortex®-M33 Processor Integration and Implementation Manual	100323	Confidential

Arm architecture and specifications	Document ID	Confidentiality
Arm®v8-M Architecture Reference Manual	DDI 0553	Non-Confidential
Arm® AMBA® 5 AHB Protocol Specification	IHI 0033	Non-Confidential
AMBA® APB Protocol Version 2.0 Specification	IHI 0024	Non-Confidential
AMBA® ATB Protocol Specification	IHI 0032	Non-Confidential
CoreSight™ Components Technical Reference Manual	DDI 0314	Non-Confidential
Lazy Stacking and Context Switching Application Note 298	DAI0298	Non-Confidential
AMBA® Low Power Interface Specification	IHI 0068	Non-Confidential
Arm® Embedded Trace Macrocell Architecture Specification ETMv4	IHI 0064	Non-Confidential
Arm® CoreSight™ Architecture Specification v2.0	IHI 0029	Non-Confidential
Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2	IHI 0031	Non-Confidential

Non-Arm resources	Document ID	Organization
IEEE Std 1149.1-2001, Test Access Port and Boundary-Scan Architecture (JTAG)	IEEE Std 1149.1-2001	https://www.ieee.org
IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic	IEEE 754-2008	https://www.ieee.org

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. Introduction

This chapter introduces the Cortex®-M33 processor and its features, configurable options, and product documentation.

Note

A Cortex®-M33 *Debug Access Port* (DAP) and a Cortex®-M33 *Trace Port Interface Unit* (TPIU), which form part of an example system, are included in the Cortex®-M33 processor deliverables. As a result, your implementation might include a Cortex®-M33 DAP and a Cortex®-M33 TPIU.

2.1 About the processor

The Cortex®-M33 processor is a low gate count, highly energy efficient processor that is intended for microcontroller and deeply embedded applications. The processor is based on the Arm®v8-M architecture and is primarily for use in environments where security is an important consideration.

The interfaces that the processor supports include:

- *Code AHB* (C-AHB) interface.
- *System AHB* (S-AHB) interface.
- *External PPB* (EPPB) APB interface.
- *Debug AHB* (D-AHB) interface.

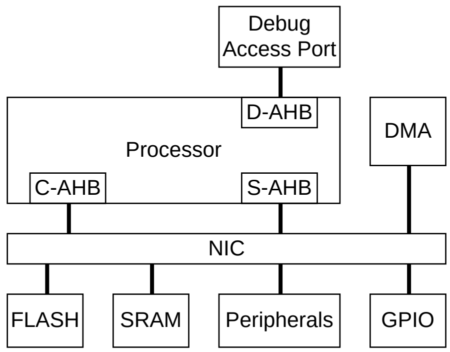
The processor has optional:

- Arm® TrustZone® technology, using the Arm®v8-M Security Extension supporting Secure and Non-secure states.
- *Memory Protection Units* (MPUs), which you can configure to protect regions of memory.
- Floating-point arithmetic functionality with support for single precision arithmetic.
- Support for ETM and MTB trace.

The processor is highly configurable and is intended for a wide range of high-performance, deeply embedded applications that require fast interrupt response features.

The following figure shows the processor in a typical system.

Figure 2-1: Example processor system



2.2 About the processor architecture

The processor implements the Arm®v8-M architecture with the Main Extension.

The processor has optional support for each of the following extensions:

- The Security Extension.
- The Floating-point Extension.
- The *Digital Signal Processing* (DSP) Extension.
- The Debug Extension.
- The *Custom Datapath Extension* (CDE).

The processor includes the following features:

- An in-order issue pipeline.
- Thumb-2 technology. See the *Arm®v8-M Architecture Reference Manual*.
- Data accesses performed as either big or little endian.
- A *Nested Vectored Interrupt Controller* (NVIC) closely integrated with the processor with up to 480 interrupts.
- An optional *Floating Point Unit* (FPU) supporting single-precision arithmetic.
- Support for exception-continuable instructions, such as `LDM`, `LDMDB`, `STM`, `STMDB`, `PUSH`, and `POP`. If the processor supports FPU, the `VLDM`, `VSTM`, `VPUSH`, `VPOP` exception-continuable instructions are also included.
- A low-cost debug solution with the optional ability to:
  - Implement breakpoints.
  - Implement watchpoints, tracing, and system profiling.
  - Support `printf()` style debugging through an *Instrumentation Trace Macrocell* (ITM).
- Support for the instruction trace option:

    - *Embedded Trace Macrocell* (ETM). See the *Arm® CoreSight™ ETM-M33 Technical Reference Manual* for more information.
- Optional coprocessor interface for external hardware accelerators.
- Support for the *Custom Datapath Extension* (CDE) which adds classes of *Arm Custom Instructions* (ACIs) in the coprocessor instruction space.
- Low-power features including architectural clock gating, sleep mode, and a power aware system with optional *Wake-up Interrupt Controller* (WIC).
- A memory system, which can include optional memory protection and security attribution.

2.3 Processor configuration options

The Cortex®-M33 processor has configurable options that you can set during the implementation and integration stages to match your functional requirements.

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

Feature	Options
Floating-point	No floating-point.
	Single-precision floating-point only.
DSP Extension	No Arm®v8-M DSP Extension.
	Arm®v8-M DSP Extension supported, including the following instruction classes: • Pack halfword. • Saturating. • Arithmetic. • Reverse bits/bytes. • Select bytes. • Sign-extend. • Sum of absolute differences. • SIMD arithmetic. • Extended signed multiplies with overflow detection. • Extended signed multiplies with optional rounding. • SIMD multiplies with overflow detection. • Extended unsigned multiply.
Security Extension	No Arm®v8-M Security Extension.
	Arm®v8-M Security Extension.
Non-secure protected memory regions	0 region, 4 regions, 8 regions, 12 regions, or 16 regions.
Secure protected memory regions	0 region, 4 regions, 8 regions, 12 regions, or 16 regions when the Arm®v8-M Security Extension is included.
Security Attribution Unit (SAU)	0 region, 4 regions, or 8 regions when the Arm®v8-M Security Extension is included.

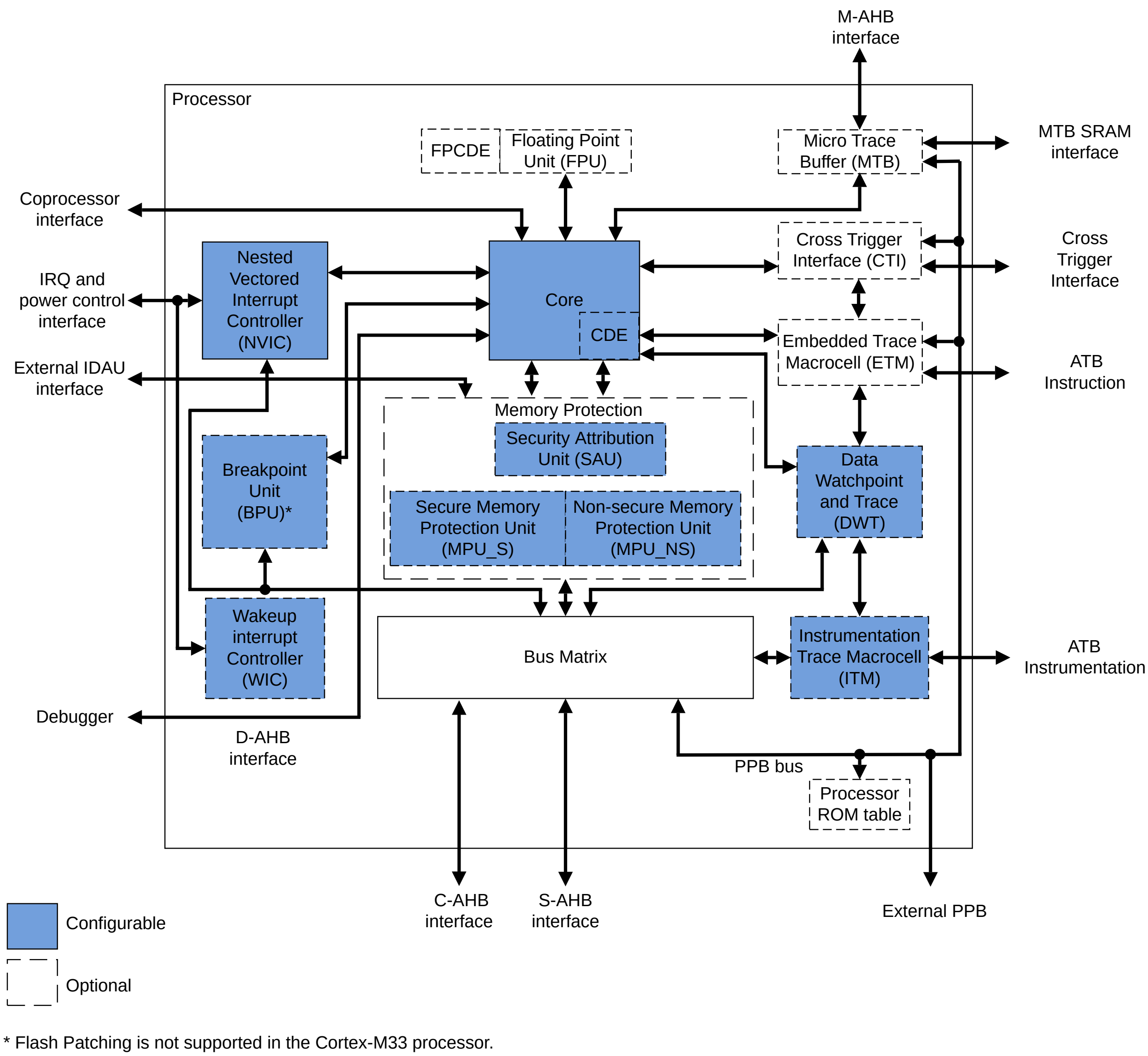
Feature	Options
Interrupts	1-480 interrupts. To support non-contiguous mapping, you can remove individual interrupts.
Number of bits of interrupt priority	Between three and eight bits of interrupt priority, between 8 and 256 levels of priority implemented.
Debug watchpoints and breakpoints	Minimal debug. No Halting debug or memory and peripheral access.
	Reduced set. Two data watchpoint comparators and four breakpoint comparators.
	Full set. Four data watchpoint comparators and eight breakpoint comparators.
ITM and *Data Watchpoint and Trace* (DWT) trace functionality	No ITM or DWT trace.
	Complete ITM and DWT trace.
Embedded Trace Macrocell (ETM)	No ETM support.
	ETM instruction execution trace.
Micro Trace Buffer (MTB)	No MTB support.
	MTB instruction trace.
Cross Trigger Interface (CTI)	No CTI.
	CTI included.
Wake-up Interrupt Controller (WIC)	No WIC controller.
	WIC controller included.
External coprocessor interface	No support for coprocessor hardware.
	Support for coprocessor hardware.
Arm Custom Instructions (ACIs) with *Custom Datapath Extension* (CDE) modules on a coprocessor basis	The coprocessor executes instructions and the CDE modules are not used.
	The CDE module executes instructions and the coprocessor is bypassed.

2.4 Component blocks

The processor has fixed and optional component blocks.

The following figure shows the optional and fixed components of the processor.

Figure 2-2: Functional block diagram



Note

- The MPU_NS, WIC, CTI, and FPU are always optional.
- If the processor is configured with minimal debug, the ETM, MTB, and ITM cannot be included.
- If the processor is configured with reduced set or full set debug, the ETM, MTB, and ITM are optional.
- If the processor is configured with the reduced set or the full set debug, the BPU and DWT are always included.
- The MPU_S is optional if the Security Extension is present.
- The SAU is included if the Security Extension is present.

2.4.1 Processor core

The processor core provides:

- Limited dual-issue of common 16-bit instruction pairs.
- Integer divide unit with support for operand-dependent early termination.
- Support for interrupted continuable load and store multiple operations.
- Load and store operations that both support precise bus errors.

To support *Arm Custom Instructions* (ACIs), the processor core includes an optional CDE module. This module is used to execute user-defined instructions that work on general-purpose registers. See 9. Arm Custom Instructions on page 60 for more information.

2.4.2 Security attribution and memory protection

The Cortex®-M33 processor supports the Arm®v8-M *Protected Memory System Architecture* (PMSA) that provides programmable support for memory protection using a number of software controllable regions.

Memory regions can be programmed to generate faults when accessed inappropriately by unprivileged software reducing the scope of incorrectly written application code. The architecture includes fault status registers to allow an exception handler to determine the source of the fault and to apply corrective action or notify the system.

The Cortex®-M33 processor also includes optional support for defining memory regions as *Secure* or *Non-secure*, as defined in the Arm®v8-M Security Extension, and protecting the regions from accesses with an inappropriate level of security.

Related information

Security Attribution and Memory Protection on page 45

2.4.3 Floating-Point Unit

The FPU provides:

- Instructions for single-precision (C programming language `float` type) data-processing operations.
- Instructions for double-precision (C `double` type) load and store operations.
- Combined multiply-add instructions for increased precision (Fused MAC).
- Hardware support for conversion, addition, subtraction, multiplication with optional accumulate, division, and square-root.
- Hardware support for denormals and all IEEE Standard 754-2008 rounding modes.
- 32 32-bit single-precision registers or 16 64-bit double-precision registers.

- Lazy floating-point context save. Automated stacking of floating-point state is delayed until the ISR attempts to execute a floating-point instruction. This reduces the latency to enter the ISR and removes floating-point context save for ISRs that do not use floating-point.

To support *Arm Custom Instructions* (ACIs), the FPU includes a floating-point CDE module. This module is used to execute user-defined instructions that work on floating-point registers. If the optional FPU is not present, then the optional floating-point CDE module is not present either. See 9. Arm Custom Instructions on page 60 for more information.

Related information

Floating-Point Unit on page 52

2.4.4 Nested Vectored Interrupt Controller

The *Nested Vectored Interrupt Controller* (NVIC) is closely integrated with the core to achieve low-latency interrupt processing.

Functions of the NVIC include:

- External interrupts, configurable from 1 to 480 using a contiguous or non-contiguous mapping. This is configured at implementation.
- Configurable levels of interrupt priority from 8 to 256. This is configured at implementation.
- Dynamic reprioritization of interrupts.
- Priority grouping. This enables selection of preempting interrupt levels and non-preempting interrupt levels.
- Support for tail-chaining and late arrival of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
- Optional support for the Arm®v8-M Security extension. Secure interrupts can be prioritized above any Non-secure interrupt.

Related information

Nested Vectored Interrupt Controller on page 49

2.4.5 Cross Trigger Interface Unit

The optional CTI enables the debug logic, MTB, and ETM to interact with each other and with other CoreSight™ components.

Related information

Cross Trigger Interface on page 81

2.4.6 ETM

The optional ETM provides instruction-only capabilities when configured.

See the *Arm® CoreSight™ ETM-M33 Technical Reference Manual* for more information.

Related information

Useful resources for M33 Processor TRM - referable content

2.4.7 MTB

The MTB provides a simple low-cost execution trace solution for the Cortex®-M33 processor.

Trace is written to an SRAM interface, and can be extracted using a dedicated AHB slave interface (M-AHB) on the processor. The MTB can be controlled by memory mapped registers in the PPB region or by events generated by the DWT or through the CTI.

See the *Arm® CoreSight™ MTB-M33 Technical Reference Manual* for more information.

2.4.8 Debug and trace

Debug and trace components include a configurable *Breakpoint Unit* (BPU) for implementing breakpoints, and configurable *Data Watchpoint and Trace* (DWT) unit for implementing watchpoints, data tracing, and system profiling.

Other debug and trace components include:

- Optional ITM for support of `printf()` style debugging, using instrumentation trace.
- Interfaces suitable for:
  - Passing on-chip data through a *Trace Port Interface Unit* (TPIU) to a *Trace Port Analyzer* (TPA), including *Serial Wire Output* (SWO) mode.
  - A ROM table to allow debuggers to determine which components are implemented in the Cortex®-M33 processor
  - Debugger access to all memory and registers in the system, including access to memory-mapped devices, access to internal core registers when the core is halted, and access to debug control registers even when reset is asserted.

2.5 Interfaces

The processor has various external interfaces.

Code and System AHB interfaces

Harvard AHB bus architecture supporting exclusive transactions and security state.

System AHB interface

The *System AHB* (S-AHB) interface is used for any instruction fetch and data access to the memory-mapped SRAM, Peripheral, External RAM and External device, or Vendor_SYS regions of the Arm®v8-M memory map.

Code AHB interface

The *Code AHB* (C-AHB) interface is used for any instruction fetch and data access to the Code region of the Arm®v8-M memory map.

External Private Peripheral Bus

The *External PPB* (EPPB) APB interface enables access to CoreSight-compatible debug and trace components in a system connected to the processor.

Secure attribution interface

The processor has an interface that connects to an external *Implementation Defined Attribution Unit* (IDAU), which enables your system to set security attributes based on address.

ATB interfaces

The ATB interfaces output trace data for debugging. The ATB interfaces are compatible with the CoreSight architecture. See the *Arm® CoreSight™ Architecture Specification v2.0* for more information. The instruction ATB interface is used by the optional ETM, and the instrumentation ATB interface is used by the optional *Instrumentation Trace Macrocell* (ITM).

Micro Trace Buffer interfaces

The *Micro Trace Buffer* (MTB) AHB slave interface and SRAM interface are for the optional CoreSight Micro Trace Buffer.

Coprocessor interface

The coprocessor interface is designed for closely coupled external accelerator hardware.

Debug AHB interface

The *Debug AHB* (D-AHB) slave interface allows a debugger access to registers, memory, and peripherals. The D-AHB interface provides debug access to the processor and the complete memory map.

Cross Trigger Interface

The processor includes an optional *Cross Trigger Interface* (CTI) Unit that has an interface that is suitable for connection to external CoreSight components using a *Cross Trigger Matrix* (CTM).

Power control interface

The processor optionally supports a number of internal power domains which can be enabled and disabled using Q-channel interfaces connected to a *Power Management Unit* (PMU) in the system.

2.6 Compliance

The processor complies with, or implements, the relevant Arm® architectural standards and protocols, and relevant external standards.

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

Arm® architecture

The processor is compliant with the following:

- Arm®v8-M Main Extension.
- Arm®v8-M Security Extension.
- Arm®v8-M *Protected Memory System Architecture* (PMSA).
- Arm®v8-M Floating-point Extension.
- Arm®v8-M *Digital Signal Processing* (DSP) Extension.
- Arm®v8-M Debug Extension.
- Arm®v8-M *Flash Patch Breakpoint* (FPB) architecture version 2.0.
- Arm®v8-M *Custom Datapath Extension* (CDE).

Bus architecture

The processor provides external interfaces that comply with the AMBA 5 AHB5 protocol. The processor also implements interfaces for CoreSight and other debug components using the APB4 protocol and ATBv1.1 part of the AMBA 4 ATB protocol.

For more information, see the:

- *Arm® AMBA® 5 AHB Protocol Specification*.
- *AMBA® APB Protocol Version 2.0 Specification*.
- *Arm® AMBA® 4 ATB Protocol Specification ATBv1.0 and ATBv1.1* .

The processor also provides a Q-Channel interface. See the *AMBA® Low Power Interface Specification*

Debug

The debug features of the processor implement the Arm® debug interface architecture.

See the *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2*

Embedded Trace Macrocell

The trace features of the processor implement the Arm® *Embedded Trace Macrocell* (ETM) v4.2 architecture.

See the *Arm® CoreSight™ ETM-M33 Technical Reference Manual* for more information.

Floating-Point Unit

The Cortex®-M33 processor with FPU supports single-precision arithmetic as defined by the FPv5 architecture that is part of the Arm®v8-M architecture. The FPU provides floating-point computation functionality that is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*.

2.7 Design process

The processor is delivered as synthesizable RTL that must go through 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 implemented design into a SoC. This includes connecting it to a memory system and peripherals.

Programming

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

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

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

The operation of the final device depends on:

Build configuration

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

Configuration inputs

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

Software configuration

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

Note

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

2.8 Documentation

The Cortex®-M33 processor documentation can help you complete the top-level processes of implementation, integration, and programming that are required to use the product correctly.

The Cortex®-M33 processor documentation comprises a Technical Reference Manual, an Integration and Implementation Manual, and User Guide Reference Material.

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the Cortex®-M33 processor. It is required at all stages of the design flow. Some behavior described in the TRM might not be relevant because of the way that the Cortex®-M33 processor is implemented and integrated. If you are programming the Cortex®-M33 processor, then contact the implementer to determine:

- The build configuration of the implementation.
- What integration, if any, was performed before implementing the processor.

Integration and Implementation Manual

The *Integration and Implementation Manual* (IIM) describes:

- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- How to integrate the processor into a SoC. This includes a description of the integration kit and describes the pins that the integrator must tie off to configure the macrocell for the required integration.
- 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.

User Guide Reference Material

This document provides reference material that Arm® partners can configure and include in a User Guide for an Arm® Cortex®-M33 processor. Typically:

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

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

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

See the *Additional reading* section for more information about the books that are associated with the Cortex®-M33 processor.

Related information

Useful resources for M33 Processor TRM - referable content

2.9 Product revisions

This section describes the differences in functionality between product revisions.

r0p0 First release.

r0p1 This release includes the following changes:

- Updated CPUID reset value, `0x410FD211`.
- The Cortex®-M33 processor optionally supports stalls to guarantee the delivery of trace packets. As a result, the ITM_TCR.STALLENA bit field is now RW.
- Various engineering errata fixes.

r0p2 This release includes the following changes:

- Updated CPUID reset value, `0x410FD212`.
- Various engineering errata fixes.

r0p3 This release includes the following changes:

- Updated CPUID reset value, `0x410FD213`.
- Various engineering errata fixes.

r0p4 This release includes the following changes:

- Updated CPUID reset value, `0x410FD214`.
- Various engineering errata fixes.

r1p0 This release includes the following changes:

- Updated CPUID reset value, `0x411FD210`.
- Implementation of the *Custom Datapath Extension* (CDE) with support for the *Arm Custom Instructions* (ACIs).
- Support for *Software Test Library* (STL) and *Software Built-In Self Test* (SBIST) controller.

- Various engineering errata fixes.

3. Programmers Model

This chapter describes the Cortex®-M33 processor register set, modes of operation, and provides other information for programming the processor.

3.1 About the programmers model

The Cortex®-M33 programmers model is an implementation of the Arm®v8-M Main Extension architecture.

For a complete description of the programmers model, refer to the *Arm®v8-M Architecture Reference Manual*, which also contains the Arm®v8-M Thumb® instructions. In addition, other options of the programmers model are described in the System Control, MPU, NVIC, FPU, Debug, DWT, ITM, and TPIU features topics.

Related information

System Control on page 36
Security Attribution and Memory Protection on page 45
Nested Vectored Interrupt Controller on page 49
Floating-Point Unit on page 52
Debug on page 66
Data Watchpoint and Trace Unit on page 78
Instrumentation Trace Macrocell Unit on page 73

3.2 Modes of operation and execution

The Cortex®-M33 processor supports Secure and Non-secure security states, Thread and Handler operating modes, and can run in either Thumb or Debug operating states. In addition, the processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode.

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

Security states

When the Arm®v8-M Security Extension is included in the processor, the programmers model includes two orthogonal security states, Secure state and Non-secure state. When the Security Extension is implemented, the processor always resets into Secure state. When the security state is not implemented, the processor resets into Non-secure state. Each security state includes a set of independent operating modes and supports both privileged and unprivileged user access. Registers in the System Control Space are banked across Secure and Non-secure state, with the Non-secure register view available at an aliased address to Secure state. When the Arm®v8-M Security Extension is not included in the processor, the programmers model includes only the Non-secure state.

Operating modes

For each security state, the processor can operate in Thread or Handler mode. 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 to Thread mode. Privileged and Unprivileged code can run in Thread mode.
- The processor enters Handler mode as a result of an exception. All code is privileged in Handler mode.

The processor can change security state on taking an exception, for example when a Secure exception is taken from Non-secure state, the Thread mode enters the Secure state Handler mode.

The processor can also call Secure functions from Non-secure state and Non-secure functions from Secure state. The Security Extension includes requirements for these calls to prevent secure data from being accessed in Non-secure state.

Operating states

The processor can operate in Thumb® or Debug state:

- Thumb® state is the state of normal execution running 16-bit and 32-bit halfword-aligned Thumb® instructions.
- Debug state 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 appropriate to the current security state. Privileged execution has access to all resources available to the security state. Handler mode is always privileged. Thread mode can be privileged or unprivileged.

3.3 Instruction set summary

The processor implements the following instruction from Arm®v8-M:

- All base instructions.
- All instructions in the Main Extension.
- Optionally all instructions in the Security Extension.
- Optionally all instructions in the DSP Extension.
- Optionally all single-precision instructions and double precision load and store instructions in the Floating-point Extension.

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

The processor also implements *Custom Datapath Extension* (CDE) instructions. The CDE introduces 2×3 classes of instructions in the coprocessor instruction space:

- Three classes operate on the general-purpose register file.
- Three classes operate on the floating-point register file.

For specific information on the CDE instructions implemented in the processor, see 9. Arm Custom Instructions on page 60. For general information on CDE instructions, see the Arm® *Custom Datapath Extension Architecture*.

3.4 Memory model

The processor contains a bus matrix that arbitrates instruction fetches and memory accesses from the processor core between the external memory system and the internal *System Control Space* (SCS) and debug components.

Priority is usually given to the processor to ensure that any debug accesses are as non-intrusive as possible.

The system memory map is Arm®v8-M Main Extension compliant, and is common both to the debugger and processor accesses.

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.

The following table 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 NVIC and debug components, can be modified using an implemented MPU.

Table 3-1: Default memory map

Address Range (inclusive)	Region	Interface
0x00000000 - 0x1FFFFFFF	Code	Instruction and data accesses performed on C-AHB.
0x20000000 - 0x3FFFFFFF	SRAM	Instruction and data accesses performed on S-AHB. Any attempt to execute instructions from the peripheral and external device region results in a MemManage fault.
0x40000000 - 0x5FFFFFFF	Peripheral	
0x60000000 - 0x9FFFFFFF	External RAM	
0xA0000000 - 0xDFFFFFFF	External device	

Address Range (inclusive)	Region	Interface
`0xE0000000` - `0xE00FFFFF`	PPB	Reserved for system control and debug. Cannot be used for exception vector tables. Data accesses are either performed internally or on EPPB. Accesses in the range: **`0xE0000000-0xE0043FFF`** Are handled within the processor. **`0xE0044000-0xE00FFFFF`** Appear as APB transactions on the EPPB interface of the processor. Any attempt to execute instructions from the region results in a MemManage fault.
`0xE0100000` - `0xFFFFFFFF`	Vendor_SYS	Partly reserved for future processor feature expansion. Any attempt to execute instructions from the region results in a MemManage fault. Data accesses are performed on S-AHB

When the Arm®v8-M Security Extension is included, the security level associated with an address is determined by either the internal *Secure Attribution Unit* (SAU) or an external *Implementation Defined Attribution Unit* (IDAU) in the system. Some internal peripherals have memory-mapped registers in the PPB region which are banked between Secure and Non-secure state. When the processor is in Secure state, software can access both the Secure and Non-secure versions of these registers. The Non-secure versions are accessed using an aliased address. If the Arm®v8-M Security Extension is not included, all memory is treated as Non-secure.

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

3.4.1 Private Peripheral Bus

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

The internal PPB provides access to:

- The *System Control Space* (SCS), including the *Memory Protection Unit* (MPU), *Secure Attribution Unit* (SAU), if included, and the *Nested Vectored Interrupt Controller* (NVIC).
- The *Data Watchpoint and Trace* (DWT) unit, if included.
- The *Breakpoint Unit* (BPU), if included.
- The *Embedded Trace Macrocell* (ETM), if included.
- CoreSight *Micro Trace Buffer* (MTB), if included.
- *Cross Trigger Interface* (CTI), if included.
- The ROM table.

The *external PPB* (EPPB) provides access to implementation-specific external areas of the PPB memory map.

3.4.2 Unaligned accesses

The Cortex®-M33 processor supports unaligned accesses. They are converted into two or more aligned AHB transactions on the C-AHB or S-AHB master ports on the processor.

Unaligned support is only available for load/store singles (LDR, LDRH, STR, STRH, TBH) to addresses in Normal memory. Load/store double and load/store multiple instructions already support word aligned accesses, but do not permit other unaligned accesses, and generate a fault if this is attempted. Unaligned accesses in Device memory are not permitted and fault. Unaligned accesses that cross memory map boundaries are architecturally **UNPREDICTABLE**.

Note

If CCR.UNALIGN_TRP for the current Security state is set, any unaligned accesses generate a fault.

3.5 Exclusive monitor

The Cortex®-M33 processor implements a local exclusive monitor. The local monitor within the processor has been constructed so that it does not hold any physical address, but instead treats any store-exclusive access as matching the address of the previous load-exclusive. This means that the implemented exclusives reservation granule is the entire memory address range.

For more information about semaphores and the local exclusive monitor, see the *Arm®v8-M Architecture Reference Manual*.

3.6 Processor core registers summary

The following table shows the processor core register set summary. Each of these registers is 32 bits wide. When the Arm®v8-M Security Extension is included, some of the registers are banked. The Secure view of these registers is available when the Cortex®-M33 processor is in Secure state and the Non-secure view when Cortex®-M33 processor is in Non-secure state.

Table 3-2: Processor core register set summary

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

Name	Description
MSP (R13) PSP (R13)	The *Stack Pointer* (SP) is register R13. In Thread mode, the CONTROL register indicates the stack pointer to use, *Main Stack Pointer* (MSP) or *Process Stack Pointer* (PSP). When the Arm®v8-M Security Extension is included, there are two MSP registers in the Cortex®-M33 processor: • MSP_NS for the Non-secure state. • MSP_S for the Secure state. When the Arm®v8-M Security Extension is included, there are two PSP registers in the Cortex®-M33 processor: • PSP_NS for the Non-secure state. • PSP_S for the Secure state.
MSPLIM PSPLIM	The stack limit registers limit the extent to which the MSP and PSP registers can descend respectively. When the Arm®v8-M Security Extension is included, there are two MSPLIM registers in the Cortex®-M33 processor: • MSPLIM_NS for the Non-secure state. • MSPLIM_S for the Secure state. When the Arm®v8-M Security Extension is included, there are two PSPLIM registers in the Cortex®-M33 processor: • PSPLIM_NS for the Non-secure state. • PSPLIM_S for the Secure state.
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 exceptions with configurable priority. For information about the exception model the processor supports, see 3.7 Exceptions on page 34. When the Arm®v8-M Security Extension is included, there are two PRIMASK registers in the Cortex®-M33 processor: • PRIMASK_NS for the Non-secure state. • PRIMASK_S for the Secure state.
BASEPRI	The BASEPRI register defines the minimum priority for exception processing. When the Arm®v8-M Security Extension is included, there are two BASEPRI registers in the Cortex®-M33 processor: • BASEPRI_NS for the Non-secure state. • BASEPRI_S for the Secure state.
FAULTMASK	The FAULTMASK register prevents activation of all exceptions except for **NON-MASKABLE INTERRUPT** (NMI) and optionally Secure HardFault. When the Arm®v8-M Security Extension is included, there are two FAULTMASK registers in the Cortex®-M33 processor: • FAULTMASK_NS for the Non-secure state. • FAULTMASK_S for the Secure state.

Name	Description
CONTROL	The CONTROL register controls the stack used, and optionally the privilege level, when the processor is in Thread mode. When the Arm®v8-M Security Extension is included, there are two CONTROL registers in the Cortex®-M33 processor: • CONTROL_NS for the Non-secure state. • CONTROL_S for the Secure state.

Note

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

3.7 Exceptions

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

3.7.1 Exception handling and prioritization

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

When handling exceptions:

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

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

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

When the Arm®v8-M Security Extension is included, exceptions can be specified as either Secure or Non-secure. When an exception is taken the processor switches to the associated security state. The priority of Secure and Non-secure exceptions can be programmed independently. It is possible to deprioritize Non-secure configurable exceptions using the AIRCR.PRIS bit field to enable Secure interrupts to take priority.

When taking and returning from an exception, the register state is always stored using the stack pointer associated with the background security state. When taking a Non-secure exception from

Secure state, all the register state is stacked and then registers are cleared to prevent Secure data being available to the Non-secure handler. The vector base address is banked between Secure and Non-secure state. VTOR_S contains the Secure vector base address, and VTOR_NS contains the Non-secure vector base address. These registers can be programmed by software, and also initialized at reset by the system. If the Arm®v8-M Security Extension is not included all exceptions are Non-secure and only VTOR_NS is used to determine the vector base address.

Note

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

---

4. System Control

This chapter describes registers that contain **IMPLEMENTATION DEFINED** information or functionality.

4.1 Identification register summary

Identification registers allow software to determine the features and functionality available in the implemented processor.

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

Note

If the Arm®v8-M Security Extension is not included, then only the Non-secure entries are available and the entire alias space is RAZ/WI.

Table 4-1: Identification register summary

Address	Register	Type	Processor security state	Reset value	Description
0xE000ED00	CPUID	RO	Secure	0x411FD210	CPUID Base Register
			Non-secure		CPUID Base Register (NS)
0xE002ED00	CPUID_NS	RO	Secure		CPUID Base Register (NS)
			Non-secure		RAZ/WI
0xE000ED40	ID_PFR0	RO	Secure	0x00000030	Processor Feature Register 0
			Non-secure		Processor Feature Register 0 (NS)
0xE002ED40	ID_PFR0_NS	RO	Secure		Processor Feature Register 0 (NS)
			Non-secure		RAZ/WI
0xE000ED44	ID_PFR1	RO	Secure	0x000002x0[3]	Processor Feature Register 1
			Non-secure		Processor Feature Register 1 (NS)
0xE002ED44	ID_PFR1_NS	RO	Secure		Processor Feature Register 1 (NS)
			Non-secure		RAZ/WI
0xE000ED48	ID_DFR0	RO	Secure	0x00200000 0x[2]	Debug Feature Register 0
			Non-secure		Debug Feature Register 0 (NS)
0xE002ED48	ID_DFR0_NS	RO	Secure		Debug Feature Register 0 (NS)
			Non-secure		RAZ/WI
0xE000ED4C	ID_AFR0	RO	Secure	Depends on the CDEMAPPEDONCP and CDERTLID parameters.	Auxiliary Feature Register 0
			Non-secure		Auxiliary Feature Register 0 (NS)

Address	Register	Type	Processor security state	Reset value	Description
0xE002ED4C	ID_AFR0_NS	RO	Secure		Auxiliary Feature Register 0 (NS)
			Non-secure		RAZ/WI
0xE000ED50	ID_MMFR0	RO	Secure	0x00101F40	Memory Model Feature Register 0
			Non-secure		Memory Model Feature Register 0 (NS)
0xE002ED50	ID_MMFR0_NS	RO	Secure		Memory Model Feature Register 0 (NS)
			Non-secure		RAZ/WI
0xE000ED54	ID_MMFR1	RO	Secure	0x00000000	Memory Model Feature Register 1
			Non-secure		Memory Model Feature Register 1 (NS)
0xE002ED54	ID_MMFR1_NS	RO	Secure		Memory Model Feature Register 1 (NS)
			Non-secure		RAZ/WI
0xE000ED58	ID_MMFR2	RO	Secure	0x01000000	Memory Model Feature Register 2
			Non-secure		Memory Model Feature Register 2 (NS)
0xE002ED58	ID_MMFR2_NS	RO	Secure		Memory Model Feature Register 2 (NS)
			Non-secure		RAZ/WI
0xE000ED5C	ID_MMFR3	RO	Secure	0x00000000	Memory Model Feature Register 3
			Non-secure		Memory Model Feature Register 3 (NS)
0xE002ED5C	ID_MMFR3_NS	RO	Secure		Memory Model Feature Register 3 (NS)
			Non-secure		RAZ/WI
0xE000ED60	ID_ISAR0	RO	Secure	0x011x1110[5]	Instruction Set Attributes Register 0
			Non-secure		Instruction Set Attributes Register 0 (NS)
0xE002ED60	ID_ISAR0_NS	RO	Secure		Instruction Set Attributes Register 0 (NS)
			Non-secure		RAZ/WI
0xE000ED64	ID_ISAR1	RO	Secure	0x0221x000[6]	Instruction Set Attributes Register 1
			Non-secure		Instruction Set Attributes Register 1 (NS)
0xE002ED64	ID_ISAR1_NS	RO	Secure		Instruction Set Attributes Register 1 (NS)
			Non-secure		RAZ/WI

Address	Register	Type	Processor security state	Reset value	Description
0xE000ED68	ID_ISAR2	RO	Secure	0x20xx2232[6]	Instruction Set Attributes Register 2
			Non-secure		Instruction Set Attributes Register 2 (NS)
0xE002ED68	ID_ISAR2_NS	RO	Secure		Instruction Set Attributes Register 2 (NS)
			Non-secure		RAZ/WI
0xE000ED6C	ID_ISAR3	RO	Secure	0x011111xx[6]	Instruction Set Attributes Register 3
			Non-secure		Instruction Set Attributes Register 3 (NS)
0xE002ED6C	ID_ISAR3_NS	RO	Secure		Instruction Set Attributes Register 3 (NS)
			Non-secure		RAZ/WI
0xE000ED70	ID_ISAR4	RO	Secure	0x01310132	Instruction Set Attributes Register 4
			Non-secure		Instruction Set Attributes Register 4 (NS)
0xE002ED70	ID_ISAR4_NS	RO	Secure		Instruction Set Attributes Register 4 (NS)
			Non-secure		RAZ/WI
0xE000ED78	CLIDR	RO	Secure	0x00000000	Cache Level ID Register
			Non-secure		Cache Level ID Register (NS)
0xE002ED78	CLIDR _NS	RO	Secure		Cache Level ID Register (NS)
			Non-secure		RAZ/WI
0xE000ED7C	CTR	RO	Secure	0x8000C000	Cache Type Register
			Non-secure		Cache Type Register (NS)
0xE002ED7C	CTR_NS	RO	Secure		Cache Type Register (NS)
			Non-secure		RAZ/WI
0xE000EF40	MVFR0	RO	Secure	0x10110021[4]	Media and VFP Feature Register 0
			Non-secure		Media and VFP Feature Register 0 (NS)
0xE002EF40	MVFR0_NS	RO	Secure		Media and VFP Feature Register 0 (NS)
			Non-secure		RAZ/WI
0xE000EF44	MVFR1	RO	Secure	0x11000011[4]	Media and VFP Feature Register 1
			Non-secure		Media and VFP Feature Register 1 (NS)
0xE002EF44	MVFR1_NS	RO	Secure		Media and VFP Feature Register 1 (NS)
			Non-secure		RAZ/WI

Address	Register	Type	Processor security state	Reset value	Description
0xE000EF48	MVFR2	RO	Secure	0x00000040[4]	Media and VFP Feature Register 2
			Non-secure		Media and VFP Feature Register 2 (NS)
0xE002EF48	MVFR2_NS	RO	Secure		Media and VFP Feature Register 2 (NS)
			Non-secure		RAZ/WI
0xE000EFD0	PIDR4	RO	Secure	0x00000004	CoreSight Peripheral ID Register 4
			Non-secure		CoreSight Peripheral ID Register 4 (NS)
0xE002EFD0	PIDR4_NS	RO	Secure		CoreSight Peripheral ID Register 4 (NS)
			Non-secure		RAZ/WI
0xE000EFD4	PIDR5	RO	Secure	0x00000000	CoreSight Peripheral ID Register 5
			Non-secure		CoreSight Peripheral ID Register 5 (NS)
0xE002EFD4	PIDR5_NS	RO	Secure		CoreSight Peripheral ID Register 5 (NS)
			Non-secure		RAZ/WI
0xE000EFD8	PIDR6	RO	Secure	0x00000000	CoreSight Peripheral ID Register 6
			Non-secure		CoreSight Peripheral ID Register 6 (NS)
0xE002EFD8	PIDR6_NS	RO	Secure		CoreSight Peripheral ID Register 6 (NS)
			Non-secure		RAZ/WI
0xE000EFDC	PIDR7	RO	Secure	0x00000000	CoreSight Peripheral ID Register 7
			Non-secure		CoreSight Peripheral ID Register 7 (NS)
0xE002EFDC	PIDR7_NS	RO	Secure		CoreSight Peripheral ID Register 7 (NS)
			Non-secure		RAZ/WI
0xE000EFE0	PIDR0	RO	Secure	0x00000021	CoreSight Peripheral ID Register 0
			Non-secure		CoreSight Peripheral ID Register 0 (NS)
0xE002EFE0	PIDR0_NS	RO	Secure		CoreSight Peripheral ID Register 0 (NS)
			Non-secure		RAZ/WI
0xE000EFE4	PIDR1	RO	Secure	0x000000BD	CoreSight Peripheral ID Register 1

Address	Register	Type	Processor security state	Reset value	Description
			Non-secure		CoreSight Peripheral ID Register 1 (NS)
0xE002EFE4	PIDR1_NS	RO	Secure		CoreSight Peripheral ID Register 1 (NS)
			Non-secure		RAZ/WI
0xE000EFE8	PIDR2	RO	Secure	0x0000000B	CoreSight Peripheral ID Register 2
			Non-secure		CoreSight Peripheral ID Register 2 (NS)
0xE002EFE8	PIDR2_NS	RO	Secure		CoreSight Peripheral ID Register 2 (NS)
			Non-secure		RAZ/WI
0xE000EFEC	PIDR3	RO	Secure	0x00000000[1]	CoreSight Peripheral ID Register 3
			Non-secure		CoreSight Peripheral ID Register 3 (NS)
0xE002EFEC	PIDR3_NS	RO	Secure		CoreSight Peripheral ID Register 3 (NS)
			Non-secure		RAZ/WI
0xE000EFF0	CIDR0	RO	Secure	0x0000000D	CoreSight Component ID Register 0
			Non-secure		CoreSight Component ID Register 0 (NS)
0xE002EFF0	CIDR0_NS	RO	Secure		CoreSight Component ID Register 0 (NS)
			Non-secure		RAZ/WI
0xE000EFF4	CIDR1	RO	Secure	0x00000090	CoreSight Component ID Register 1
			Non-secure		CoreSight Component ID Register 1 (NS)
0xE000EFF4	CIDR1_NS	RO	Secure		CoreSight Component ID Register 1 (NS)
			Non-secure		RAZ/WI
0xE002EFF8	CIDR2	RO	Secure	0x00000005	CoreSight Component ID Register 2
			Non-secure		CoreSight Component ID Register 2 (NS)
0xE002EFF8	CIDR2_NS	RO	Secure		CoreSight Component ID Register 2 (NS)
			Non-secure		RAZ/WI
0xE000EFFC	CIDR3	RO	Secure	0x000000B1	CoreSight Component ID Register 3
			Non-secure		CoreSight Component ID Register 3 (NS)

[1] Dependent on the exact revision of the silicon as documented in *Arm® CoreSight™ Architecture Specification v2.0*.

Address	Register	Type	Processor security state	Reset value	Description
0xE002EFFC	CIDR3_NS	RO	Secure		CoreSight Component ID Register 3 (NS)
			Non-secure		RAZ/WI
0xE000EFBC	DDEVARCH	RO	Secure	0x47702A04	CoreSight Device Architecture Register
			Non-secure		CoreSight Device Architecture Register (NS)
0xE002EFBC	DDEVARCH_NS	RO	Secure		CoreSight Device Architecture Register (NS)
			Non-secure		RAZ/WI

4.2 Auxiliary Control Register

The ACTLR Register contains a number of fields that allow software to control the processor features and functionality.

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

Configurations This register is always implemented.

Attributes A 32-bit RW register located at `0xE000E008`. Non-secure alias is provided using ACTLR_NS, located at `0xE002E008`. This register is banked between Security domains.

Field	Name	Description
[31:30]	Reserved	These bits are reserved for software testing purposes only.
[29]	EXTEXCLALL	0= Normal operation; memory requests on C-AHB or S-AHB interfaces associated with LDREX and STREX instructions or LDAEX and STLEX instructions only assert HEXCL and respond to HEXOKAY if the address is shareable. 1= All memory requests on C-AHB or S-AHB interfaces associated with LDREX and STREX instructions or LDAEX and STLEX instructions assert HEXCL and respond to HEXOKAY irrespective of the shareable attribute associated with the address. Setting EXTEXCLALL allows external exclusive operations to be used in a configuration with no MPU. This is because the default memory map does not include any shareable Normal memory.
[28:14]	Reserved	These bits are reserved for future use and must be treated as UNK/SBZP

[2] When minimal debug support is implemented, this value is `0x00000000`.
[3] ID_PFR1[7:4] indicates support for the Arm®v8-M Security Extension. ID_PFR1[7:4] reads as `0b0001` if the Security Extension is supported otherwise ID_PFR1[7:4] reads as `0b0000`.
[4] When the FPU is not implemented, this value is `0x00000000`.
[5] ID_ISAR0[19:16] depend on whether the external coprocessor interface is included in the processor.
[6] ID_ISAR1[15:12], ID_ISAR2[31:28], ID_ISAR2[23:16] and ID_ISAR3[7:0] depend on whether theArm®v8-M DSP extension is included in the processor.

Field	Name	Description
[13]	SBIST	0= Reset and recommended value. 1= Features enabled internally by Software Test Library (STL). This is restored to the original value when the STL exists. See the *Arm® Cortex®-M33 STL User Guide* for more information about the STL.
[12]	DISITMATBFLUSH	0= Normal operation. 1= ITM/DWT ATB flush disabled. When disabled AFVALID is ignored and AFREADY is held HIGH.
[11]	Reserved	This bit is reserved for future use and must be treated as UNK/SBZP
[10]	FPEXCODIS	0= normal operation 1= FPU exception outputs are disabled See Floating-point Unit Chapter for more information about the FPU exception outputs.
[9]	DISOOFP	0= normal operation 1= disables floating-point instructions completing out of order with respect to non-floating-point instructions.
[8:3]	Reserved	These bits are reserved for future use and must be treated as UNK/SBZP.
[2]	DISFOLD	0= normal operation. 1= dual-issue functionality is disabled **Note:** Setting this bit decreases performance.
[1]	Reserved	These bits are reserved for future use and must be treated as UNK/SBZP.
[0]	DISMCYCINT	0= normal operation. 1= disables interruption of multi-cycle instructions. This increases the interrupt latency of the processor because load/store and multiply/divide operations complete before interrupt stacking occurs.

4.3 CPUID Base Register

The CPUID Register specifies the ID number, the version number, and implementation details of the processor core.

Usage Constraints

Privileged access permitted only. Unprivileged accesses generate a fault.

This register is word accessible only, sub-word transactions are **UNPREDICTABLE**.

Configurations

This register is always implemented.

Attributes

Described in the System control registers table.

The following figure shows the CPUID bit assignments.

Figure 4-1: CPUID bit assignments

31 – 24	23 – 20	19 – 16	15 – 4	3 – 0
IMPLEMENTER	VARIANT	(Constant)	PARTNO	REVISION

The following table shows the CPUID bit assignments.

Table 4-3: CPUID bit assignments

Bits	NAME	Function
[31:24]	IMPLEMENTER	Indicates implementer: `0x41` = Arm®
[23:20]	VARIANT	Indicates processor revision: `0x1` = Revision 1
[19:16]	(Constant)	Reads as `0xF`
[15:4]	PARTNO	Indicates part number: `0xD21` = Cortex®-M33
[3:0]	REVISION	Indicates patch release: `0x0` = Patch 0

4.4 Auxiliary Feature Register 0

The ID_AFR0 register provides information about the **IMPLEMENTATION DEFINED** features of the processor.

Usage constraints

Privileged access permitted only. Unprivileged accesses generate a fault.

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

Configurations

This register is always implemented.

Attributes

A 32-bit read-only register located at `0xE000ED4C` for Secure world. Software can access the Non-secure version of this register using ID_AFR0_NS located at `0xE002ED4C`. `0xE002ED4C` is **RES0** to software executing in Non-secure state and the debugger. This register is not banked between security states.

Figure 4-2: ID_AFR0 bit assignments

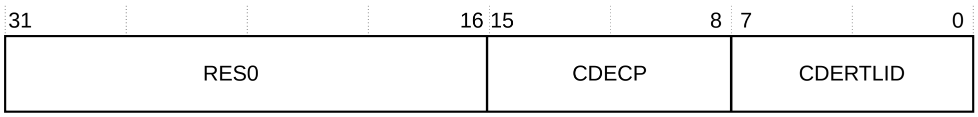


Table 4-4: ID_AFR0 bit assignments

Field	Name	Description
[31:16]	RES0	Reserved
[15:8]	CDECP	Indicates for each coprocessor whether it is used by a CDE module and not by the coprocessor interface: **0** Coprocessor used by the coprocessor interface **1** Coprocessor used by a CDE module
[7:0]	CDERTLID	Software can use this field to read the value of the CDERTLID parameter. This parameter manages the CDE customization that might be needed in systems with more than one Cortex®-M33 processor.

5. Security Attribution and Memory Protection

This chapter describes the security attribution and memory protection facilities that the Cortex®-M33 processor provides.

5.1 About security attribution and memory protection

Security attribution and memory protection in the processor is provided by the optional *Security Attribution Unit* (SAU) and the optional *Memory Protection Units* (MPUs).

The SAU is a programmable unit that determines the security of an address. The SAU is only implemented if the Arm®v8-M Security Extension is included in the processor. The number of regions that are included in the SAU can be configured in the Cortex®-M33 implementation to be 0, 4 or 8.

For instructions and data, the SAU returns the security attribute that is associated with the address.

For instructions, the attribute determines the allowable Security state of the processor when the instruction is executed. It can also identify whether code at a Secure address can be called from Non-secure state.

For data, the attribute determines whether a memory address can be accessed from Non-secure state, and also whether the external memory request is marked as Secure or Non-secure.

If a data access is made from Non-secure state to an address marked as Secure, then a SecureFault exception is taken by the processor. If a data access is made from Secure state to an address marked as Non-secure, then the associated memory access is marked as Non-secure.

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 5-1: Examples of Highest Security Level Region

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

The register fields SAU_CTRL.EN and SAU_CTRL.ALLNS control the enable state of the SAU and the default security level when the SAU is disabled. Both SAU_CTRL.EN and SAU_CTRL.ALLNS reset to zero disabling the SAU and setting all memory, apart from some specific regions in the PPB space to Secure level. If the SAU is not enabled, and SAU_CTRL.ALLNS is zero, then the IDAU cannot set any regions of memory to a security level lower than Secure, for example Secure NSC or NS. If the SAU is enabled, then SAU_CTRL.ALLNS does not affect the Security level of memory.

The Cortex®-M33 processor supports the Arm®v8-M *Protected Memory System Architecture* (PMSA). The MPU is an optional component and, when implemented, provides full support for:

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

MPU mismatches and permission violations invoke the MemManage 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 MPU can be configured to support 0, 4, 8, 12 or 16 memory regions.

If the Arm®v8-M Security Extension is included in the Cortex®-M33 processor, the MPU is banked between Secure and Non-secure states. The number of regions in the Secure and Non-secure MPU can be configured independently and each can be programmed to protect memory for the associated Security state.

5.2 SAU register summary

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

Address	Name	Type	Reset value	Processor security state	Description
`0xE000EDD0`	SAU_CTRL	RW	`0x00000000`	Secure	SAU Control register
				Non-secure	RAZ/WI
`0xE000EDD4`	SAU_TYPE	RO	`0000000x`[7]	Secure	SAU Type register
				Non-secure	RAZ/WI
`0xE000EDD8`	SAU_RNR	RW	UNKNOWN	Secure	SAU Region Number Register
				Non-secure	RAZ/WI
E000EDDC	SAU_RBAR	RW	UNKNOWN	Secure	SAU Region Base Address Register
				Non-secure	RAZ/WI
`0xE000EDE0`	SAU_RLAR	RW	UNKNOWN	Secure	SAU Region Limit Address Register
			Non-secure	RAZ/WI	

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

5.3 MPU register summary

The *Memory Protection Unit* (MPU) has various registers associated to its function.

Each of these registers is 32 bits wide. If the MPU is not present in the implementation, then all of these registers read as zero. The following table shows the MPU register summary.

Table 5-3: MPU register summary

Address	Name	Type	Reset value	Processor security state	Description
0xE000ED90	MPU_TYPE	RO	0x0000xx00[8]	Secure	MPU Type Register (S)
			0x0000xx00[9]	Non-secure	MPU Type Register (NS)
0xE002ED90	MPU_TYPE_NS		0x0000xx00[9]	Secure	MPU Type Register (NS)
				Non-secure	RAZ/WI
0xE000ED94	MPU_CTRL	RW	0x00000000	Secure	MPU Control Register (S)
			0x00000000	Non-secure	MPU Control register (NS)
0xE002ED94	MPU_CTRL_NS		0x00000000	Secure	MPU Control register (NS)
				Non-secure	RAZ/WI
0xE000ED98	MPU_RNR	RW	UNKNOWN	Secure	MPU Region Number Register (S)
			UNKNOWN	Non-secure	MPU Region Number Register (NS)
0xE002ED98	MPU_ RNR_NS	RW	UNKNOWN	Secure	MPU Region Number Register (NS)
				Non-secure	RAZ/WI
0xE000ED9C	MPU_RBAR_A0-MPU_RBAR_A3	RW	UNKNOWN	Secure	MPU Region Base Address Register Aliases 0-3 (S)
			UNKNOWN	Non-secure	MPU Region Base Address Register Aliases 0-3 (NS)
0xE002ED9C	MPU_RBAR_A_0_NS-MPU_RBAR_A_3_NS	RW	UNKNOWN	Secure	MPU Region Base Address Register Aliases 0-3 (NS)
				Non-secure	RAZ/WI
0xE000EDA0	MPU_RLAR_A0-MPU_RLAR_A3	RW	UNKNOWN	Secure	MPU Region Limit Address Register Aliases 0-3 (S)
			UNKNOWN	Non-secure	MPU Region Limit Address Register Aliases 0-3 (NS)
0xE002EDA0	MPU_RLAR_A_0_NS-MPU_RLAR_A_3_NS	RW	UNKNOWN	Secure	MPU Region Limit Address Register Aliases 0-3 (NS)
				Non-secure	RAZ/WI
0xE000EDC0	MPU_MAIR0	RW	UNKNOWN	Secure	MPU Memory Attribute Indirection Register 0 (S)

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

Address	Name	Type	Reset value	Processor security state	Description
			UNKNOWN	Non-secure	MPU Memory Attribute Indirection Register 0 (NS)
`0xE002EDC0`	MPU_MAIR0_NS	RW	UNKNOWN	Secure	MPU Memory Attribute Indirection Register 0 (NS)
				Non-secure	RAZ/WI
`0xE000EDC4`	MPU_MAIR1	RW	UNKNOWN	Secure	MPU Memory Attribute Indirection Register 1 (S)
			UNKNOWN	Non-secure	MPU Memory Attribute Indirection Register 1 (NS)
`0xE002EDC4`	MPU_MAIR1_NS	RW	UNKNOWN	Secure	MPU Memory Attribute Indirection Register 1 (NS)
				Non-secure	RAZ/WI

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

[8] MPU_TYPE[15:8] depends on the number of Secure MPU regions configured. This value can be 0, 4, 8, 12, or 16.
[9] MPU_TYPE[15:8] depends on the number of Non-secure MPU regions configured. This value can be 0, 4, 8, 12, or 16.

6. Nested Vectored Interrupt Controller

This chapter describes the *Nested Vectored Interrupt Controller* (NVIC).

6.1 NVIC programmers model

This section includes a summary of the NVIC registers whose implementation is specific to the Cortex®-M33 processor.

6.1.1 NVIC register summary

The following table shows the NVIC registers with address, name, type, reset, and description information for each register.

Note

- If the Arm®v8-M Security Extension is not included, only the Non-secure entries are available and the entire alias space is RAZ/WI.
- The NVIC_ISERn, NVIC_ICERn, NVIC_ISPRn, NVIC_ICPRn, NVIC_IABRn, and NVIC_IPRn registers are not banked between security states. If an interrupt is configured as Secure in the NVIC_ITNSn register, any access to the corresponding NVIC_ISERn, NVIC_ICERn, NVIC_ISPRn, NVIC_ICPRn, NVIC_IABRn, or NVIC_IPRn registers from Non-secure are treated as RAZ/WI.

Table 6-1: NVIC register summary

Address offset	Name	Type	Processor security state	Reset value	Description
`0xE000E004`	ICTR	RO	Secure	`0x0000000x`[10]	Interrupt Controller Type Register
			Non-secure	`0x0000000x`[10]	Interrupt Controller Type Register (NS)
`0xE002E004`	ICTR_NS	RO	Secure	`0x0000000x`[10]	Interrupt Controller Type Register (NS)
			Non-secure		RAZ/WI
`0xE000E100-0xE000E13C`	NVIC_ISER0-NVIC_ISER15	RW	Secure	`0x00000000`	Interrupt Set-Enable Registers
			Non-secure	`0x00000000`	Interrupt Set-Enable Registers (NS)
`0xE002E100-0xE002E13C`	NVIC_ISER0_NS-NVIC_ISER15_NS	RW	Secure	`0x00000000`	Interrupt Set-Enable Registers (NS)
			Non-secure		RAZ/WI
`0xE000E180-0xE000E1BC`	NVIC_ICER0-NVIC_ICER15	RW	Secure	`0x00000000`	Interrupt Clear-Enable Registers
			Non-secure	`0x00000000`	Interrupt Clear-Enable Registers (NS)

Address offset	Name	Type	Processor security state	Reset value	Description
`0xE002E180-0xE002E1BC`	NVIC_ICER0_NS-NVIC_ICER15_NS	RW	Secure		Interrupt Clear-Enable Registers (NS)
			Non-secure		RAZ/WI
`0xE000E200-0xE000E23C`	NVIC_ISPR0-NVIC_ISPR15	RW	Secure	`0x00000000`	Interrupt Set-Pending Registers
			Non-secure		Interrupt Set-Pending Registers (NS)
`0xE002E200-0xE002E23C`	NVIC_ISPR0_NS-NVIC_ISPR15_NS	RW	Secure		Interrupt Set-Pending Registers (NS)
			Non-secure		RAZ/WI
`0xE000E280-0xE000E2BC`	NVIC_ICPR0-NVIC_ICPR15	RW	Secure	`0x00000000`	Interrupt Clear-Pending Registers
			Non-secure		Interrupt Clear-Pending Registers (NS)
`0xE002E280-0xE002E2BC`	NVIC_ICPR0_NS-NVIC_ICPR15_NS	RW	Secure		Interrupt Clear-Pending Registers (NS)
			Non-secure		RAZ/WI
`0xE000E300-0xE000E33C`	NVIC_IABR0-NVIC_IABR15	RO	Secure	`0x00000000`	Interrupt Active Bit Register
			Non-secure		Interrupt Active Bit Register (NS)
`0xE002E300-0xE002E33C`	NVIC_IABR0_NS-NVIC_IABR15_NS	RO	Secure		Interrupt Active Bit Register (NS)
			Non-secure		RAZ/WI
`0xE000E380-0xE000E3BC`	NVIC_ITNS0-NVIC_ITNS15	RW	Secure	`0x00000000`	Interrupt Target Non-secure Registers
			Non-secure		RAZ/WI
`0xE000E400-0xE000E5DC`	NVIC_IPR0-NVIC_IPR119	RW	Secure	`0x00000000`	Interrupt Priority Registers
			Non-secure		Interrupt Priority Registers (NS)
`0xE002E400-0xE002E5DC`	NVIC_IPR0_NS-NVIC_IPR119_NS	RW	Secure		Interrupt Priority Registers (NS)
			Non-secure		RAZ/WI

6.1.2 Interrupt Controller Type Register

The ICTR register shows the number of interrupt lines that the NVIC supports.

Usage Constraints

There are no usage constraints.

Configurations

This register is available in all processor configurations.

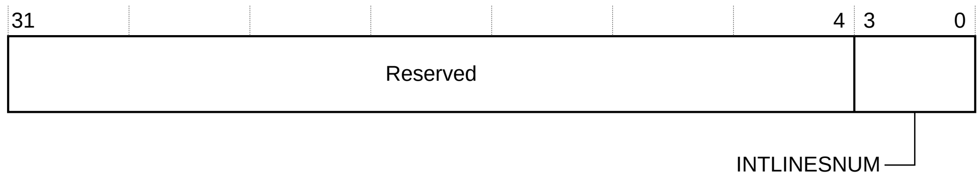
Attributes

See the register summary information.

The following figure shows the ICTR bit assignments.

[10] ICTR[3:0] depends on the number of interrupts included in the processor.

Figure 6-1: ICTR bit assignments



The following table shows the ICTR bit assignments.

Table 6-2: ICTR bit assignments

Bits	Name	Function	Notes
[31:4]	-	Reserved.	-
[3:0]	INTLINESNUM	Total number of interrupt lines in groups of 32: `0b0000` = 1...32 `0b0001` = 33...64 `0b0010` = 65...96 `0b0011` = 97...128 `0b0100` = 129...160 `0b0101` = 161...192 `0b0110` = 193...224 `0b0111` = 225...256 `0b1000` = 257...288 `0b1001` = 289...320 `0b1010` = 321...352 `0b1011` = 353...384 `0b1100` = 385...416 `0b1101` = 417...448 `0b1110` = 449...480	The processor supports a maximum of 480 external interrupts.

7. Floating-Point Unit

This chapter describes the *Floating-Point Unit* (FPU).

7.1 About the FPU

The Cortex®-M33 FPU is an implementation of the single precision variant of the Arm®v8-M Floating-point extension, FPv5 architecture. It provides floating-point computation functionality that is compliant with the ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic, referred to as the IEEE 754 standard.

The FPU supports all single-precision data-processing instructions and data types described in the *Arm®v8-M Architecture Reference Manual*.

7.2 FPU functional description

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

7.2.1 FPU views of the register bank

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

The registers can be viewed as:

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

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

The modes of operation are controlled using the Floating-Point Status and Control Register, FPSCR. For more information about the FPSCR, see the *Arm®v8-M Architecture Reference Manual*.

7.2.2 Modes of operation

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

7.2.2.1 Full-compliance mode

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

7.2.2.2 Flush-to-zero mode

Setting the FPSCR.FZ bit enables *Flush-to-Zero* (FZ) mode.

In FZ mode, the FPU treats all subnormal input operands of arithmetic operations as zeros in the operation. Exceptions that result from a zero operand are signaled appropriately. `VABS`, `VNEG`, and `VMOV` are not considered arithmetic operations and are not affected by FZ mode. A result that is *tiny*, as described in the IEEE 754 standard, where the destination precision is smaller in magnitude than the minimum normal value *before rounding*, is replaced with a zero. The FPSCR.IDC bit indicates when an input flush occurs. The FPSCR.UFC bit indicates when a result flush occurs.

7.2.2.3 Default NaN mode

Setting the FPSCR.DN bit enables *Default NaN* (DN) mode.

In NaN mode, the result of any arithmetic data processing operation that involves an input NaN, or that generates a NaN result, returns the default NaN. Propagation of the fraction bits is maintained only by `VABS`, `VNEG`, and `VMOV` operations. All other arithmetic operations ignore any information in the fraction bits of an input NaN.

7.2.3 Compliance with the IEEE 754 standard

When DN and FZ modes are disabled, FPv5 functionality is compliant with the IEEE 754 standard in hardware. No Support code is required to achieve this compliance.

See the *Arm®v8-M Architecture Reference Manual* for information about FP architecture compliance with the IEEE 754 standard.

7.2.4 Exceptions

The FPU sets the cumulative exception status flag in the FPSCR register as required for each instruction, in accordance with the FPv5 architecture. The FPU does not support exception traps.

The processor also has six output pins, each reflect the status of one of the cumulative exception flags:

FPIXC Masked floating-point inexact exception.
FPUFC Masked floating-point underflow exception.
FPOFC Masked floating-point overflow exception.
FPDZC Masked floating-point divide by zero exception.
FPIDC Masked floating-point input denormal exception.

FPIOC Invalid operation.

When a floating-point context is active, the stack frame is extended to accommodate the floating-point registers. To reduce the additional interrupt latency associated with writing the larger stack frame on exception entry, the processor supports lazy stacking. This means that the processor reserves space on the stack for the FP state, but does not save that state information to the stack unless the processor executes an FPU instruction inside the exception handler.

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

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

7.3 FPU programmers model

This section shows a floating-point system register summary.

7.3.1 Floating-point system registers

The following table shows a summary of the FP system registers in the Cortex®-M33 processor, where FPU is included.

All Cortex®-M33 FPU registers are described in the *Arm®v8-M Architecture Reference Manual*.

Table 7-1: FPU register summary

Address	Name	Type	Reset value	Processor security state	Description
`0xE000EF34`	FPCCR	RW	`0xC0000004`	Secure	FP Context Control Register (S)
			`0xC0000000`	Non-secure	FP Context Control Register (NS)
`0xE002EF34`	FPCCR_NS	RW	`0xC0000000`	Secure	FP Context Control Register (NS)
			-	Non-secure	RAZ/WI
`0xE000EF38`	FPCAR	RW	`0x00000000`	Secure	FP Context Address Register (S)
			-	Non-secure	FP Context Address Register (NS)
`0xE002EF38`	FPCAR_NS	RW	`0x00000000`	Secure	FP Context Address Register (NS)
			-	Non-secure	RAZ/WI
`0xE000EF3C`	FPDSCR	RW	`0x00000000`	Secure	FP Default Status Control Register (S)
			-	Non-secure	FP Default Status Control Register (NS)
`0xE002EF3C`	FPDSCR_NS	RW	`0x00000000`	Secure	FP Default Status Control Register (NS)
			-	Non-secure	RAZ/WI

7.3.2 Low-power operation

If the Cortex®-M33 *Floating Point Unit* (FPU) is in a separate power domain, the way the FPU domain is powered down depends on whether the FPU domain includes state retention logic.

To power down the FPU:

- If FPU domain includes state retention logic, disable the FPU by clearing the CPACR.CP10 and CPACR.CP11 bitfields.
- If FPU domain does not include state retention logic, disable the FPU by clearing the CPACR.CP10 and CPACR.CP11 bitfields and set both the CPPWR.SU10 and CPPWR.SU11 bitfields to 1.

Note

Setting the CPPWR.SU10 and CPPWR.SU11 bitfields indicates that FPU state can be lost.

---

8. External coprocessors

This chapter describes the external coprocessors.

8.1 About external coprocessors

The Cortex®-M33 processor supports an external coprocessor interface which allows the integration of tightly coupled accelerator hardware with the processor. The programmers model allows software to communicate with the hardware using architectural coprocessor instructions.

The external coprocessor interface:

- Supports up to eight separate coprocessors, CP0-CP7, depending on your implementation. The remaining coprocessor numbers, C8-C15, are reserved. CP10 and CP11 are always reserved for hardware floating-point. For more information, see the *Arm®v8-M Architecture Reference Manual*.
- Supports low-latency data transfer from the processor to and from the accelerator components.
- Has a sustained bandwidth up to twice of the processor memory interface.

For each coprocessor CP0-CP7, the encoding space can be dedicated to either the external coprocessor or the *Custom Datapath Extension* (CDE) modules. See 9. Arm Custom Instructions on page 60 for information on the CDE implementation in the processor.

8.2 Operation

The following instruction types are supported:

- Register transfer from the Cortex®-M33 processor to the coprocessor `MCR`, `MCRR`, `MCR2`, `MCRR2`.
- Register transfer from the coprocessor to the Cortex®-M33 processor `MRC`, `MRRC`, `MRC2`, `MRRC2`.
- Data processing instructions `CDP`, `CDP2`.

Note

The regular and extension forms of the coprocessor instructions for example, `MCR` and `MCRR2`, have the same functionality but different encodings.
The `MRC` and `MRC2` instructions support the transfer of APSR.NZVC flags when the processor register field is set to PC, for example Rt == `0xF`.

8.3 Usage restrictions

The following restrictions apply when the Cortex®-M33 processor uses coprocessor instructions:

- The `LDC(2)` or `STC(2)` instructions are not supported. If these are included in software with the <coproc> field set to a value between 0-7 and the coprocessor is present and enabled in the appropriate fields in the CPACR/NSACR registers, the Cortex®-M33 processor always attempts to take an *Undefined instruction* (UNDEFINSTR) UsageFault exception.
- The processor register fields for data transfer instructions must not include the stack pointer (Rt = `0xD`), this encoding is **UNPREDICTABLE** in the Arm®v8-M architecture and results in an UNDEFINSTR UsageFault exception in Cortex®-M33 if the coprocessor is present and enabled in the CPACR/NSACR registers.
- If any coprocessor instruction is executed when the corresponding coprocessor is either not present or disabled in the CPACR/NSACR register, the Cortex®-M33 processor always attempts to take a *No coprocessor* (NOCP) UsageFault exception.

8.4 Data transfer rates

The following table shows the ideal data transfer rates for the coprocessor interface. This means that the coprocessor responds immediately to an instruction. The ideal data transfer rates are sustainable if the corresponding coprocessor instructions are executed consecutively.

Table 8-1: Data transfer rates

Instructions	Direction	Ideal data rate
`MCR`, `MCR2`	Processor to coprocessor	32 bits per cycle
`MRC`, `MRC2`	Coprocessor to processor	32 bits per cycle
`MCRR`, `MCRR2`	Processor to coprocessor	64 bits per cycle
`MRRC`, `MRRC2`	Coprocessor to processor	64 bits per cycle

8.5 Configuring which coprocessors are included in Secure and Non-secure states

If the Cortex®-M33 processor is configured with the Arm®v8-M Security extension, then it can support systems where coprocessors are only accessible from Secure state or from both Secure and Non-secure states.

Software can discover which coprocessors are available by accessing the CPACR and NSACR registers in the SCS memory region as documented in the *Arm®v8-M Architecture Reference Manual*.

The following table shows the relationship between the coprocessor security type and the access control registers.

Table 8-2: Coprocessor security type and access control registers

Coprocessor n security type	CPACR[2n+1:2n]		NSACR[n]
	From Secure	From Non-secure	
Not present	RAZ/WI	RAZ/WI	RAZ/WI
Available in Secure only	RW, reset to 0	RAZ/WI	RAZ/WI
Available in Secure and Non-secure	RW, reset to 0	RW, reset to 0	UNKNOWN

Note

- From coprocessors which can be accessed in Secure and Non-secure state the Secure software can further restrict access from Non-secure by using the NSACR register.
- If the Cortex®-M33 processor is not configured with the Arm®v8-M Security Extension, CPACR[2n+1:2n] is RAZ/WI. However, in Non-secure mode, the coprocessor is still available, therefore, NSACR is UNKNOWN.

Using a coprocessor instruction for a coprocessor which is not accessible in the current security state results in a NOCP UsageFault exception.

8.6 Debug access to coprocessor registers usage constraints

The Cortex®-M33 processor does not support a mechanism to read and write registers located in external coprocessors from a debugger.

Arm® recommends you implement a coprocessor with a dedicated AHB or APB slave interface for the system to access the registers. If the debug view of the coprocessor is located in the PPB region of the memory map, you can use this interface to connect to the EPPB interface of the Cortex®-M33 processor.

If Secure debug is disabled, you must ensure the Secure information in the coprocessors is protected and not accessible when using a Non-secure debugger.

If the debug slave interface to the coprocessor is connected to the processor C-AHB or S-AHB master interfaces or the EPPB interface, you can use the HNONSEC and PPROT[2] signals on the AHB and APB interfaces respectively. This is because the security level of the debug requests routed through the processor from the D-AHB interface are subject to the debug access and authentication checks. If the coprocessor state is memory-mapped, then software can also access the information using load and store instructions. If your implementation uses this functionality, you must ensure the appropriate barrier instructions are included to guarantee ordering between coprocessor instructions and load/store operations to the same state.

8.7 Exceptions and context switch

The Cortex®-M33 processor does not include support for automatic save and restore of coprocessor registers on entry and exit to exceptions, unlike the internal processor integer and floating-point registers. Any coprocessor state that must be maintained across a context switch must be carried out by the software that is aware of the coprocessor requirements.

You must ensure that when the coprocessor contains Secure data it cannot be accessed by software running in a Non-secure exception handler.

9. Arm Custom Instructions

This chapter describes the support for *Arm Custom Instructions* (ACIs) and the implementation of the *Custom Datapath Extension* (CDE) in the processor.

9.1 Arm Custom Instructions support

The Cortex®-M33 processor supports *Arm Custom Instructions* (ACIs) and implements the *Custom Datapath Extension* (CDE) for Armv8-M.

The ACI support provides the following:

- New architecturally defined instructions.
- An interface that supports the addition of user-defined instructions.
- Compliance tests to check the integration of the user-defined instructions as part of the execution testbench.

CDE modules

For each coprocessor CP0-CP7, the CDE architecture allows you to choose to either use the coprocessor or bypass it and use CDE modules instead.

The Cortex®-M33 processor includes two CDE modules.

You are responsible for the content of these modules in your implementation. Arm is responsible for the interfaces to these modules.

CDE

The core CDE module executes instructions that access the general-purpose registers. This module is reset and clocked in the same way as the processor core, and it is included in the Core power domain.

FPCDE

The floating-point CDE module executes instructions that access the floating-point registers. This module is reset and clocked in the same way as the *Floating-Point Unit* (FPU), and it is included in the FPU power domain. If the core CDE module is present and used, and if the FPU is present, then the floating-point CDE module is also present.

User-defined instructions

The CDE architecture defines instruction classes depending on the number of source or destination registers. For each class, an accumulation variant exists. You define the function of these instruction classes in the dedicated CDE module added to the processor core or to the FPU.

The classes are:

CX1, CX2, CX3

These three classes operate on the general-purpose register file, including the condition code flags APSR_nzcv.

You can define different functions for a given instruction class depending on the coprocessor number and the opcode value `<imm>`.

VCX1, VCX2, VCX3

These three classes operate on the floating-point register file only.

You can define different functions for a given instruction class depending on the coprocessor number and the opcode value `<imm>`.

ACI support in multi-Cortex®-M33 systems with different CDE customization

In a system with several Cortex®-M33 processors, it is possible to configure a different CDE customization for each processor using the `CDERTLID` parameter. This parameter can be used to implement different functions for an identical instruction.

Software can read the `CDERTLID` parameter using the ID_AFR0, Auxiliary Feature Register 0. See 4.4 Auxiliary Feature Register 0 on page 43.

9.2 Operation

The architecture extension defines instruction classes that depend on the number of source or destination registers. For each class, an accumulation variant exists.

The processor supports the following CDE instruction classes that access the general-purpose registers and APSR_nzcv flags:

- `CX1{A} {cond}, <coproc>, <Rd>, #<imm>`.
- `CX1D{A} {cond}, <coproc>, <Rd>, <Rd+1>, #<imm>`.
- `CX2{A} {cond}, <coproc>, <Rd>, <Rn>, #<imm>`.
- `CX2D{A} {cond}, <coproc>, <Rd>, <Rd+1>, <Rn>, #<imm>`.
- `CX3{A} {cond}, <coproc>, <Rd>, <Rn>, <Rm>, #<imm>`.
- `CX3D{A} {cond}, <coproc>, <Rd>, <Rd+1>, <Rn>, <Rm>, #<imm>`.

Where:

- `{A}` indicates an accumulation variant.
- `{cond}` indicates an optional condition code.
- `<coproc>` indicates the name of the coprocessor the instruction is for.
- `<Rd>` is the general-purpose R0 - R14 or APSR_nzcv destination register, encoded in the "Rd" field.
- `<Rn>` is the general-purpose R0 - R14 or APSR_nzcv source register, encoded in the "Rn" field.

- `<Rm>` is the general-purpose R0 - R14 or APSR_nzcv source register, encoded in the "Rm" field.
- `<imm>` is the immediate value.

The immediate size, `#<imm>`, differs for each instruction class:

CX1	[12:0]
CX2	[8:0]
CX3	[5:0]

As architecturally defined:

- Double variants imply that `Rd` is 64 bits and saved into `Rd` and `Rd+1`, while `Rn` and `Rm` are 32 bits.
- Single variants imply that `Rd`, `Rn`, and `Rm` are 32 bits.

The processor supports the following CDE instruction classes that access the floating-point registers:

- `VCX1{A} <coproc>, <Sd>, #<imm>`.
- `VCX2{A} <coproc>, <Sd>, <Sm>, #<imm>`.
- `VCX3{A} <coproc>, <Sd>, <Sn>, <Sm>, #<imm>`.

Where:

- `<Sd>` is the 32-bit name of the floating-point source and destination register S0 - S31.
- `<Sm>` is the 32-bit name of the floating-point source and destination register S0 - S31.
- `<Sn>` is the 32-bit name of the floating-point source and destination register S0 - S31.
- `<imm>` is the immediate value.

The immediate size, `#<imm>`, differs for each instruction class:

VCX1	[10:0]
VCX2	[5:0]
VCX3	[2:0]

Some restrictions apply when using these instructions, see 8.3 Usage restrictions on page 56 for information.

9.3 Usage restrictions

Some restrictions apply when the Cortex®-M33 processor uses *Custom Datapath Extension* (CDE) instructions.

Depending on your processor implementation at hardware and software level and on your implementation of the CDE and FPCDE modules, NOCP or UNDEFINSTR exceptions might occur when *Arm Custom Instructions* (ACIs) are in use.

When the `FPU` parameter is set to 0 and the FPU is not included, CDE instructions that work on floating-point registers cannot be executed and result in a NOCP UsageFault exception.

The following tables show the usage restrictions that are specific to instruction classes.

Table 9-1: Usage restrictions applicable to instructions classes for the CDE module

Instruction class	Restriction
`CX1{A} {cond}, <coproc>, <Rd>, #<imm>`	The architectural **UNPREDICTABLE** case where d==13 results in an **UNDEFINED** exception. No stack limit check is performed. The architectural **UNPREDICTABLE** case with InITBlock() for non-accumulation variants results in an **UNDEFINED** exception.
`CX1D{A} {cond}, <coproc>, <Rd>, <Rd+1>, #<imm>`	If d is odd, then the instruction is **UNDEFINED**. The architectural **UNPREDICTABLE** case where d==12 results in an **UNDEFINED** exception. No stack limit check is performed. The architectural **UNPREDICTABLE** case with InITBlock() for non-accumulation variants results in an **UNDEFINED** exception.
`CX2{A} {cond}, <coproc>, <Rd>, <Rn>, #<imm>`	The architectural **UNPREDICTABLE** case where d==13 or n==13 results in an **UNDEFINED** exception. No stack limit check is performed. The architectural **UNPREDICTABLE** case with InITBlock() for non-accumulation variants results in an **UNDEFINED** exception.
`CX2D{A} {cond}, <coproc>, <Rd>, <Rd+1>, <Rn>, #<imm>`	If d is odd, then the instruction is **UNDEFINED**. The architectural **UNPREDICTABLE** case where d==12 or n==13 results in an **UNDEFINED** exception. No stack limit check is performed. The architectural **UNPREDICTABLE** case with InITBlock() for non-accumulation variants results in an **UNDEFINED** exception.
`CX3{A} {cond}, <coproc>, <Rd>, <Rn>, <Rm>, #<imm>`	The architectural **UNPREDICTABLE** case where d==13 or n==13 or m=13 results in an **UNDEFINED** exception. No stack limit check is performed. The architectural **UNPREDICTABLE** case with InITBlock() for non-accumulation variants results in an **UNDEFINED** exception.

Instruction class	Restriction
CX3D{A} {cond}, <coproc>, <Rd>, <Rd+1>, <Rn>, <Rm>, #<imm>	If d is odd, then the instruction is **UNDEFINED**. The architectural **UNPREDICTABLE** case where d==12 or n==12 or m==12 results in an **UNDEFINED** exception. No stack limit check is performed. The architectural **UNPREDICTABLE** case with InITBlock() for non-accumulation variants results in an **UNDEFINED** exception.

Table 9-2: Usage restrictions applicable to instructions classes for the FPCDE module

Instruction class	Restriction
VCX1{A} <coproc>, <Sd>, #<imm>	The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.
VCX1{A} <coproc>, <Dd>, #<imm>	The processor does not support the *M-profile Vector Extension* (MVE). Attempting to execute some double-register operation results in an **UNDEFINED** exception. The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.
VCX1{A} <coproc>, <Qd>, #<imm>	The processor does not support MVE. Attempting to execute some quadword-register operation results in an **UNDEFINED** exception. The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.
VCX2{A} <coproc>, <Sd>, <Sm>, #<imm>	The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.
VCX2{A} <coproc>, <Dd>, <Dm>, #<imm>	The processor does not support MVE. Attempting to execute some double-register operation results in an **UNDEFINED** exception. The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.
VCX2{A} <coproc>, <Qd>, <Qm>, #<imm>	The processor does not support MVE. Attempting to execute some quadword-register operation results in an **UNDEFINED** exception. The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.
VCX3{A} <coproc>, <Sd>, <Sn>, <Sm>, #<imm>	The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.
VCX3{A} <coproc>, <Dd>, <Dn>, <Dm>, #<imm>	The processor does not support MVE. Attempting to execute some double-register operation results in an **UNDEFINED** exception. The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.

Instruction class	Restriction
`VCX3{A} <coproc>, <Qd>, <Qn>, <Qm>, #<imm>`	The processor does not support MVE. Attempting to execute some quadword-register operation results in an **UNDEFINED** exception. The architectural **UNPREDICTABLE** case with InITBlock() results in an **UNDEFINED** exception.

See the *Arm®v8-M Architecture Reference Manual* for information on InITBlock().

10. Debug

This chapter summarizes the debug system.

10.1 Debug functionality

Cortex®-M33 debug functionality includes processor halt, single-step, processor core register access, Vector Catch, unlimited software breakpoints, and full system memory access.

The processor also includes support for hardware breakpoints and watchpoints configured during implementation:

- A breakpoint unit supporting four to eight instruction comparators.
- A watchpoint unit supporting two or four data watchpoint comparators.

The Cortex®-M33 processor supports system level debug authentication to control access from a debugger to resources and memory. If the Arm®v8-M Security Extension is included, the authentication can be used to allow a debugger full access to Non-secure code and data without exposing any Secure information.

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

All debug registers are accessible by the D-AHB interface.

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

10.1.1 CoreSight™ discovery

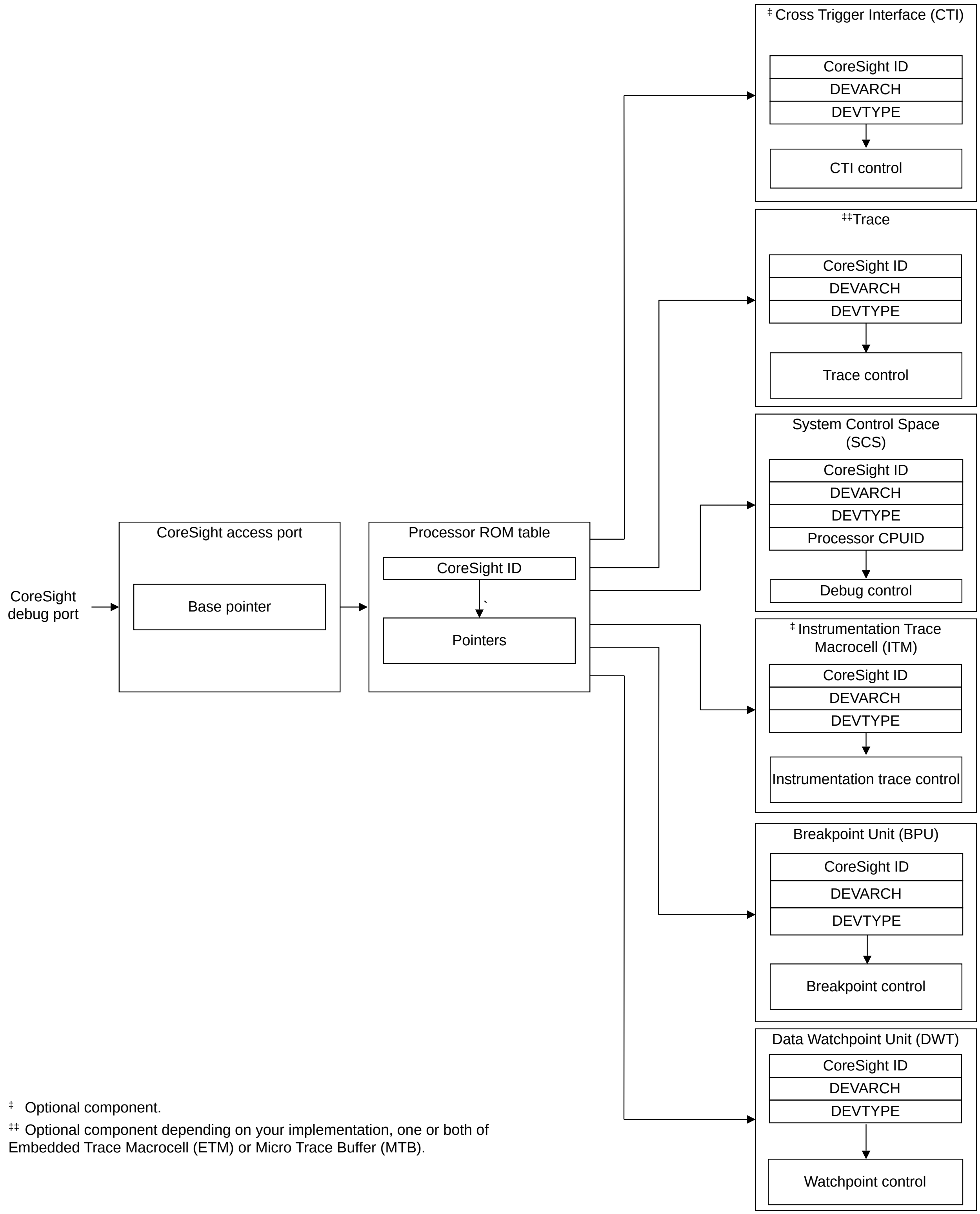
For processors that implement debug, Arm® recommends that a debugger identifies and connects to the debug components using the CoreSight™ debug infrastructure.

See the *CoreSight™ Components Technical Reference Manual* for more information.

Arm® recommends that a debugger follows the flow in the following figure to discover the components present in the CoreSight™ debug infrastructure. In this case, for each CoreSight™ component in the CoreSight™ system, a debugger reads:

- The peripheral and component ID registers.
- The DEVARCH and DEVTYPE registers.

Figure 10-1: CoreSight™ discovery



To identify the Cortex®-M33 processor and debug components within the CoreSight™ system, Arm® recommends that a debugger perform the following actions:

1. Locate and identify the Cortex®-M33 Processor ROM table using its CoreSight™ identification.
2. Follow the pointers in the Cortex®-M33 Processor ROM table to identify the presence of the following components:
    a. *Cross Trigger Interface* (CTI).
    b. *Embedded Trace Macrocell* (ETM)
    c. *Micro Trace Buffer* (MTB).
    d. *System Control Space* (SCS).
    e. *Instrumentation Trace Macrocell* (ITM).
    f. *Breakpoint Unit* (BPU).
    g. *Data Watchpoint Unit* (DWT).

10.1.2 Debugger actions for identifying the processor

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

A debugger cannot rely on the Cortex®-M33 Processor ROM table being the first ROM table encountered. One or more system ROM tables might be included between the access port and the processor ROM table if other CoreSight™ components are in the system. If a system ROM table is present, it can include a unique identifier for the implementation.

10.1.3 Processor ROM table identification and entries

The ROM table identification registers and values that the following table shows allow debuggers to identify the processor and its debug capabilities.

Table 10-1: Cortex®-M33 Processor ROM table identification values

Address offset	Register	Value	Description
`0xE00FFFD0`	PIDR4	`0x00000004`	*Component and Peripheral ID register formats* in the *Arm®v8-M Architecture Reference Manual*
`0xE00FFFD4`	PIDR5	`0x00000000`	
`0xE00FFFD8`	PIDR6	`0x00000000`	
`0xE00FFFDC`	PIDR7	`0x00000000`	
`0xE00FFFE0`	PIDR0	`0x000000C9`	
`0xE00FFFE4`	PIDR1	`0x000000B4`	
`0xE00FFFE8`	PIDR2	`0x0000000B`	
`0xE00FFFEC`	PIDR3	`0x00000000`[1]	
`0xE00FFFF0`	CIDR0	`0x0000000D`	
`0xE00FFFF4`	CIDR1	`0x00000010`	
`0xE00FFFF8`	CIDR2	`0x00000005`	

Address offset	Register	Value	Description
0xE00FFFFC	CIDR3	0x000000B1	

These values for the Peripheral ID registers identify this as the Cortex®-M33 Processor ROM table. The Component ID registers identify this as a CoreSight™ ROM table.

Note

The Cortex®-M33 Processor ROM table only supports word-size transactions.

The following table shows the CoreSight™ components that the Cortex®-M33 Processor ROM table points to.

Table 10-2: Cortex®-M33 Processor ROM table components

Address	Component	Value	Description
0xE00FF000	SCS	0xFFF0F003.	See System Control.
0xE00FF004	DWT	0xFFF02003. Reads as 0xFFF02002 if the DWT is not implemented.	See DWT
0xE00FF008	BPU	0xFFF03003. Reads as 0xFFF03002 if the BPU is not implemented.	See BPU
0xE00FF00C	ITM	0xFFF01003. Reads as 0xFFF01002 if the ITM is not implemented.	See ITM.
0xE00FF014	ETM	0xFFF42003. Reads as 0xFFF42002 if the ETM is not implemented.	See the *Arm® CoreSight™ ETM-Cortex®-M33 Technical Reference Manual*
0xE00FF018	CTI	0xFFF43003. Reads as 0xFFF43002 if the CTI is not implemented.	See CTI.
0xE00FF01C	MTB	0xFFF44003. Reads as 0xFFF44002 if the MTB is not implemented.	See MTB.
0xE00FF020-0xE00FFFC8	Reserved	0x00000000.	See the *Arm® CoreSight™ Architecture Specification (v2.0)*
0xE00FFFCC	SYSTEM ACCESS	0x00000001.	

The Cortex®-M33 Processor ROM table entries point to the debug components of the processor. The offset for each entry is the offset of that component from the ROM table base address, 0xE00FF000.

See the *Arm® CoreSight™ Architecture Specification (v2.0)* for more information about the ROM table ID and component registers, and access types.

10.1.4 System Control Space registers

The processor provides debug through registers in the *System Control Space* (SCS).

10.1.4.1 SCS CoreSight™ identification

The following table shows the SCS CoreSight™ identification registers and values for debugger detection. Final debugger identification of the Cortex®-M33 processor is through the CPUID register in the SCS.

Table 10-3: SCS identification values

Address offset	Register name	Reset value	Description
0xE000EFD0	DPIDR4	0x00000004	*Component and Peripheral ID register formats* in the *Arm®v8-M Architecture Reference Manual*
0xE000EFD4	DPIDR5	0x00000000	
0xE000EFD8	DPIDR6	0x00000000	
0xE000EFDC	DPIDR7	0x00000000	
0xE000EFE0	DPIDR0	0x00000021	
0xE000EFE4	DPIDR1	0x000000BD	
0xE000EFE8	DPIDR2	0x0000000B	
0xE000EFEC	DPIDR3	0x00000000[1]	
0xE000EFF0	DCIDR0	0x0000000D	
0xE000EFF4	DCIDR1	0x00000090	
0xE000EFF8	DCIDR2	0x00000005	
0xE000EFFC	DCIDR3	0x000000B1	
0xE000EFBC	DDEVARCH	0x47702A04	

10.1.5 Debug register summary

The following table shows the debug registers, with address, name, type, reset, and description information for each register.

Each register is 32-bits wide and is described in the *Arm®v8-M Architecture Reference Manual*.

Table 10-4: Debug registers

Address offset	Name	Type	Reset value	Processor security state	Description
0xE000ED30	DFSR	RW	0x00000000 Power-on reset only.	Secure	Debug Fault Status Register
				Non-secure	
0xE002ED30	DFSR_NS	RW	0x00000000	Secure	Debug Fault Status Register (NS)
			-	Non-secure	RAZ/WI

Address offset	Name	Type	Reset value	Processor security state	Description
0xE000EDF0	DHCSR	RW	0x00000000	Secure	Debug Halting Control and Status Register
				Non-secure	
0xE002EDF0	DHCSR_NS	RW	0x00000000	Secure	Debug Halting Control and Status Register (NS)
			-	Non-secure	RAZ/WI
0xE000EDF4	DCRSR	WO	UNKNOWN	Secure	Debug Core Register Selector Register
				Non-secure	
0xE000EDF8	DCRDR	RW	UNKNOWN	Secure	Debug Core Register Data Register
				Non-secure	
0xE002EDF8	DCRDR_NS	RW	UNKNOWN	Secure	Debug Core Register Data Register (NS)
			-	Non-secure	RAZ/WI
0xE000EDFC	DEMCR	RW	0x00000000	Secure	Debug Exception and Monitor Control Register
				Non-secure	
0xE000EDFC	DEMCR_NS	RW	0x00000000	Secure	Debug Exception and Monitor Control Register (NS)
			-	Non-secure	RAZ/WI
0xE000EE04	DAUTHCTRL	RW	0x00000000	Secure	Debug Authentication Control Register
				Non-secure	
0xE002EE04	DAUTHCTRL_NS	RW	0x00000000	Secure	Debug Authentication Control Register (ns)
			-	Non-secure	RAZ/WI
0xE000EE08	DSCSR	RW	0x00000000	Secure	Debug Security Control and Status Register
				Non-secure	
0xE000EFB8	DAUTHSTATUS	RO	UNKNOWN[11]	Secure	Debug Authentication Status Register
				Non-secure	
0xE002EFB8	DAUTHSTATUS_NS	RO	UNKNOWN[11]	Secure	Debug Authentication Status Register (ns)
				Non-secure	RAZ/WI

10.2 About the D-AHB interface

The 32-bit *Debug AHB* (D-AHB) interface implements the AMBA® 5 AHB protocol. It can be used with a CoreSight™ AHB-AP to provide debugger access to all processor control and debug

[11] The value of DAUTHSTATUS at reset is dependent on the debug authentication defined in the system and whether the Arm®v8-M Security Extension is included in the processor. *Arm®v8-M Architecture Reference Manual* for more information.

resources, and a view of memory that is consistent with that observed by load/store instructions acting on the processor.

Access to all resources from the debugger can be controlled by system level debug authentication supported by the processor. If the Arm®v8-M Security Extension is included, the authentication can prevent a debugger from accessing any Secure data or code while providing full access to Non-secure information.

The accesses to individual registers and memory might be restricted according to the debug authorization that your system uses.

D-AHB interface accesses are only in little-endian format.

11. Instrumentation Trace Macrocell Unit

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

11.1 ITM programmers model

This is a summary of the ITM register table, and characteristics and bit assignments of the ITM registers.

11.1.1 ITM register summary table

The following table shows the ITM registers whose implementation is specific to this processor.

Other registers are described in the *Arm®v8-M Architecture Reference Manual*.

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

Note

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

Table 11-1: ITM register summary

Address	Name	Type	Reset	Description
`0xE0000000-` `0xE000007C`	ITM_STIM0- ITM_STIM31	RW	-	Stimulus Port Registers 0-31
`0xE0000E00`	ITM_TER	RW	`0x00000000`	Trace Enable Register
`0xE0000E40`	ITM_TPR	RW	`0x00000000`	ITM Trace Privilege Register
`0xE0000E80`	ITM_TCR	RW	`0x00000000`	Trace Control Register
`0xE0000EF0`	INT_ATREADY	RO	`0x00000000`	Integration Mode: Read ATB Ready
`0xE0000EF8`	INT_ATVALID	WO	`0x00000000`	Integration Mode: Write ATB Valid
`0xE0000F00`	ITM_ITCTRL	RW	`0x00000000`	Integration Mode Control Register
`0xE0000FCC`	ITM_DEVTYPE	RW	`0x00000043`	ITM CoreSight Device Type Register
`0xE0000FBC`	ITM_DEVARCH	RO	`0x47701A01`	ITM CoreSight Device Architecture Register

Address	Name	Type	Reset	Description
0xE0000FD0	ITM_PIDR4	RO	0x00000004	Peripheral identification registers
0xE0000FD4	ITM_PIDR5	RO	0x00000000	Peripheral identification register
0xE0000FD8	ITM_PIDR6	RO	0x00000000	Peripheral identification register
0xE0000FDC	ITM_PIDR7	RO	0x00000000	Peripheral identification register
0xE0000FE0	ITM_PIDR0	RO	0x00000021	Peripheral identification register
0xE0000FE4	ITM_PIDR1	RO	0x000000BD	Peripheral identification register
0xE0000FE8	ITM_PIDR2	RO	0x0000000B	Peripheral identification register
0xE0000FEC	ITM_PIDR3	RO	0x00000000[1]	Peripheral identification register
0xE0000FF0	ITM_CIDR0	RO	0x0000000D	Component identification register
0xE0000FF4	ITM_CIDR1	RO	0x00000090	Component identification register
0xE0000FF8	ITM_CIDR2	RO	0x00000005	Component identification register
0xE0000FFC	ITM_CIDR3	RO	0x000000B1	Component identification register

Note

ITM registers are fully accessible in privileged mode. In user mode, all registers can be read, but only the Stimulus registers and Trace Enable registers can be written, and only when the corresponding Trace Privilege Register bit is set. Invalid user mode writes to the ITM registers are discarded. When the Arm®v8-M Security Extension is included in the Cortex®-M33 processor, writes to the Stimulus registers from the software running in Secure state are ignored if Secure non-invasive debug authentication is not enabled.

11.1.2 ITM Trace Privilege Register

The ITM_TPR enables an operating system to control the stimulus ports that are accessible by user code.

Usage constraints

You can only write to this register in privileged mode.

Configurations

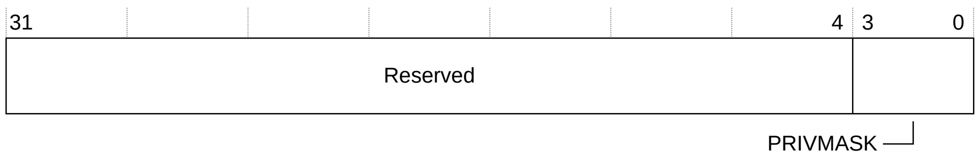
This register is available if the ITM is configured in your implementation.

Attributes

See the ITM register summary table.

The following figure shows the ITM_TPR bit assignments.

Figure 11-1: ITM_TPR bit assignments



The following table shows the ITM_TPR bit assignments.

Table 11-2: ITM_TPR bit assignments

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

11.1.3 ITM Integration Mode Control Register

The ITM_ITCTRL controls whether the trace unit is in integration mode.

Usage constraints

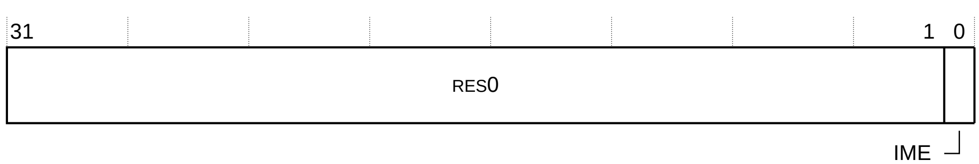
- Accessible only from the memory-mapped interface or from an external agent such as a debugger.
- Arm® recommends that you perform a debug reset after using integration mode.

Configurations Available in all configurations.

Attributes A 32-bit management register. See also the register summary table.

The following figure shows the ITM_ITCTRL bit assignments.

Figure 11-2: ITM_ITCTRL bit assignments



The following table shows the ITM_ITCTRL bit assignments.

Table 11-3: ITM_ITCTRL bit assignments

Bits	Name	Function
[31:1]	-	Reserved, RES0.
[0]	IME	Integration mode enable bit. The possible values are: **0** The trace unit is not in integration mode. **1** The trace unit is in integration mode. This mode enables: • A debug agent to perform topology detection. • SoC test software to perform integration testing.

The ITM_ITCTRL register can be accessed through the external debug interface, at address `0xE0000F00`.

11.1.4 Integration Mode Write ATB Valid Register

The Integration Mode Write ATB Valid Register is used for integration test.

Usage constraints There are no usage constraints.

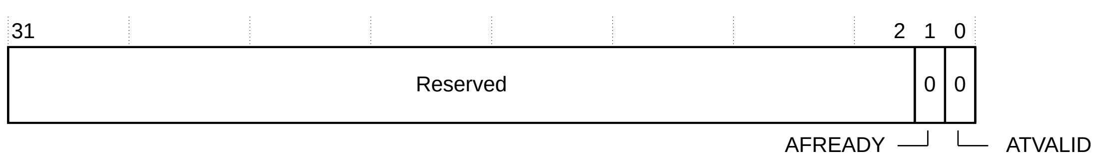
Configurations This register is:

- Only present in integration mode, when ITM_ITCTRL.IME is set to 1.
- Available in all configurations.

Attributes See the register summary table.

The following figure INT_ATVALID shows the bit assignments.

Figure 11-3: INT_ATVALID bit assignments



The following table shows the INT_ATVALID bit assignments.

Table 11-4: INT_ATVALID bit assignments

Bits	Name	Function
[31:2]	Reserved	RES0
[1]	AFREADY	When ITM_ITCTRL.IME is set, the value of this bit determines the value of AFREADYI.
[0]	ATVALID	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of ATVALIDI.

11.1.5 Integration Mode Read ATB Ready Register

The Integration Mode Read ATB Ready Register, INT_ATREADY, is used for integration test.

Usage constraints There are no usage constraints.

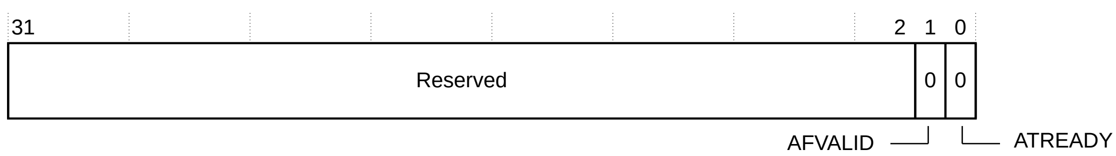
Configurations This register is:

- Only present in integration mode, when ITM_ITCTRL.IME is set to 1.
- Available in all configurations.

Attributes See the register summary table.

The following figure INT_ATREADY shows the bit assignments.

Figure 11-4: INT_ATREADY bit assignments



The following table shows the INT_ATREADY bit assignments.

Table 11-5: INT_ATREADY bit assignments

Bits	Name	Function
[1]	AFVALID	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of AFVALIDI. When ITM_ITCTRL.IME is not set, this bit returns zero.
[0]	ATREADY	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of ATREADYI. When ITM_ITCTRL.IME is not set, this bit returns zero.

12. Data Watchpoint and Trace Unit

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

12.1 DWT functional description

A Reduced DWT contains two comparators (DWT_COMP0 to DWT_COMP1) and a Full DWT contains four comparators (DWT_COMP0 to DWT_COMP3). These comparators support the following features:

- Hardware watchpoint support.
- Hardware trace packet support, only if your implementation includes an ITM.
- CMPMATCH support for ETM/MTB/CTI triggers (only if your implementation includes an ETM, MTB, or CTI).
- Cycle counter matching support (DWT_COMP0 only).
- Instruction address matching support.
- Data address matching support.
- Data value matching support (DWT_COMP1 only in a reduced DWT, DWT_COMP3 only in a Full DWT).
- Linked/limit matching support (DWT_COMP1 and DWT_COMP3 only).

The DWT contains counters for:

- Cycles (CYCCNT).
- Folded Instructions (FOLDCNT).
- Additional cycles required to execute all load or store instructions (LSUCNT).
- Processor sleep cycles (SLEEPCNT).
- Additional cycles required to execute multi-cycle instructions and instruction fetch stalls (CPICNT)
- Cycles spent in exception processing (EXCCNT).

You can configure the DWT to generate PC samples at defined intervals, and to generate interrupt event information. Before using DWT, the TRCENA bit in the DEMCR register should be set to 1.

The DWT provides periodic requests for protocol synchronization to the ITM and the TPIU, if your implementation includes the Cortex®-M33 TPIU.

Related information

Trace Port Interface Unit on page 109

12.2 DWT programmers model

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

Table 12-1: DWT register summary

Address offset	Name	Type	Reset value	Description
0xE0001000	DWT_CTRL	RW	Possible reset values are: **0x2800000** Reduced DWT with no ITM trace. **0x2000000** Reduced DWT with ITM trace. **0x4800000** Full DWT with no ITM trace. **0x4000000** Full DWT with ITM trace.	Control Register.
0xE0001004	DWT_CYCCNT	RW	0x00000000	Cycle Count Register
0xE0001008	DWT_CPICNT	RW	-	CPI Count Register
0xE000100C	DWT_EXCCNT	RW	-	Exception Overhead Count Register
0xE0001010	DWT_SLEEPCNT	RW	-	Sleep Count Register
0xE0001014	DWT_LSUCNT	RW	-	LSU Count Register
0xE0001018	DWT_FOLDCNT	RW	-	Folded-instruction Count Register
0xE000101C	DWT_PCSR	RO	-	Program Counter Sample Register
0xE0001020	DWT_COMP0	RW	-	Comparator Register0
0xE0001028	DWT_FUNCTION0	RW	0x58000000	Function Register0
0xE0001030	DWT_COMP1	RW	-	Comparator Register1
0xE0001038	DWT_FUNCTION1	RW	Possible reset values are: **0xF0000000** Reduced DWT. **0xD0000000** Full DWT.	Function Register1
0xE0001040	DWT_COMP2	RW	-	Comparator Register2
0xE0001048	DWT_FUNCTION2	RW	0x50000000	Function Register2
0xE0001050	DWT_COMP3	RW	-	Comparator Register3
0xE0001058	DWT_FUNCTION3	RW	Possible reset values are: **0x50000000** Reduced DWT. **0xF0000000** Full DWT.	Function Register3
0xE0001FBC	DWT_DEVARCH	RO	0x47701A02	Device Type Architecture register
0xE0001FCC	DWT_DEVTYPE	RO	0x00000000	Device Type Identifier register
0xE0001FD0	DWT_PID4	RO	0x00000004	Peripheral identification registers
0xE0001FD4	DWT_PID5	RO	0x00000000	
0xE0001FD8	DWT_PID6	RO	0x00000000	
0xE0001FDC	DWT_PID7	RO	0x00000000	
0xE0001FE0	DWT_PIDR0	RO	0x00000021	
0xE0001FE4	DWT_PIDR1	RO	0x000000BD	
0xE0001FE8	DWT_PIDR2	RO	0x0000000B	

Address offset	Name	Type	Reset value	Description
0xE0001FEC	DWT_PIDR3	RO	0x00000000[1]	
0xE0001FF0	DWT_CIDR0	RO	0x0000000D	Component identification registers
0xE0001FF4	DWT_CIDR1	RO	0x00000090	
0xE0001FF8	DWT_CIDR2	RO	0x00000005	
0xE0001FFC	DWT_CIDR3	RO	0x000000B1	

DWT registers are described in the *Arm®v8-M Architecture Reference Manual*. Peripheral Identification and Component Identification registers are described in the *CoreSight™ Components Technical Reference Manual*.

13. Cross Trigger Interface

This chapter describes the *Cross Trigger Interface* (CTI).

13.1 About the Cross Trigger Interface

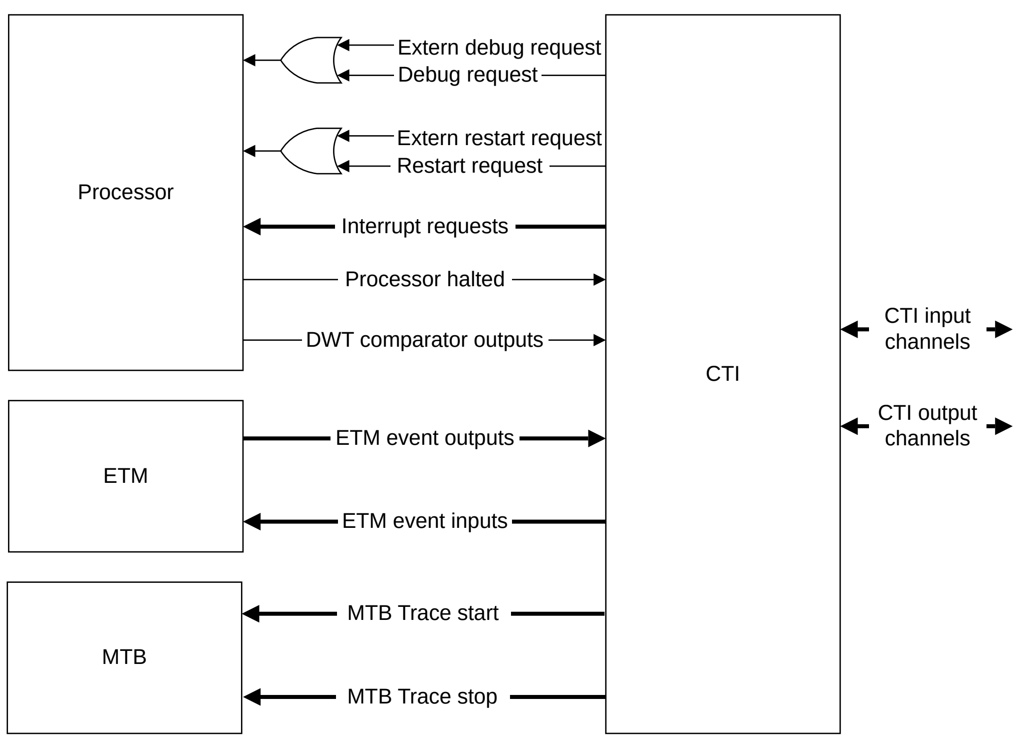
If implemented, the CTI enables the debug logic, MTB, and ETM to interact with each other and with other CoreSight™ components. This is called cross triggering. For example, you can configure the CTI to generate an interrupt when the ETM trigger event occurs or to start tracing when a DWT comparator match is detected.

13.2 CTI functional description

The Cortex®-M33 CTI interacts with several debug system components, and is connected to various trigger inputs and trigger outputs.

The following figure shows the debug system components and the available trigger inputs and trigger outputs.

Figure 13-1: Debug system components



The following table shows how the CTI trigger inputs are connected to the Cortex®-M33 processor.

Table 13-1: Trigger signals to the CTI

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[7]	-	ETM to CTI	Pulsed
CTITRIGIN[6]	-		
CTITRIGIN[5]	ETM Event Output 1		
CTITRIGIN[4]	ETM Event Output 0 or Comparator Output 3	ETM/Processor to CTI	
CTITRIGIN[3]	DWT Comparator Output 2	Processor to CTI	
CTITRIGIN[2]	DWT Comparator Output 1		
CTITRIGIN[1]	DWT Comparator Output 0		
CTITRIGIN[0]	Processor Halted		

The following table shows how the CTI trigger outputs are connected to the processor and ETM.

Table 13-2: Trigger signals from the CTI

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	ETM Event Input 3	CTI to ETM	Pulsed
CTITRIGOUT[6]	ETM Event Input 2		Pulsed
CTITRIGOUT[5]	ETM Event Input 1 or MTB Trace stop	CTI to ETM or MTB	Pulsed
CTITRIGOUT[4]	ETM Event Input 0 or MTB Trace start		Pulsed
CTITRIGOUT[3]	Interrupt request 1	CTI to system.	Acknowledged by writing to the CTIINTACK register in ISR
CTITRIGOUT[2]	Interrupt request 0		
CTITRIGOUT[1]	Processor Restart	CTI to Processor	Processor Restarted
CTITRIGOUT[0]	Processor debug request		Acknowledged by the debugger writing to the CTIINTACK register

Note

- After the processor is halted using CTI Trigger Output 0, the Processor Debug Request signal remains asserted. The debugger must write to CTIINTACK to clear the halting request before restarting the processor.
- After asserting an interrupt using the CTI Trigger Output 1 or 2, the *Interrupt Service Routine* (ISR) must clear the interrupt request by writing to the CTI Interrupt Acknowledge, CTIINTACK.
- Interrupt requests from the CTI to the system are only asserted when invasive debug is enabled in the processor.

If the CTI is not included in the processor, the trigger signals are tied off internally and the cross trigger functionality between the processor, MTB and ETM is not available.

13.3 CTI programmers model

The following table shows the CTI programmable registers, with address offset, type, and reset value for each register.

See the *Arm® CoreSight™ SoC-400 Technical Reference Manual* for register descriptions.

Table 13-3: CTI register summary

Address offset	Name	Type	Reset value	Description
0xE0042000	CTICONTROL	RW	0x00000000	CTI Control Register
0xE0042010	CTIINTACK	WO	UNKNOWN	CTI Interrupt Acknowledge Register
0xE0042014	CTIAPPSET	RW	0x00000000	CTI Application Trigger Set Register
0xE0042018	CTIAPPCLEAR	RW	0x00000000	CTI Application Trigger Clear Register
0xE004201C	CTIAPPPULSE	WO	UNKNOWN	CTI Application Pulse Register
0xE0042020-0xE004203C	CTIINEN[7:0]	RW	0x00000000	CTI Trigger to Channel Enable Registers
0xE00420A0-0xE00420BC	CTIOUTEN[7:0]	RW	0x00000000	CTI Channel to Trigger Enable Registers
0xE0042130	CTITRIGINSTATUS	RO	0x00000000	CTI Trigger In Status Register
0xE0042134	CTITRIGOUTSTATUS	RO	0x00000000	CTI Trigger Out Status Register
0xE0042138	CTICHINSTATUS	RO	0x00000000	CTI Channel In Status Register
0xE0042140	CTIGATE	RW	0x0000000F	Enable CTI Channel Gate register
0xE0042144	ASICCTL	RW	0x00000000	External Multiplexer Control register
0xE0042EE4	ITCHOUT	WO	UNKNOWN	Integration Test Channel Output register
0xE0042EE8	ITTRIGOUT	WO	UNKNOWN	Integration Test Trigger Output register
0xE0042EF4	ITCHIN	RO	0x00000000	Integration Test Channel Input register
0xE0042F00	ITCTRL	RW	0x00000000	Integration Mode Control register
0xE0042FC8	DEVID	RO	0x00040800	Device Configuration register
0xE0042FBC	DEVARCH	RO	0x47701A14	Device Architecture register
0xE0042FCC	DEVTYPE	RO	0x00000014	Device Type Identifier register
0xE0042FD0	PIDR4	RO	0x00000004	Peripheral ID4 Register
0xE0042FD4	PIDR5	RO	0x00000000	Peripheral ID5 Register
0xE0042FD8	PIDR6	RO	0x00000000	Peripheral ID6 Register
0xE0042FDC	PIDR7	RO	0x00000000	Peripheral ID7 Register
0xE0042FE0	PIDR0	RO	0x00000021	Peripheral ID0 Register
0xE0042FE4	PIDR1	RO	0x000000BD	Peripheral ID1 Register
0xE0042FE8	PIDR2	RO	0x0000000B	Peripheral ID2 Register
0xE0042FEC	PIDR3	RO	0x00000000[1]	Peripheral ID3 Register
0xE0042FF0	CIDR0	RO	0x0000000D	Component ID0 Register
0xE0042FF4	CIDR1	RO	0x00000090	Component ID1 Register
0xE0042FF8	CIDR2	RO	0x00000005	Component ID2 Register
0xE0042FFC	CIDR3	RO	0x000000B1	Component ID3 Register

14. Breakpoint Unit

This section describes the *Breakpoint Unit* (BPU).

14.1 About the Breakpoint Unit

The *Breakpoint Unit* (BPU) with configurable support can implement four or eight hardware breakpoints and up to eight instruction comparators.

The BPU does not support Flash patching. The FP_REMAP register is not implemented and is RAZ/WI.

14.2 BPU programmers model

The following table shows the BPU registers, with address, name, type and reset information for each register.

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

All BPU registers are described in the *Arm®v8-M Architecture Reference Manual*.

Table 14-1: BPU register summary

Address offset	Name	Type	Reset value	Description

`0xE0002000`	FP_CTRL	RW	`0x10000040`
			
			If four instruction comparators are implemented.
			
			`0x10000080`
			
			If eight instruction comparators are implemented.
				FlashPatch Control Register

`0xE0002004`	FP_REMAP	RAZ/WI	-	Flash Patch Remap Register not implemented
`0xE0002008`	FP_COMP0[12]	RW	`0x00000000`	FlashPatch Comparator Register0
`0xE000200C`	FP_COMP1[12]	RW	`0x00000000`	Flash Patch Comparator Register 1
`0xE0002010`	FP_COMP2[12]	RW	`0x00000000`	Flash Patch Comparator Register 2
`0xE0002014`	FP_COMP3[12]	RW	`0x00000000`	Flash Patch Comparator Register 3
`0xE0002018`	FP_COMP4[12]	RW	`0x00000000`	Flash Patch Comparator Register 4
`0xE000201C`	FP_COMP5[12]	RW	`0x00000000`	FlashPatch Comparator Register 5
`0xE0002020`	FP_COMP6[12]	RW	`0x00000000`	Flash Patch Comparator Register 6
`0xE0002024`	FP_COMP7[12]	RW	`0x00000000`	Flash Patch Comparator Register 7
`0xE0002FCC`	FP_DEVTYPE	RO	`0x00000000`	FPB CoreSight Device Type Register

Address offset	Name	Type	Reset value	Description
0xE0002FBC	FP_DEVARCH	RO	0x47701A03	FPB CoreSight Device Architecture Register
0xE0002FD0	FP_PIDR4	RO	0x00000004	Peripheral identification registers
0xE0002FD4	FP_PIDR5	RO	0x00000000	
0xE0002FD8	FP_PIDR6	RO	0x00000000	
0xE0002FDC	FP_PIDR7	RO	0x00000000	
0xE0002FE0	FP_PIDR0	RO	0x00000021	
0xE0002FE4	FP_PIDR1	RO	0x000000BD	
0xE0002FE8	FP_PIDR2	RO	0x0000000B	
0xE0002FEC	FP_PIDR3	RO	0x00000000[1]	
0xE0002FF0	FP_CIDR0	RO	0x0000000D	Component identification registers
0xE0002FF4	FP_CIDR1	RO	0x00000090	
0xE0002FF8	FP_CIDR2	RO	0x00000005	
0xE0002FFC	FP_CIDR3	RO	0x000000B1	

14.3 BPU functional description

The BPU contains both a global enable and individual enables for each of the comparators implemented.

If the BPU supports only four breakpoints, only comparators 0-3 are used, and comparators 4-7 are implemented as RAZ/WI.

[12] FP_COMPn[0] is reset to 0.

FP_COMPn[31:1] is reset to **UNKNOWN**

If only 4 breakpoints are implemented, FP_COMP4-FP_COMP7 are RAZ/WI.

Appendix A Debug Access Port

This appendix describes the DAP for the Cortex®-M33 processor.

A.1 About the Debug Access Port

The Cortex®-M33 DAP (Verilog module name `TEALDAP`) is an optional component that provides an interface to allow off-chip debug tools to access the Cortex®-M33 processor.

It is an implementation of the *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* .

The key features of the Cortex®-M33 DAP are:

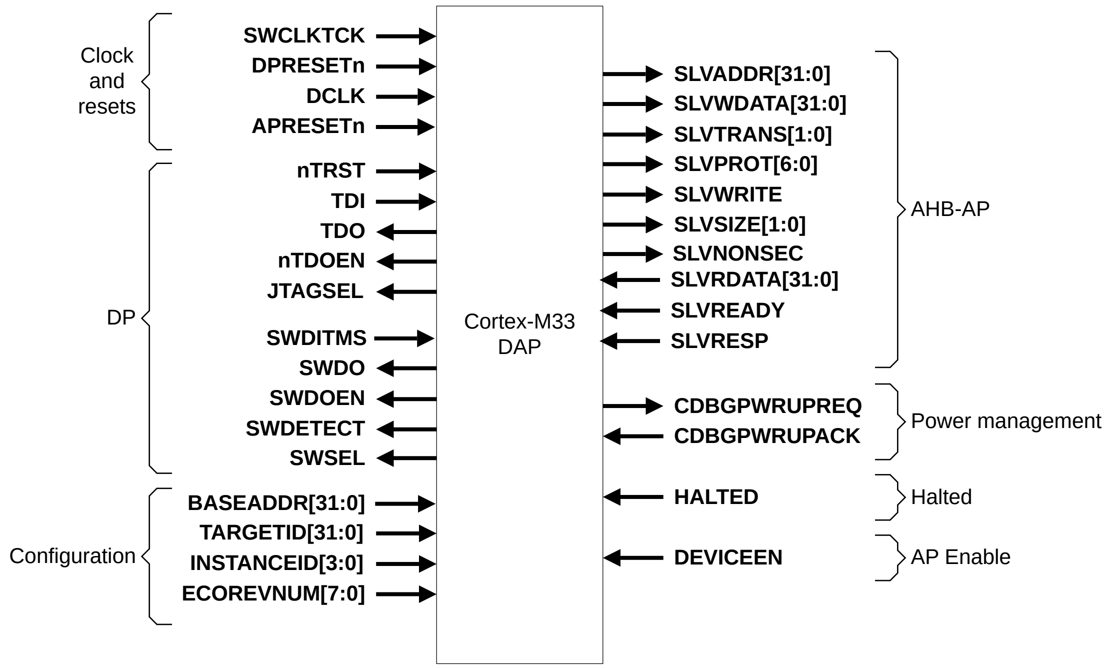
- It can be configured as a *JTAG Debug Port* (JTAG-DP), *Serial Wire Debug Port* (SW-DP), or *Serial Wire/JTAG Debug Port* (SWJ-DP) via an implementation option, see A.1.1 Configuration options on page 87. For more information on the various debug ports, see the *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* .
- Includes an AHB-AP, intended to be directly connected to the Cortex®-M33 processor D-AHB slave port.
- Implements the Minimal Debug Port programmers model, see the *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* .

Note

The Cortex®-M33 DAP is a low gate-count DAP implementation. If you require a full DAP implementation, Arm recommends using the DAP provided in the CoreSight SoC-400 product. See *Arm® CoreSight™ SoC-400 Technical Reference Manual*.

The following figure shows the Cortex®-M33 DAP interface, as specified in the `TEALDAP` Verilog module.

Figure A-1: Cortex®-M33 DAP interface



A.1.1 Configuration options

The following table shows the configuration options for the Cortex®-M33 DAP that can be set at implementation time.

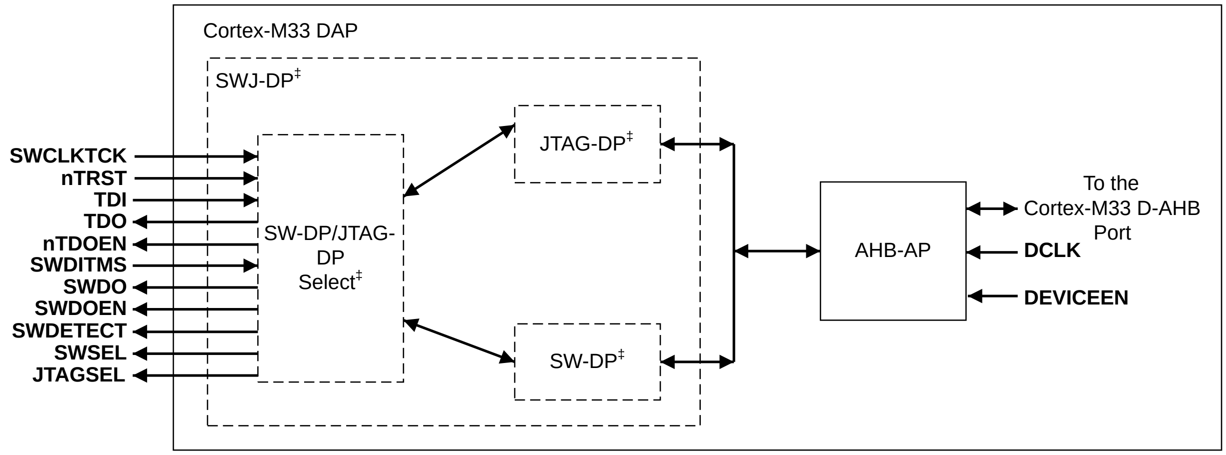
Table A-1: Configuration options for the Cortex®-M33 DAP

Parameter	Description
`DPSEL`	Debug port select: **0** JTAG-DP. **1** SW-DP. **2** SWJ-DP.
`RAR`	Reset all registers: **0** Only required registers are reset. **> 0** All registers are reset.

A.2 Functional description

The following figure shows the main functional blocks in the Cortex®-M33

Figure A-2: Cortex®-M33 block diagram



‡ Optional component.

The *Debug Port* (DP)s and *Access Port* (AP) are compliant with ADIv5.2 architecture. An overview of each is as follows:

JTAG-DP

The JTAG-DP implements the JTAG debug interface and is compliant with DP architecture version 1.

SW-DP

The SW-DP implements the Serial Wire debug interface and is compliant with DP architecture version 2 and Serial Wire protocol version 2.

SWJ-DP

The SWJ-DP implements both the JTAG-DP and SW-DP. The SWJ-DP provides a mechanism to dynamically switch between the debug ports as described in *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* .

AHB-AP

The AHB-AP is an AHB master interface that is intended to be directly connected to the Cortex®-M33 processor D-AHB port. It is compliant with the MEM-AP definition and performs 8-bit, 16-bit, and 32-bit accesses.

The Dormant mode, and the switching to and from the Dormant mode, is supported in all configurations.

A.3 DAP register summary

This section shows the DAP component register summaries.

A.3.1 AHB-AP register summary

The following table shows the AHB-AP register summary.

Table A-2: AHB-AP register summary

Offset	Type	Reset value	Name
0x00	RW	0x03000000	A.4.1.1 AHB-AP Control/Status Word register, CSW, 0x00 on page 90
0x04	RW	-	A.4.1.2 AHB-AP Transfer Address Register, TAR, 0x04 on page 93
0x08	-	-	Reserved, RAZ/SBZP
0x0C	RW	-	A.4.1.3 AHB-AP Data Read/Write register, DRW, 0x0C on page 94
0x10	RW	-	A.4.1.4 AHB-AP Banked Data registers, BD0-BD03, 0x10-0x1C on page 94
0x14	RW	-	
0x18	RW	-	
0x1C	RW	-	
0x20-0xF3	-	-	Reserved, RAZ/SBZP
0xF4	RO	0x00000000	A.4.1.6 AHB-AP Configuration register, CFG, 0xF4 on page 95
0xF8	RO	IMPLEMENTATION DEFINED	A.4.1.5 AHB-AP Debug Base Address register, ROM, 0xF8 on page 95
0xFC	RO	0x04770051	A.4.1.7 AHB-AP Identification Register, IDR, 0xFC on page 96

A.3.2 Debug port register summary

The following table shows the Cortex®-M33 DP registers, and summarizes which registers are implemented in the JTAG-DP and which are implemented in the SW-DP.

Table A-3: Debug port register summary

Name	JTAG-DP	SW-DP	Description
ABORT	Yes	Yes	AP Abort register. See A.4.2.1 AP Abort register, ABORT on page 97.
IDCODE	Yes	No	ID Code register. See A.4.2.2 Identification Code register, IDCODE on page 98.
DPIDR	Yes	Yes	Debug Port Identification register. See A.4.2.3 Debug Port Identification Register, DPIDR on page 99.
CTRL/STAT	Yes	Yes	Control/Status register. See A.4.2.4 Control/Status register, CTRL/STAT on page 100.
SELECT	Yes	Yes	AP Select register. See A.4.2.5 AP Select register, SELECT on page 102.
RDBUFF	Yes	Yes	Read Buffer register. See A.4.2.6 Read Buffer register, RDBUFF on page 104.
EVENTSTAT	No	Yes	Event Status register. See A.4.2.7 Event Status register, EVENTSTAT on page 104.
DLCR	No	Yes	Data Link Control Register. See A.4.2.8 Data Link Control Register, DLCR (SW-DP only) on page 105.
TARGETID	No	Yes	Target Identification register. See A.4.2.9 Target Identification register, TARGETID (SW-DP only) on page 106.

Name	JTAG-DP	SW-DP	Description
DLPIDR	No	Yes	Data Link Protocol Identification Register. See A.4.2.10 Data Link Protocol Identification Register, DLPIDR (SW-DP only) on page 107.
RESEND	No	Yes	Read Resend register. See A.4.2.11 Read Resend register, RESEND (SW-DP only) on page 108.
IR	Yes	No	Instruction Register. See *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* for more information.
BYPASS	Yes	No	Bypass register. See *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* for more information.
DPACC	Yes	No	DP Access register. See *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* for more information.
APACC	Yes	No	AP Access register. See *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* for more information.

A.4 DAP register descriptions

This section describes the following DAP component registers and their bit assignments.

A.4.1 AHB-AP register descriptions

This section describes the programmable AHB-AP registers. It contains the following registers:

- A.4.1.1 AHB-AP Control/Status Word register, CSW, 0x00 on page 90.
- A.4.1.2 AHB-AP Transfer Address Register, TAR, 0x04 on page 93.
- A.4.1.3 AHB-AP Data Read/Write register, DRW, 0x0C on page 94.
- A.4.1.4 AHB-AP Banked Data registers, BD0-BD03, 0x10-0x1C on page 94.
- A.4.1.5 AHB-AP Debug Base Address register, ROM, 0xF8 on page 95.
- A.4.1.6 AHB-AP Configuration register, CFG, 0xF4 on page 95.
- A.4.1.7 AHB-AP Identification Register, IDR, 0xFC on page 96.

A.4.1.1 AHB-AP Control/Status Word register, CSW, 0x00

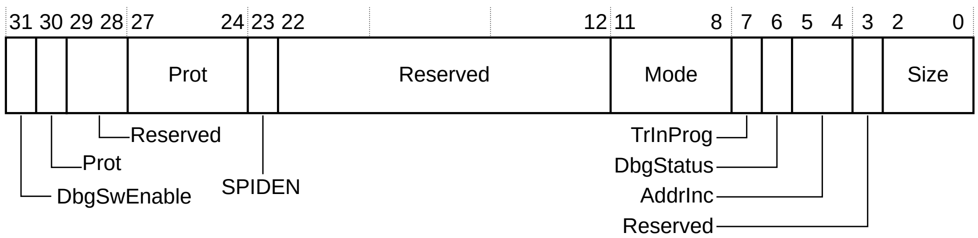
AHB-AP Control/Status Word register configures and controls transfers through the AHB interface.

Attributes

See A.3 DAP register summary on page 88.

The following figure shows the AHB-AP CSW register bit assignments.

Figure A-3: AHB-AP CSW register bit assignments



The following table shows the AHB-AP CSW register bit assignments.

Table A-4: AHB-AP Control/Status Word register bit assignments

Bits	Type	Name	Function
[31]	RO	DbgSwEnable	Not implemented in Cortex®-M33 DAP. Treat as RAZ/SBZP.
[30]	RW	Prot	Specifies the security of the AHB transfer output on SLVNONSEC. **0** Secure transfer. **1** Non-secure transfer. This bit resets to 0.
[29:28]	RW	-	Reserved, SBZ.
[27:24]	RW	Prot	Specifies the signal encodings to be output on SLVPROT[6], SLVPROT[4], and SLVPROT[3:0]. **SLVPROT[6]** CSW.Prot[27] **SLVPROT[4]** CSW.Prot[27] **SLVPROT[3:0]** CSW.Prot[27:24] **Note:** • CSW.Prot[27] is tied to 0. • SLVPROT[5] is tied to 0.
[23]	RO	SPIDEN	Not implemented in Cortex®-M33 DAP. Treat as RAZ/SBZP.
[22:12]	-	-	Reserved. Treat as RAZ/SBZP.
[11:8]	RO	Mode	Not implemented in Cortex®-M33 DAP. Treat as RAZ/SBZP.
[7]	RO	TrInProg	Not implemented in Cortex®-M33 DAP. Treat as RAZ/SBZP.

Bits	Type	Name	Function
[6]	RO	DbgStatus	Indicates the status of the DEVICEEN port. If DbgStatus is LOW, no AHB transfers are carried out. **0** AHB transfers not permitted. **1** AHB transfers permitted.
[5:4]	RW	AddrInc	Auto address increment and packing mode on RW data access. Only increments if the current transaction completes without an error response and the transaction is not aborted. Auto address incrementing and packed transfers are not performed on access to Banked Data registers, `0x10`-`0x1C`. The status of these bits is ignored in these cases. Incrementing and wrapping is performed within a 1KB address boundary, for example, for word incrementing from `0x1400`-`0x17FC`. If the start is at `0x14A0`, then the counter increments to `0x17FC`, wraps to `0x1400`, then continues incrementing to `0x149C`. **`0b00`** Auto increment OFF. **`0b01`** Increment, single. Single transfer from corresponding byte lane. **`0b10`** Reserved, SBZ. No transfer. **`0b11`** Reserved, SBZ. No transfer. The Size field, bits[2:0] defines the size of address increment The reset value is `0b00`. **Note:** Bit[5] is RO and RAZ.
[3]	RW	-	Reserved, SBZ. The reset value is 0.

Bits	Type	Name	Function
[2:0]	RW	Size	Size of the data access to perform: **0b000** 8 bits. **0b001** 16 bits. **0b010** 32 bits. **0b011-0b111** Reserved, SBZ. The reset value is `0b000`. **Note:** Bit[2] is RO and RAZ.

A.4.1.1.1 Prot field bit descriptions

The following table describes Prot field bits.

Table A-5: Prot field bit descriptions

Bit	Description
27	Shareable, Lookup, Modifiable: **0** Non-shareable, no-look up, non-modifiable. **1** Shareable, lookup, modifiable.
26	Bufferable: **0** Non-bufferable. **1** Bufferable.
25	Privileged: **0** Non-privileged. **1** Privileged.
24	Data/Instruction access: **1** Data access. This bit is RO.

A.4.1.2 AHB-AP Transfer Address Register, TAR, 0x04

AHB-AP Transfer Address Register holds the memory address to be accessed.

Attributes

See A.3 DAP register summary on page 88.

The following table shows the AHB-AP Transfer Address Register bit assignments.

Table A-6: AHB-AP Transfer Address Register bit assignments

Bits	Type	Name	Function
[31:0]	RW	Address	Address of the current transfer **Note:** This register is not reset

A.4.1.3 AHB-AP Data Read/Write register, DRW, 0x0C

AHB-AP Data Read/Write register maps an AP access directly to one or more memory accesses. The AP access does not complete until the memory access, or accesses, complete.

Attributes

See A.3 DAP register summary on page 88.

The following table shows the AHB-AP Data Read/Write register bit assignments.

Table A-7: AHB-AP Data Read/Write register bit assignments

Bits	Type	Name	Function
[31:0]	RW	Data	**Write mode** Data value to write for the current transfer. **Read mode** Data value that is read from the current transfer.

A.4.1.4 AHB-AP Banked Data registers, BD0-BD03, 0x10-0x1C

AHB-AP Banked Data registers, BD0-BD03 provide a mechanism for directly mapping through DAP accesses to AHB transfers without having to rewrite the TAR within a four-location boundary. BD0 is RW from TA. BD1 is RW from TA+4.

Attributes

See A.3 DAP register summary on page 88.

The following table shows the Banked Data register bit assignments.

Table A-8: Banked Data register bit assignments

Bits	Type	Name	Function
[31:0]	RW	Data	If dapcaddr[7:4] = `0x0001`, it is accessing AHB-AP registers in the range `0x10-0x1C`, and the derived haddr[31:0] is: **Write mode** Data value to write for the current transfer to external address TAR[31:4] + dapcaddr[3:2] + `0b00`. **Read mode** Data value that is read from the current transfer from external address TAR[31:4] + dapcaddr[3:2] + `0b00`. Auto address incrementing is not performed on DAP accesses to BD0-BD3. Banked transfers are only supported for word transfers. Non-word banked transfers are reserved and **UNPREDICTABLE**. Transfer size is ignored for banked transfers.

A.4.1.5 AHB-AP Debug Base Address register, ROM, 0xF8

AHB-AP Debug Base Address register provides an index into the connected memory-mapped resource. This index value points to a ROM table that describes the connected debug components.

Attributes

See A.3 DAP register summary on page 88.

The following table shows the AHB-AP Debug Base Address register bit assignments.

Table A-9: AHB-AP Debug Base Address register bit assignments

Bits	Type	Name	Function
[31:0]	RO	Debug AHB ROM Address	Base address of a ROM table. Bit[1] is always 1, bits[31:12] are set to the tie-off value on the static input port BASEADDR[31:12]. Bits[11:2] are set to `0x000` and bit[0] is set to BASEADDR[0]. The ROM provides a lookup table that points to debug components.

A.4.1.6 AHB-AP Configuration register, CFG, 0xF4

AHB-AP configuration register describes the features that are configured in the AHB-AP implementation.

Attributes

See A.3 DAP register summary on page 88.

The following table shows the AHB-AP Configuration register bit assignments.

Table A-10: AHB-AP Configuration register bit assignments

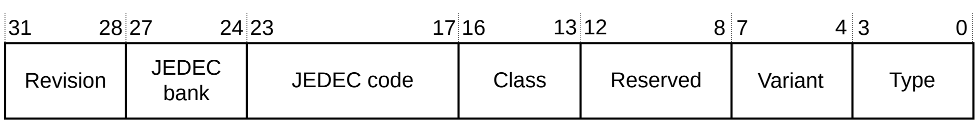
Bits	Type	Name	Value	Function
[31:3]	-	Reserved	`0x00000000`	-
[2]	RO	LD	`0x0`	Large data. Data not larger than 32-bits supported.
[1]	RO	LA	`0x0`	Long address. Physical addresses of 32 bits, or less supported. Greater than 32 bits is not supported.
[0]	RO	BE	`0x0`	Only little-endian supported.

A.4.1.7 AHB-AP Identification Register, IDR, 0xFC

AHB-AP Identification register specifies the AHB-AP identification values.

The following figure shows the AHB-AP Identification Register bit assignments.

Figure A-4: AHB-AP Identification Register bit assignments



The following table shows the AHB-AP Identification Register bit assignments.

Table A-11: AHB-AP Identification Register bit assignments

Bits	Type	Name	Value	Function
[31:28]	RO	Revision	`0x1`	r0p1
[27:24]	RO	JEDEC bank	`0x4`	Designed by Arm®
[23:17]	RO	JEDEC code	`0x3B`	Designed by Arm®
[16:13]	RO	Class	`0x8`	Is a Mem AP
[12:8]	-	Reserved	`0x00`	-
[7:4]	RO	Variant	`0x1`	Cortex®-M33
[3:0]	RO	Type	`0x5`	AHB5

A.4.2 Debug port registers

This section describes the DP registers.

A.4.2.1 AP Abort register, ABORT

AP Abort register forces an AP transaction abort.

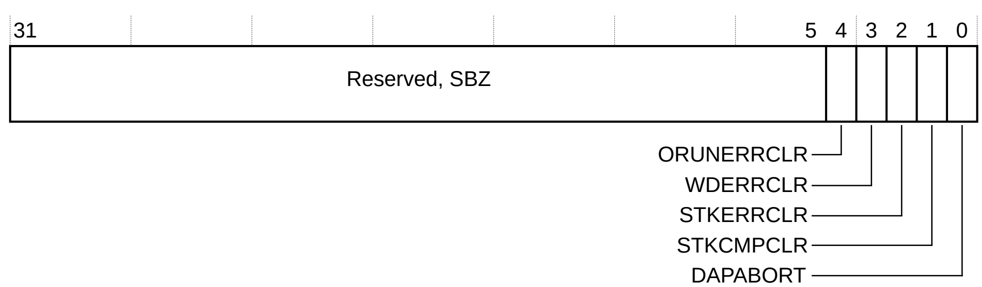
Attributes

The ABORT register is:

- A write-only register.
- Accessible through JTAG-DP and SW-DP.
- Accessed in a **DATA LINK DEFINED** manner:
  - JTAG-DP access is through its own scan-chain.
  - A write to offset `0x0` of the DP register map accesses SW-DP.
- Always accessible, completes all accesses on the first attempt, and returns an `OK` response if a valid transaction is received.

The following figure shows the ABORT bit assignments.

Figure A-5: ABORT bit assignments



The following table shows the ABORT bit assignments.

Table A-12: ABORT bit assignments

Bits	Function	Description
[31:5]	-	Reserved, SBZ.
[4]	ORUNERRCLR	Setting this bit to 1 sets the STICKYORUN overrun error flag[14] to 0.
[3]	WDERRCLR[13]	Setting this bit to 1 sets the WDATAERR write data error flag[14] to 0.
[2]	STKERRCLR	Setting this bit to 1 sets the STICKYERR sticky error flag[14] to 0.
[1]	STKCMPCLR	Reserved, SBZ. The DP is a MINDP implementation, therefore this bit is not implemented.

Bits	Function	Description
[0]	DAPABORT	Setting this bit to 1 generates a DAP abort, that aborts the current AP transaction. **Note:** Perform this only if the debugger has received WAIT responses over an extended period.

A.4.2.2 Identification Code register, IDCODE

Identification Code register provides identification information about the JTAG-DP. The IDCODE register is always accessible.

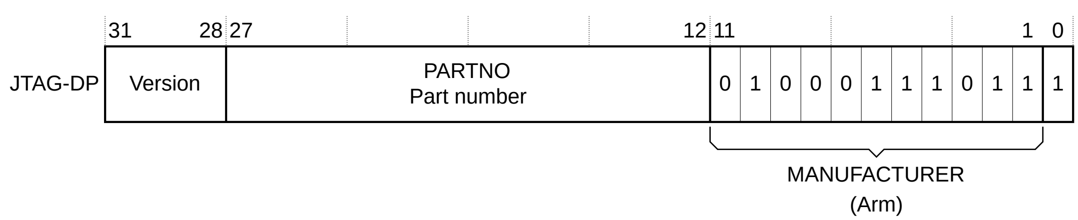
Attributes

The IDCODE register is:

- A read-only register.
- Accessed through its own scan chain when the IR contains `0b1110`.

The following figure shows the Identification Code register bit assignments.

Figure A-6: Identification Code register bit assignments



The following table shows the Identification Code register bit assignments.

Table A-13: Identification Code register bit assignments

Bits	Function	Description
[31:28]	Version	JTAG-DP revision code exclusive OR-gated with ECOREVNUM[7:4] signal: **`0x0`** r0p0.
[27:12]	PARTNO	Part Number for the JTAG-DP, `0xBA04`.
[11:1]	MANUFACTURER	JEDEC Manufacturer ID, an 11-bit JEDEC code that identifies the designer of the device. See A.4.2.2.1 JEDEC Manufacturer ID on page 99. in this figure shows the Arm value for this field as `0x23B`. This value must not be changed.

13 Implemented on SW-DP only. On a JTAG-DP, this bit is Reserved, SBZ.
14 In the Control/Status Register, see A.4.2.4 Control/Status register, CTRL/STAT on page 100.

Bits	Function	Description
[0]	-	Always 1.

A.4.2.2.1 JEDEC Manufacturer ID

This code is also described as the JEP-106 manufacturer identification code, and can be subdivided into two fields, as the following table shows. The JEDEC Solid-State Technology Association assign JEDEC codes.

See the *JEDEC Standard Manufacturer's Identification Code, JEP106*.

Table A-14: JEDEC JEP106 manufacturer ID code, with Arm® values

MANUFACTURER field	Bits[15]	Arm® registered value
Continuation code	4 bits, [11:8]	`0b0100`, `0x4`
Identity code	7 bits, [7:1]	`0b0111011`, `0x3B`

A.4.2.3 Debug Port Identification Register, DPIDR

Debug Port Identification register provides identification information about the JTAG-DP and SW-DP.

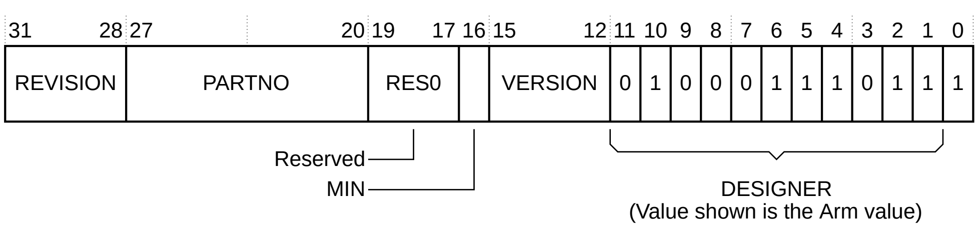
Attributes

The DPIDR register is:

- A read-only register.
- Accessed by a read at offset `0x0` of the DP register map.

The following figure shows the Debug Port Identification Register bit assignments.

Figure A-7: Debug Port Identification Register bit assignments



The following table shows Debug Port Identification Register the bit assignments.

15 Field width, in bits, and the corresponding bits in the Identification Code Register.

Table A-15: Debug Port Identification Register bit assignments

Bits	Function	Description
[31:28]	REVISION	DP revision code exclusive OR-gated with the ECOREVNUM[7:4] signal: **JTAG-DP** `0x0`, r0p0. **SW-DP** `0x0`, r0p0.
[27:20]	PARTNO	Part Number for this debug port, `0xBE`.
[19:17]	-	Reserved, RAZ.
[16]	MIN	Reads as 1, indicating that the *Minimal Debug Port* (MINDP) architecture is implemented. Transaction counter, Pushed-verify, and Pushed-find operations are not implemented.
[15:12]	VERSION	JTAG-DP is DP architecture version is `0x1`. SW-DP is DP architecture version is `0x2`.
[11:1]	MANUFACTURER	JEDEC Manufacturer ID, an 11-bit JEDEC code that identifies the designer of the device. See A.4.2.2.1 JEDEC Manufacturer ID on page 99. A.4.2.2 Identification Code register, IDCODE on page 98 shows the Arm value for this field as `0x23B`. This value must not be changed.
[0]	-	Always 1.

A.4.2.4 Control/Status register, CTRL/STAT

Control/Status register provides control of the DP and its status information.

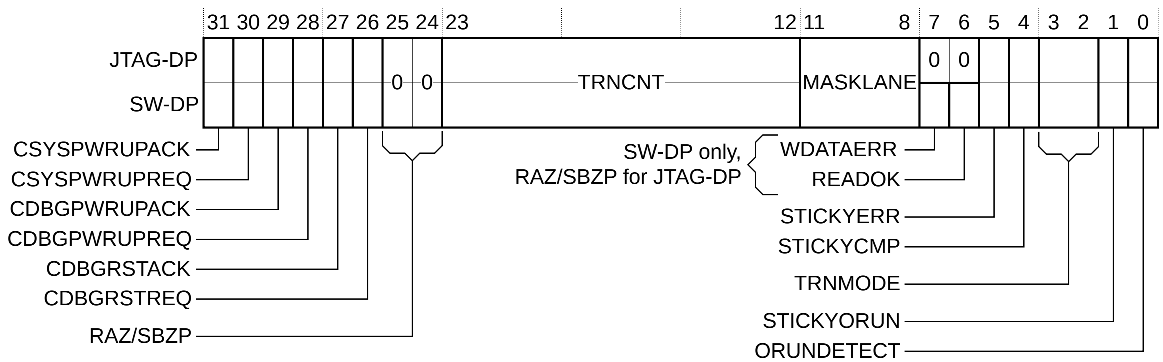
Attributes

The CTRL/STAT register is:

- A read/write register. Some fields are RO, meaning they ignore writes, see the field descriptions for more information.
- JTAG-DP. At address `0x4` when the IR contains DPACC, when SELECT.DPBANKSEL is `0x0`.
- SW-DP. At address `0x4` when APnDP bit is 0, and SELECT.DPBANKSEL is `0x0`.

The following figure shows the Control/Status register bit assignments.

Figure A-8: Control/Status register bit assignments



The following table shows the Control/Status register bit assignments.

Table A-16: Control/Status register bit assignments

Bits	Access	Function	Description
[31]	RO	CSYSPWRUPACK	System powerup acknowledge.
[30]	RW	CSYSPWRUPREQ	System powerup request. The reset value is 0.
[29]	RO	CDBGPWRUPACK	Debug powerup acknowledge.
[28]	RW	CDBGPWRUPREQ	Debug powerup request. The reset value is 0.
[27]	RO	CDBGRSTACK	Debug reset acknowledge.
[26]	RW	CDBGRSTREQ	Debug reset request. The reset value is 0.
[25:24]	-	-	Reserved, RAZ/SBZP.
[23:12]	RAZ/ SBZP	TRNCNT	The Cortex®-M33 is a MINDP implementation, therefore this field is reserved.
[11:8]	RAZ/ SBZP	MASKLANE	The Cortex®-M33 is a MINDP implementation, therefore this field is reserved.
[7]	RO[16]	WDATAERR[17]	If a Write Data Error occurs, this bit is set to 1. It is set if: • There is a parity or framing error on the data phase of a write. • A write that the debug port accepted is then discarded without being submitted to the access port. This bit can only be set to 0 by writing 1 to ABORT.WDERRCLR. The reset value after a Powerup reset is 0.

Bits	Access	Function	Description
[6]	RO[16]	READOK[17]	If the response to the previous access port read or RDBUFF read was OK, this bit is set to 1. If the response was not OK, it is set to 0. This flag always indicates the response to the last access port read access. The reset value after a Powerup reset is 0.
[5]	RO[16]	STICKYERR	If an error is returned by an access port transaction, this bit is set to 1. To set this bit to 0: **JTAG-DP** Either: • Write 1 to this bit of this register. • Write 1 to ABORT.STKERRCLR. **SW-DP** Write 1 to ABORT.STKERRCLR. After a Powerup reset, this bit is LOW.
[4]	RAZ	STICKYCMP	The Cortex®-M33 is a MINDP implementation, therefore this field is reserved.
[3:2]	RAZ/ SBZP	TRNMODE	The Cortex®-M33 is a MINDP implementation, therefore this field is reserved.
[1]	RO[16]	STICKYORUN	If overrun detection is enabled (see bit[0] of this register), this bit is set to 1 when an overrun occurs. To set this bit to 0: **JTAG-DP** Either: • Write 1 to this bit of this register. • Write 1 to ABORT.ORUNERRCLR. **SW-DP** Write 1 to ABORT.ORUNERRCLR. After a Powerup reset, the reset value is 0.
[0]	RW	ORUNDETECT	This bit is set to 1 to enable overrun detection. The reset value is 0.

A.4.2.5 AP Select register, SELECT

The AP Select register selects, an *Access Port* (AP) and the active register banks within that AP, and the DP address bank.

Attributes

The SELECT register is:

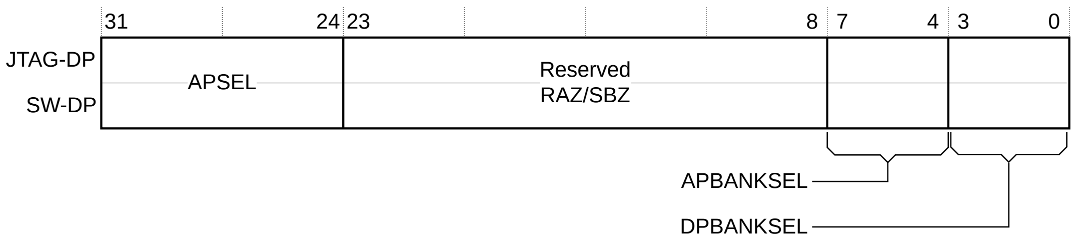
- A write-only register.
- JTAG-DP. At address `0x8` when the IR contains DPACC, and is a WO register.

[16] RO on SW-DP. On a JTAG-DP, this bit can be read normally. Writing a 1 to this bit sets the bit to 0.
[17] Implemented on SW-DP only. On a JTAG-DP, this bit is Reserved,RAZ.

- SW-DP. At address `0x8` on write operations, when the APnDP bit is 0.

The following figure shows the AP Select register bit assignments.

Figure A-9: AP Select register bit assignments



The following table shows the AP Select register bit assignments.

Table A-17: AP Select register bit assignments

Bits	Function	Description
[31:24]	APSEL	Selects the current access port: **`0x00`** Selects the AHB-AP. **`0x01-0x1F`** AP `0x01-0x1F` do not exist, and if selected, AP read transactions return zero and AP writes are ignored. The reset value is **UNPREDICTABLE**.
[23:8]	Reserved. SBZ/ RAZ	Reserved. SBZ/RAZ.
[7:4]	APBANKSEL	Selects the active 4-word register window on the current access port. The reset value is **UNPREDICTABLE**.
[3:0]	DPBANKSEL	Selects the register that appears at DP register `0x4`. JTAG-DP register allocation: **`0x0`** CTRL/STAT. SW-DP register allocation in DPv1: **`0x0`** CTRL/STAT. **`0x1`** DLCR. SW-DP register allocation in DPv2: **`0x0`** CTRL/STAT. **`0x1`** DLCR. **`0x2`** TARGETID. **`0x3`** DLPIDR. **`0x4`** EVENTSTAT.

A.4.2.6 Read Buffer register, RDBUFF

Read Buffer register captures data from the AP that is presented as the result of a previous read.

Attributes

The RDBUFF register is:

- A 32-bit read-only buffer.
- JTAG-DP. Accessed at address `0xC` when the IR contains DPACC.
- SW-DP. Accessed at address `0xC` on read operations when the APnDP bit is 0.
- Has **DATA LINK DEFINED** behavior:
  - JTAG-DP, see A.4.2.6.1 Read Buffer implementation and use on a JTAG-DP on page 104.
  - SW-DP, see A.4.2.6.2 Read Buffer implementation and use on an SW-DP on page 104.

A.4.2.6.1 Read Buffer implementation and use on a JTAG-DP

On a JTAG-DP, the read buffer is RAZ/WI.

The read buffer is architecturally defined to provide a debug port read operation that does not have any side effects. This means that a debugger can insert a debug port read of the read buffer at the end of a sequence of operations to return the final AP read result and ACK values.

A.4.2.6.2 Read Buffer implementation and use on an SW-DP

On an SW-DP, performing a read of the read buffer captures data from the access port, presented as the result of a previous read, without initiating a new access port transaction. This means that reading the read buffer returns the result of the last access port read access, without generating a new AP access.

After you read the read buffer, its contents are no longer valid. The result of a second read of the read buffer is **UNPREDICTABLE**.

If you require the value from an access port register read, that read must be followed by one of:

- A second access port register read. You can read the CSW if you want to ensure that this second read has no side effects.
- A read of the DP Read Buffer.

This second access, to the access port or the debug port depending on which option you use, stalls until the result of the original access port read is available.

A.4.2.7 Event Status register, EVENTSTAT

Event Status register signals to the debugger that the Cortex®-M33 processor is halted.

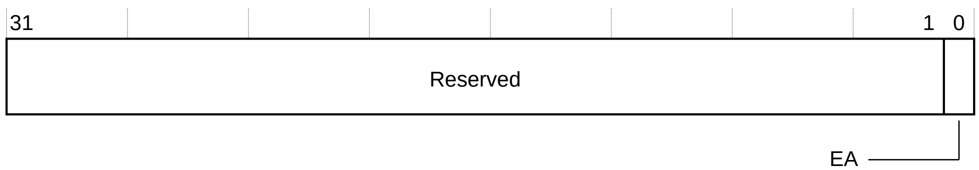
Attributes

The EVENTSTAT register is:

- A read-only register.
- Accessed by a read at offset `0x4` of the DP register map when SELECT.DPBANKSEL is set to `0x4`.

The following figure shows the Event Status register bit assignments.

Figure A-10: Event Status register bit assignments



The following table shows the Event Status register bit assignments.

Table A-18: Event Status register bit assignments

Bits	Function	Description
[31:1]	-	Reserved, RAZ.
[0]	EA	Event status flag. Indicates that the Cortex®-M33 processor is halted: **0** Processor is halted. **1** Processor is not halted.

A.4.2.8 Data Link Control Register, DLCR (SW-DP only)

Data Link Control register controls the operating mode of the Data Link.

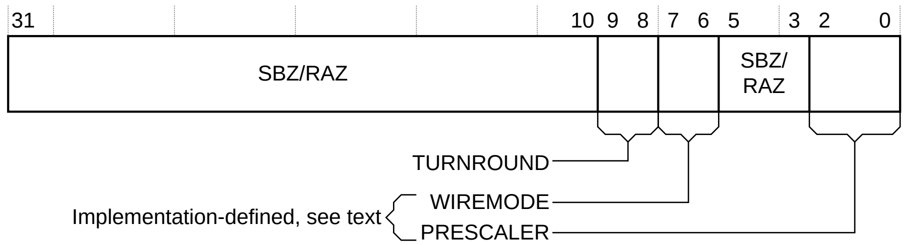
Attributes

The DLCR register is:

- A read/write register.
- Accessed by a read or write at offset `0x4` of the DP address map when SELECT.DPBANKSEL is set to `0x1`.

The following figure shows the Data Link Control Register bit assignments.

Figure A-11: Data Link Control Register bit assignments



The following table shows the Data Link Control Register bit assignments.

Table A-19: Data Link Control Register bit assignments

Bits	Function	Description
[31:10]	-	Reserved, SBZ/RAZ.
[9:8]	TURNROUND	Turnaround tristate period. This field only supports `0b00`, other write values are treated as a protocol error. The reset value is `0b00`.
[7:6]	WIREMODE	This field identifies SW-DP as operating in Synchronous mode only. It is fixed to `0b00`. The reset value is `0b00`.
[5:3]	-	Reserved, SBZ/RAZ.
[2:0]	PRESCALER	Reserved, SBZ/RAZ.

A.4.2.9 Target Identification register, TARGETID (SW-DP only)

Target Identification register provides information about the target when the host is connected to a single device.

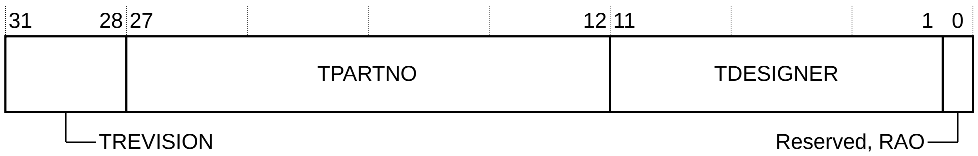
Attributes

The TARGETID register is:

- A read-only register.
- Accessed by a read at offset `0x4` of the DP register map when SELECT.DPBANKSEL is set to `0x2`.

The following figure shows the Target Identification register bit assignments.

Figure A-12: Target Identification register bit assignments



The following table shows the Target Identification register bit assignments.

Table A-20: Target Identification register bit assignments

Bits	Function	Description
[31:28]	TREVISION	Target revision.
[27:12]	TPARTNO	Configuration dependent. The designer of the part assigns this value and must be unique to that part.
[11:1]	TDESIGNER	Arm® designer code (`0x23B`).
[0]	-	Reserved, RAO.

A.4.2.10 Data Link Protocol Identification Register, DLPIDR (SW-DP only)

Data Link Protocol Identification register provides protocol version information.

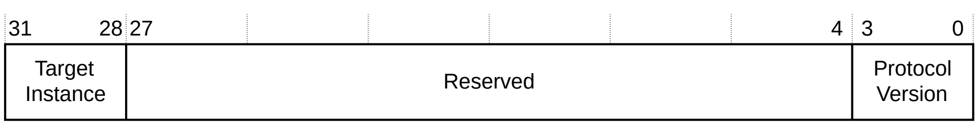
Attributes

The DLPIDR is:

- A read-only register.
- Accessed by a read at offset `0x4` of the DP register map when SELECT.DPBANKSEL is set to `0x3`.

The following figure shows the Data Link Protocol Identification Register bit assignments.

Figure A-13: Data Link Protocol Identification Register bit assignments



The following table shows the Data Link Protocol Identification Register bit assignments.

Table A-21: Data Link Protocol Identification Register bit assignments

Bits	Function	Description
[31:28]	Target Instance	Configuration dependent. This field defines a unique instance number for this device within the system. This value must be unique for all devices that are connected together in a multidrop system with identical values in the TREVISION fields in the TARGETID register. The value of this field reflects the value of the instanceid[3:0] input.
[27:4]	-	Reserved.
[3:0]	Protocol Version	Defines the serial wire protocol version. This value is `0x1`, that indicates SW protocol version 2.

A.4.2.11 Read Resend register, RESEND (SW-DP only)

Read Resend register enables the read data to be recovered from a corrupted debugger transfer without repeating the original AP transfer.

Attributes

The RESEND register is:

- A read-only register.
- Accessed by a read at offset `0x8` in the DP register map.

Performing a read to the RESEND register does not capture new data from the AP, it returns the value that was returned by the last AP read or DP RDBUFF read.

Reading the RESEND register enables the read data to be recovered from a corrupted SW-DP transfer without having to re-issue the original read request, or generate a new access to the connected debug memory system.

The RESEND register can be accessed multiple times, it always returns the same value until a new access is made to an AP register or the DP RDBUFF register.

A.4.2.12 DP register descriptions

More information about the DP registers, their features, and how to access them can be found in the

Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2 .

Appendix B Trace Port Interface Unit

This appendix describes the Cortex®-M33 TPIU that can be used with the Cortex®-M33 processor.

B.1 About the TPIU

The Cortex®-M33 TPIU is an optional component that bridges between the on-chip trace data from the ETM and the ITM, with separate IDs, to a data stream.

The Cortex®-M33 TPIU encapsulates IDs where required, and an external *Trace Port Analyzer* (TPA) captures the data stream.

The Cortex®-M33 TPIU is specially designed for low-cost debug. If your implementation requires the additional features, like those in the CoreSight SoC-400 TPIU, your implementation can replace the Cortex®-M33 TPIU with other CoreSight components.

In this chapter, the term TPIU refers to the Cortex®-M33 TPIU. For information about the CoreSight SoC-400 TPIU, see the *Arm® CoreSight™ SoC-400 Technical Reference Manual*.

B.2 TPIU functional description

The TPIU supports up to two ATB ports.

The ATB1 and ATB2 parameters provide the following configuration options:

ATB2 = 0 and ATB1 = 0
Illegal combination

ATB2 = 0 and ATB1 = 1
ATB port 1 present

ATB2 = 1 and ATB1 = 0
ATB port 2 present

ATB2 = 1 and ATB1 = 1
Both ATB port 1 and 2 present

In a system, Arm® recommends that the ITM is connected to ATB port 1 and an ETM is connected to ATB port 2.

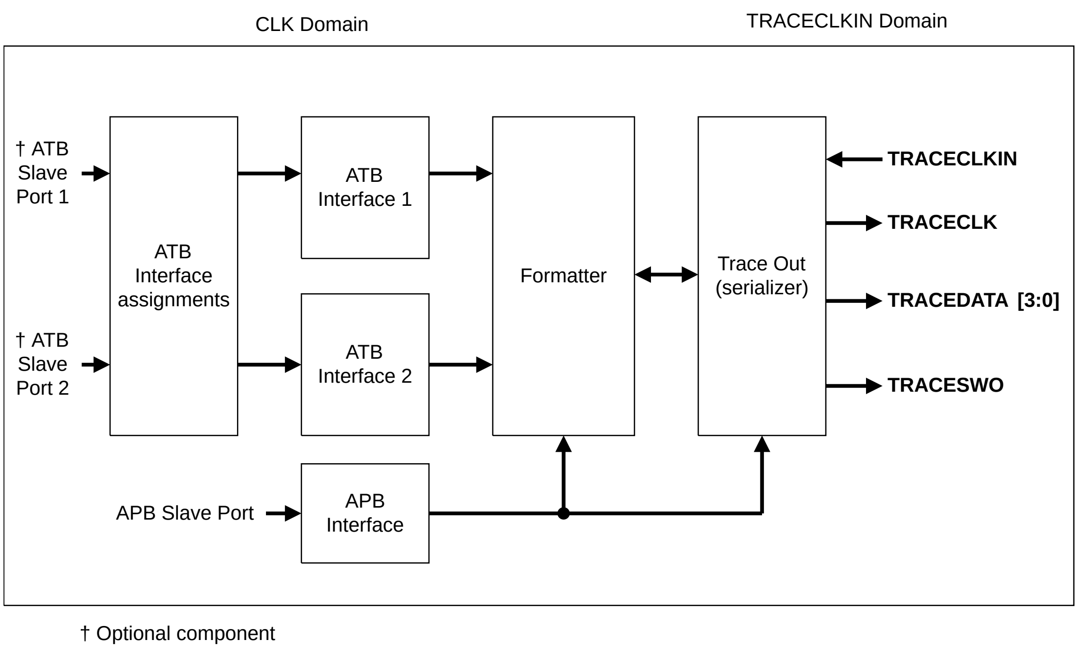
If your implementation requires no trace support, then the TPIU might not be present.

Note

If your system design uses the optional ETM component, the TPIU configuration supports both ITM and ETM debug trace. See the *Arm® CoreSight™ ETM-M33 Technical Reference Manual*.

The following figure shows the component layout of the TPIU for both configurations.

Figure B-1: TPIU block diagram



If only one ATB slave port is present, it is assigned to ATB interface 1 and ATB interface 2 is removed. If ATB slave ports 1 and 2 are present, they are assigned to ATB interface 1 and 2 respectively.

B.2.1 TPIU Formatter

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

The formatting protocol is described in the *Arm® CoreSight™ Architecture Specification v2.0*. You must enable synchronization in the DWT or TPIU_PSCR to provide synchronization for the formatter.

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

B.2.2 Serial Wire Output format

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

- TPIU_DEVID specifies the formats that are supported. See B.3.10 Device Configuration Register on page 120.
- TPIU_SPPR specifies the SWO format in use. See the *Arm®v8-M Architecture Reference Manual*.

When one of the two SWO modes is selected, you can enable the TPIU to bypass the formatter for trace output. If the formatter is bypassed, only one trace source passes through. When the formatter is bypassed, only data on the ATB interface 1 is passed through and ATB interface 2 data is discarded.

Note

When operating in bypass mode, Arm® recommends that in a configuration that supports and ETM and ITM, the ITM data is passed through by connecting the ITM to the ATB Slave Port 1.

B.3 TPIU programmers model

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

Note

Arm® recommends that the TPIU is only reprogrammed before any data has been presented on either ATB slave port and either:

- After both ATRESETn and TRESETn have been applied.
- After a flush has been completed using FFCR.FOnMan.

If this is not followed, reprogramming can lead to either momentary or permanent data corruption that might require ATRESETn and TRESETn to be applied.

Table B-1: TPIU registers

Address	Name	Type	Reset	Description
`0xE0040000`	TPIU_SSPSR	RO	-[18]	Supported Parallel Port Size Register
`0xE0040004`	TPIU_CSPSR	RW	`0x01`	Current Parallel Port Size Register
`0xE0040010`	TPIU_ACPR	RW	`0x0000`	B.3.1 Asynchronous Clock Prescaler Register on page 112
`0xE00400F0`	TPIU_SPPR	RW	`0x01`	Selected Pin Protocol Register
`0xE0040300`	TPIU_FFSR	RO	`0x08`	B.3.2 Formatter and Flush Status Register on page 113
`0xE0040304`	TPIU_FFCR	RW	`0x102`	B.3.3 Formatter and Flush Control Register on page 114

[18] The value at reset is tied to the MAXPORTSIZE configuration tie off.

Address	Name	Type	Reset	Description
0xE0040308	TPIU_PSCR	RW	0x00	TPIU Periodic Synchronization Control Register[19]
0xE0040EE8	TRIGGER	RO	0x0	B.3.4 TRIGGER Register on page 115
0xE0040EEC	ITFTTD0	RO	0x--000000	B.3.5 Integration Test FIFO Test Data 0 Register on page 116
0xE0040EF0	ITATBCTR2	RW	0x0	B.3.6 Integration Test ATB Control Register 2 on page 117
0xE0040EF8	ITATBCTR0	RO	0x0	B.3.8 Integration Test ATB Control 0 Register on page 119
0xE0040EFC	ITFTTD1	RO	0x--000000	B.3.7 Integration Test FIFO Test Data 1 Register on page 118
0xE0040F00	ITCTRL	RW	0x0	B.3.9 Integration Mode Control on page 119
0xE0040FA0	CLAIMSET	RW	0xF	Claim tag set
0xE0040FA4	CLAIMCLR	RW	0x0	Claim tag clear
0xE0040FC8	DEVID	RO	0xCA0/0xCA1	B.3.10 Device Configuration Register on page 120
0xE0040FCC	DEVTYPE	RO	0x11	B.3.11 Device Type Identifier Register on page 121
0xE0040FD0	PIDR4	RO	0x04	Peripheral identification registers
0xE0040FD4	PIDR5	RO	0x00	
0xE0040FD8	PIDR6	RO	0x00	
0xE0040FDC	PIDR7	RO	0x00	
0xE0040FE0	PIDR0	RO	0x21	
0xE0040FE4	PIDR1	RO	0xBD	
0xE0040FE8	PIDR2	RO	0x0B	
0xE0040FEC	PIDR3	RO	-[20]	
0xE0040FF0	CIDR0	RO	0x0D	Component identification registers
0xE0040FF4	CIDR1	RO	0x90	
0xE0040FF8	CIDR2	RO	0x05	
0xE0040FFC	CIDR3	RO	0xB1	

The following sections describe the TPIU registers whose implementation is specific to this processor. The Formatter, Integration Mode Control, and Claim Tag registers are described in the *CoreSight™ Components Technical Reference Manual*. Other registers are described in the *Arm®v8-M Architecture Reference Manual*.

B.3.1 Asynchronous Clock Prescaler Register

The Asynchronous Clock Prescaler Register, TPIU_ACPR, scales the baud rate of the asynchronous output.

Usage constraints

There are no usage constraints.

[19] The Synchronization Counter counts up to a maximum of 2^16 bytes, where the TPIU_PSCR.PSCount value determines the reload value of Synchronization Counter, as 2 to the power of the programmed value.

The TPIU_PSCR.PSCount value has a range between `0b100` and `0b10000`, any attempt to program register outside the range causes the Synchronization Counter to become disabled.

[20] The value at reset is ECOREVNUM value.

Configurations

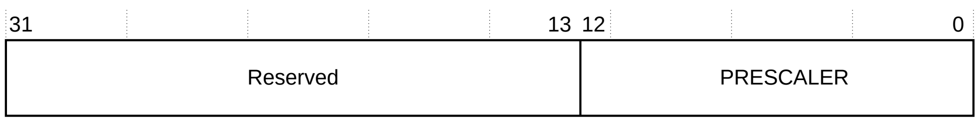
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the TPIU_ACPR bit assignments.

Figure B-2: TPIU_ACPR bit assignments



The following table shows the TPIU_ACPR bit assignments.

Table B-2: TPIU_ACPR bit assignments

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

B.3.2 Formatter and Flush Status Register

The Formatter and Flush Status Register, TPIU_FFSR, indicates the status of the TPIU formatter.

Usage constraints

There are no usage constraints.

Configurations

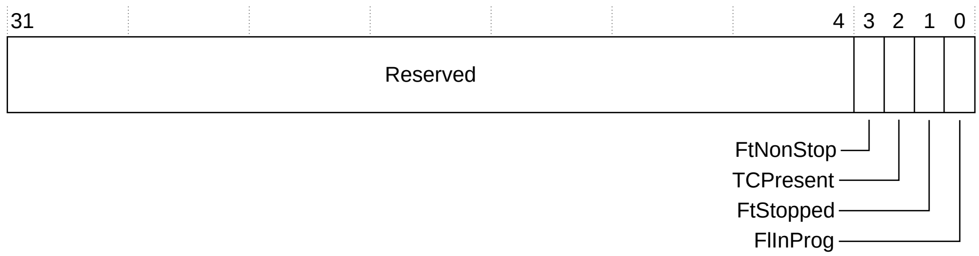
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the TPIU_FFSR bit assignments.

Figure B-3: TPIU_FFSR bit assignments



The following table shows the TPIU_FFSR bit assignments.

Table B-3: TPIU_FFSR bit assignments

Bits	Name	Function
[31:4]	-	Reserved
[3]	FtNonStop	Formatter cannot be stopped
[2]	TCPresent	This bit always reads zero
[1]	FtStopped	This bit always reads zero
[0]	FlInProg	Read only. Flush in progress. Value can be: **0** When all the data received, before the flush is acknowledged, has been output on the trace port **1** When a flush is initiated.

B.3.3 Formatter and Flush Control Register

The Formatter and Flush Control Register, TPIU_FFCR, controls the TPIU formatter.

Usage constraints

There are no usage constraints.

Configurations

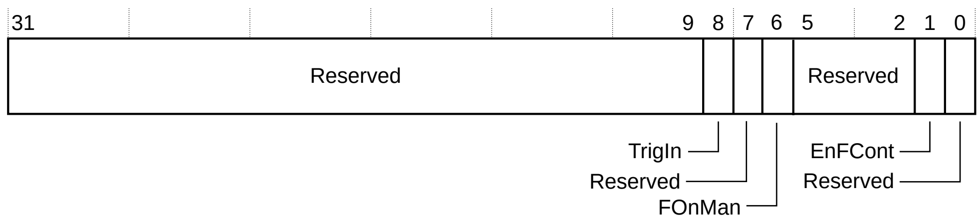
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the TPIU_FFCR bit assignments.

Figure B-4: TPIU_FFCR bit assignments



The following table shows the TPIU_FFCR bit assignments.

Table B-4: TPIU_FFCR bit assignments

Bits	Name	Function
[31:9]	-	Reserved.
[8]	TrigIn	This bit Reads-As-One (RAO), specifying that triggers are inserted when a trigger pin is asserted.
[7]	-	Reserved.
[6]	FOnMan	Flush on manual. Value can be: **0** When the flush completes. Set to 0 on a reset of the TPIU. **1** Generates a flush.
[5:2]	-	Reserved.
[1]	EnFCont	Enable continuous formatting. Value can be: **0** Continuous formatting disabled. **1** Continuous formatting enabled.
[0]	-	Reserved.

The TPIU can output trace data in a *Serial Wire Output* (SWO) format. See B.2.2 Serial Wire Output format on page 110.

Note

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

B.3.4 TRIGGER Register

The TRIGGER Register controls the integration test TRIGGER input.

Usage constraints

There are no usage constraints.

Configurations

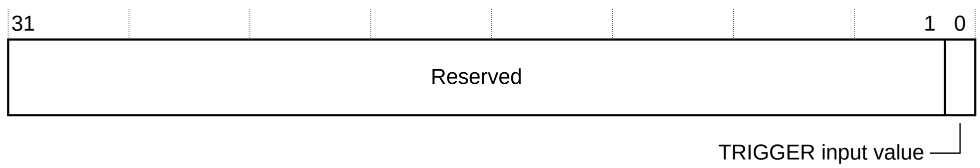
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the TRIGGER bit assignments.

Figure B-5: TRIGGER bit assignments



The following table shows the TRIGGER bit assignments.

Table B-5: TRIGGER bit assignments

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

B.3.5 Integration Test FIFO Test Data 0 Register

The Integration Test FIFO Test Data 0 Register, ITFTTD0, controls trace data integration testing.

Usage constraints

You must set bit[1] of TPIU_ITCTRL to use this register. See B.3.9 Integration Mode Control on page 119.

Configurations

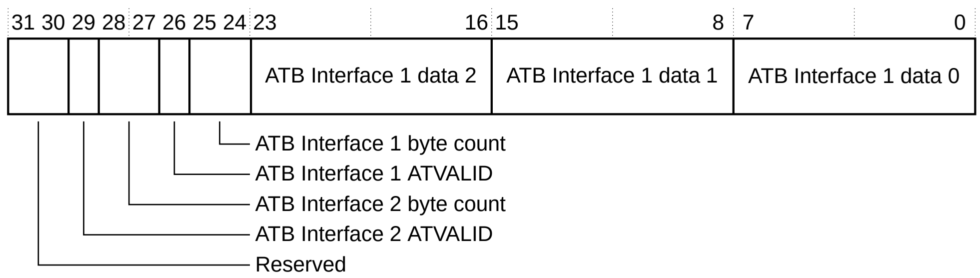
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the Integration Test FIFO Test Data 0 Register data bit assignments.

Figure B-6: ITFTTD0 bit assignments



The following table shows the ITFTTD0 bit assignments.

Table B-6: ITFTTD0 bit assignments

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

B.3.6 Integration Test ATB Control Register 2

The Integration Test ATB Control 2 Register, ITATBCTR2, controls integration test.

Usage constraints

You must set bit[0] of TPIU_ITCTRL to use this register. See B.3.9 Integration Mode Control on page 119.

Configurations

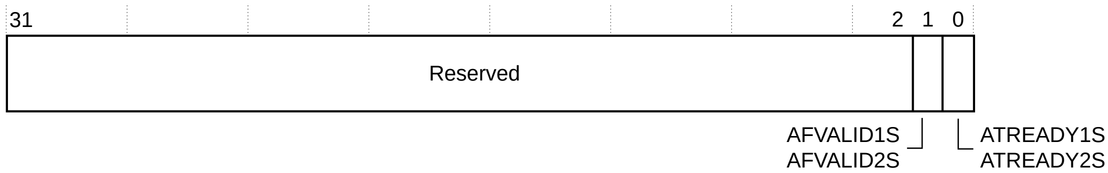
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the ITATBCTR2 bit assignments.

Figure B-7: ITATBCTR2 bit assignments



The following table shows the ITATBCTR2 bit assignments.

Table B-7: ITATBCTR2 bit assignments

Bits	Name	Function
[1]	AFVALID1S, AFVALID2S	This bit sets the value of both the ATB Interface 1 and 2 AFVALID outputs, if the TPIU is in integration test mode.
[0]	ATREADY1S, ATREADY2S	This bit sets the value of both the ATB Interface 1 and 2 ATREADY outputs, if the TPIU is in integration test mode.

B.3.7 Integration Test FIFO Test Data 1 Register

The Integration Test FIFO Test Data 1 Register, ITFTTD1, controls trace data integration testing.

Usage constraints

You must set bit[1] of TPIU_ITCTRL to use this register. See B.3.9 Integration Mode Control on page 119.

Configurations

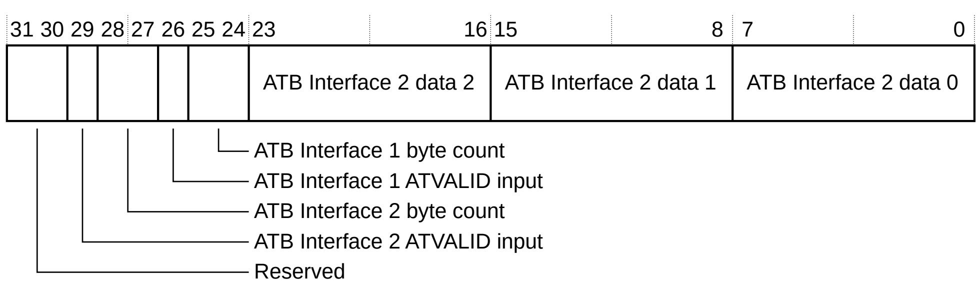
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the ITFTTD1 bit assignments.

Figure B-8: ITFTTD1 bit assignments



The following table shows the ITFTTD1 bit assignments.

Table B-8: ITFTTD1 bit assignments

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

B.3.8 Integration Test ATB Control 0 Register

The Integration Test ATB Control 0 Register, ITATBCTR0, is used for integration test.

Usage constraints

There are no usage constraints.

Configurations

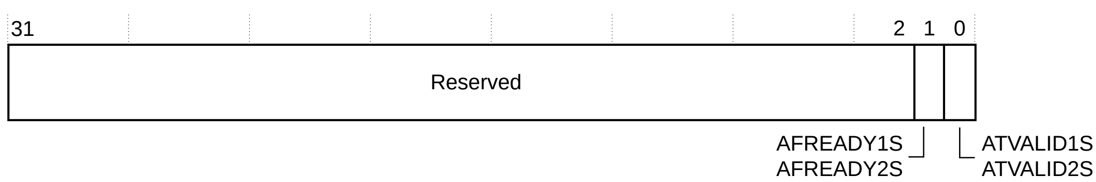
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the ITATBCTR0 bit assignments.

Figure B-9: ITATBCTR0 bit assignments



The following table shows the ITATBCTR0 bit assignments.

Table B-9: ITATBCTR0 bit assignments

Bits	Name	Function
[1]	AFREADY1S, AFREADY2S	A read of this bit returns the value of AFREADY1S OR-gated with AFREADY2S.
[0]	ATVALID1S, ATVALID2S	A read of this bit returns the value of ATVALID1S OR-gated with ATVALID2S

B.3.9 Integration Mode Control

The Integration Mode Control register, TPIU_ITCTRL, specifies normal or integration mode for the TPIU.

Usage constraints

There are no usage constraints.

Configurations

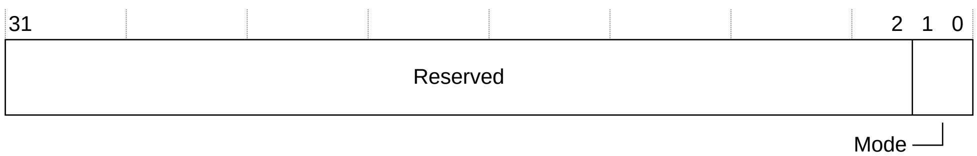
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the TPIU_ITCTRL bit assignments.

Figure B-10: TPIU_ITCTRL bit assignments



The following table shows the TPIU_ITCTRL bit assignments.

Table B-10: TPIU_ITCTRL bit assignments

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

B.3.10 Device Configuration Register

The Device Configuration register, TPIU_DEVID, indicates the functions that are provided by the TPIU for use in the topology detection.

Usage constraints

There are no usage constraints.

Configurations

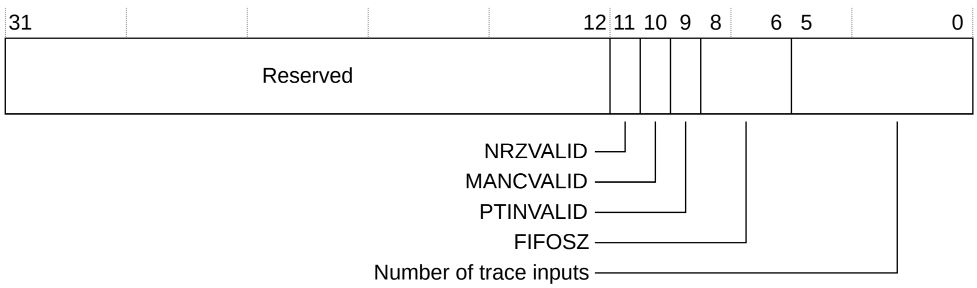
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the TPIU_DEVID bit assignments.

Figure B-11: TPIU_DEVID bit assignments



The following table shows the TPIU_DEVID bit assignments.

Table B-11: TPIU_DEVID bit assignments

Bits	Name	Function
[31:12]	-	Reserved.
[11]	NRZVALID	Indicates support for SWO using UART/NRZ encoding. Always RAO. The output is supported.
[10]	MANCVALID	Indicates support for SWO using Manchester encoding. Always RAO. The output is supported.
[9]	PTINVALID	Indicates support for parallel trace port operation. Always RAZ. Trace data and clock modes are supported.
[8:6]	FIFOSZ	Indicates the minimum implemented size of the TPIU output FIFO for trace data: **0b010** Four bytes.
[5:0]	Number of trace inputs	Specifies the number of trace inputs: **0b000000** One input. **0b000001** Two inputs.

B.3.11 Device Type Identifier Register

The Device Type Identification register, TPIU_DEVTYPE, provides a debugger with information about the component when the Part Number field is not recognized. The debugger can then report this information.

Usage Constraints

There are no usage constraints.

Configurations

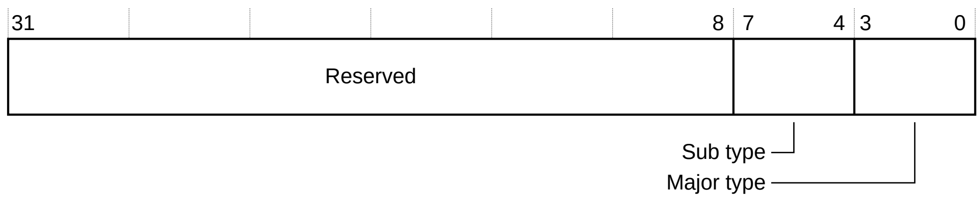
Available in all configurations.

Attributes

See Table B-1: TPIU registers on page 111.

The following figure shows the TPIU_DEVTYPE bit assignments.

Figure B-12: TPIU_DEVTYPE bit assignments



The following table shows the TPIU_DEVTYPE bit assignments.

Table B-12: TPIU_DEVTYPE bit assignments

Bits	Name	Function
[31:8]	-	Reserved.
[7:4]	Sub type	`0x1` Identifies the classification of the debug component.
[3:0]	Major type	`0x1` Indicates this device is a trace sink and specifically a TPIU.

Appendix C UNPREDICTABLE Behaviors

This appendix summarizes the behavior of the Cortex®-M33 processor in cases where the Arm®v8-M architecture is **UNPREDICTABLE**.

C.1 Use of instructions defined in architecture variants

An instruction that is provided by one or more of the architecture extensions is either **UNPREDICTABLE** or **UNDEFINED** in an implementation that does not include those extensions.

In the Cortex®-M33 processor, all instructions not explicitly supported generate an UNDEFINSTR UsageFault exception. For example, using instructions from the Arm®v8-M *Digital Signal Processing* (DSP) extension when this is not included in the processor configuration.

C.2 Use of Program Counter - R15 encoding

R15 is **UNPREDICTABLE** as a source or destination in most data processing operations. R15 is also **UNPREDICTABLE** as a transfer register in certain load/store instructions. Examples of such instructions include LDRT, LDRH, and LDRB.

In the Cortex®-M33 processor, the use of R15 as a named register specifier for any source or destination register that is indicated as **UNPREDICTABLE** generates an UNDEFINSTR UsageFault exception.

C.3 Use of Stack Pointer - as a general purpose register R13

R13 is defined in the Thumb instruction set so that its use is primarily as a stack pointer. R13 is normally identified as *Stack Pointer* (SP) in Thumb instructions.

In 32-bit Thumb instructions, if you use SP as a general purpose register beyond the architecturally defined constraints, the results are **UNPREDICTABLE**.

In the Cortex®-M33 processor, the use of R13 as a named register specifier for any source or destination register that is indicated as **UNPREDICTABLE** generates an UNDEFINSTR UsageFault exception.

In the architecture where the use of R13 as a general purpose register is defined, bits[1:0] of the register must be treated as SBZP. Writing a non-zero value to bits [1:0] results in **UNPREDICTABLE** behavior. In the Cortex®-M33 processor bits [1:0] of R13 are always RAZ/WI.

C.4 Register list in load and store multiple instructions

Load and Store Multiple instructions (`LDM`, `STM`, `PUSH`, `POP` `VLDM`, and `VSTM`) transfer multiple registers to and from consecutive memory locations using an address from a base register, which can be optionally written back when the operation is complete.

The registers are selected from a list encoded in the instruction. Some of these encodings are **UNPREDICTABLE**.

In the Cortex®-M33 processor:

- If the number of registers loaded is zero, then the instruction is a *No Operation* (NOP).
- If the number of registers loaded is one, the single register is loaded.
- If R13 is specified in the list, an UNDEFINSTR UsageFault exception is generated.
- For a Load Multiple, if PC is specified in the list and the instructions is in an IT block and is not the final instruction, an unconditional UNDEFINSTR UsageFault exception is generated.
- For a Store Multiple instruction, if PC is specified in the list an UNDEFINSTR UsageFault exception is generated.
- For a Load Multiple instruction, if base writeback is specified and the register to be written back is also in the list to be loaded, the instruction performs all the loads in the specified addressing mode and the register being written back takes the loaded value.
- For a Store Multiple instruction, if base writeback is specified and the register to be written back is also in the list to be stored, the value stored is the initial base register value. The base register is written back with the expected updated value.
- For a floating-point Load or Store Multiple instruction, `VLDM`, `VSTM` `VPUSH`, and `VPOP` if the register list extends beyond S31 or D15, then the Cortex®-M33 processor generates an UNDEFINSTR UsageFault exception.

C.5 Exception-continuable instructions

To improve interrupt response and increase processing throughput, the processor can take an interrupt during the execution of a Load Multiple or Store Multiple instruction, and continue execution of the instruction after returning from the interrupt. During the interrupt processing, the EPSR.ICI bits hold the continuation state of the Load Multiple or Store Multiple instruction.

In the Cortex®-M33 processor, any values of ICI bits that were not legally written, because of an interruption to an exception-continuable instruction, generate an INVSTATE UsageFault exception on attempt to re-execute the interrupted instruction. This includes the architecturally **UNPREDICTABLE** cases of:

- Not a register in the register list of the Load Multiple or Store Multiple instruction.
- The first register in the register list of the Load Multiple or Store Multiple instruction.

The Cortex®-M33 processor also generates an INVSTATE UsageFault exception if the ICI bits are set to any nonzero value for the following instructions, as these instructions are not eligible for continuation:

- An integer Load Multiple instruction with the base register in the register list, and ICI set to a greater register number than the base register.
- An integer Store Multiple instruction with base write-back and with the base register in the register list.

The INVSTATE UsageFault exception takes precedence over any other instruction-related fault type, including NOCP or UNDEFINSTR UsageFault.

C.6 Stack limit checking

The Arm®v8-M architecture defines the instructions which are subject to stack limit checking when operating on SP.

It states that it is **UNKNOWN** whether a stack limit check is performed on any use of the SP that was **UNPREDICTABLE** in Armv6-M and Armv7-M. In the Cortex®-M33 processor, these **UNPREDICTABLE** cases are when R13 is used as a general purpose register in instructions. In these circumstances, the processor generates an UNDEFINSTR UsageFault exception.

C.7 UNPREDICTABLE instructions within an IT block

Instructions executed in an IT block which change the PC are architecturally **UNPREDICTABLE** unless they are the last instruction in the block.

In the Cortex®-M33 processor:

- Conditional branch instructions (`Bcond label`) always generate an unconditional UNDEFINSTR UsageFault exception.
- unconditional branch instructions (`B label`) which are not the last instructions in the IT block generate an unconditional UNDEFINSTR UsageFault exception.
- Branch with link instructions (`BL label`) which are not the last instructions in the IT block generate an unconditional UNDEFINSTR UsageFault exception. `BLX` PC is always **UNPREDICTABLE** and generates an UNDEFINSTR UsageFault exception.
- Branch and exchange instructions (`BX Rm`) which are not the last instructions in the IT block generate an unconditional UNDEFINSTR UsageFault exception.
- Compare and Branch instructions (`CBNZ` and `CBZ`) always generate an unconditional UNDEFINSTR UsageFault exception.
- Table branch instructions (`TBB` and `TBH`) which are not the last instructions in the IT block generate an unconditional UNDEFINSTR Usage Fault exception.
- An `IT` instruction inside another IT block always generates an unconditional UNDEFINSTR UsageFault exception.

- Data processing instructions which have PC as the destination register and are not architecturally **UNPREDICTABLE** outside an IT block generate an unconditional UNDEFINSTR UsageFault exception unless they are the last instruction of the IT block.
- Load instructions (`LDR`, `LDM`, and `POP`) which have PC as the destination register and are not architecturally **UNPREDICTABLE** outside an IT block generate an unconditional UNDEFINSTR UsageFault exception unless they are the last instruction of the IT block.
- If the Arm®v8-M floating-point extension is included and one of the following instructions is executed in an IT block, the instruction behaves as a regular conditional instruction according to the position of the instruction in the IT block:
    - `VCVTA`.
    - `VCVTN`.
    - `VCVTP`.
    - `VCVTM`.
    - `VMAXNM`.
    - `VMINNM`.
    - `VRINTA`.
    - `VRINTN`.
    - `VRINTP`.
    - `VRINTM`.
    - `VSEL`.
- Change Processor State instructions (`CPS`) always generate an unconditional UNDEFINSTR UsageFault exception.

C.8 Memory access and address space

In the Arm®v8-M architecture, the following conditions apply.

- Any access to memory from a load or store instruction or an instruction fetch which overflows the 32-bit address space is **UNPREDICTABLE**. In the Cortex®-M33 processor, these accesses wrap around to addresses at the start of memory.
- Any unaligned access that is not faulted by the alignment restrictions and accesses Device memory has **UNPREDICTABLE** behavior. In the Cortex®-M33 processor, accesses of this type generate an UNALIGNED UsageFault exception.
- For any access X, the bytes accessed by X must all have the same memory type attribute, otherwise the behavior of the access is **UNPREDICTABLE**. That is, an unaligned access that spans a boundary between different memory types is **UNPREDICTABLE**. In the Cortex®-M33 processor, each part of an access to a different 32-byte aligned region is dealt with independently. If an MPU is included in the processor, each access to a different 32-byte region makes a new MPU lookup. If an MPU is not included, then the behavior of the associated background region is taken into account.

- For any two memory accesses X and Y that are generated by the same instruction, the bytes accessed by X and Y must all have the same memory type attribute otherwise the results are **UNPREDICTABLE**. For example, an `LDC`, `LDM`, `LDRD`, `STC`, `STM`, `STRD`, `VSTM`, `VLDM`, `VPUSH`, `VPOP`, `VLDR`, or `VSTR` that spans a boundary between Normal and Device memory is **UNPREDICTABLE**. In the Cortex®-M33 processor, each part of access to a different 32-byte aligned region is dealt with independently. If an MPU is included in the processor, each access to a different 32-byte aligned region makes a new MPU lookup. If an MPU is not included, then the behavior of the associated background region is taken into account.
- Any instruction fetch must only access Normal memory. If it accesses Device memory, the result is **UNPREDICTABLE**. For example, instruction fetches must not be performed to an area of memory that contains read-sensitive devices because there is no ordering requirement between instruction fetches and explicit accesses. In the Cortex®-M33 processor, fetches to Device memory is sent out to the system, indicated on the AHB interface as Device, unless the memory region is marked with the *Execute Never* (XN) memory attribute.
- If the Arm®v8-M Security Extension is implemented, the behavior of sequential instruction fetches that cross from Non-secure to secure memory and fulfill the secure entry criteria specified in the architecture, including the presence of a *Secure Gateway* (`SG`) instruction at the boundary of the secure memory area, is **CONSTRAINED UNPREDICTABLE**. In the Cortex®-M33 processor, this results in the transition to Secure state.

C.9 Load exclusive and Store exclusive accesses

Instructions which can generate an exclusive memory access such as `LDREX` and `STREX` have a number of restrictions and behavior defined as **UNPREDICTABLE** in the Arm®v8-M architecture.

In the Cortex®-M33 processor:

- Exclusive accesses to memory regions marked as Device outside of the PPB region behaves the same as an equivalent access to shared Normal memory. All Device memory is shared in Arm®v8-M.
- Exclusive accesses to the PPB memory region (`0xE0000000:0xE00FFFFF`) do not update the internal local exclusive monitor. Load exclusive instructions load data into a register and Store exclusive instructions store data from a register. For `STREX` and `STLEX` instructions, the status register is always updated with the value 0, indicating the store has updated memory.
- The internal exclusive monitor does not tag addresses and the reservation granule is the whole of the memory. This means exclusive Load and Store instruction pairs that only use the local monitor are not affected by the address used for the access or the data size or the attributes associated with the memory regions. The behavior of **UNPREDICTABLE** exclusive accesses to external memory depends on the global exclusive monitor in your system.

C.10 Armv8-M MPU programming

The Arm®v8-M *Protected Memory System Architecture* (PMSA) includes a number of **UNPREDICTABLE** cases when programming the MPU when it is included in an implementation.

In the Cortex®-M33 processor:

- Setting MPU_CTRL.ENABLE to 0 and MPU_CTRL.HFNMIEA to 1 is **UNPREDICTABLE**. This results in all memory accesses using the default memory map including those from Exception Handlers with a priority less than one.
- If MPU_RNR is written with a region number greater than the number of regions defined in the MPU, then the value used is masked by one less than the number of regions defined. For example:
    - The number of regions defined is given as num_regions. The value written to MPU_RNR is given as v.
    - num_regions=8 and v=9.
    - The effective region used is given as 9 & (8-1); region 1.

  The number of regions available can be read from MPU_TYPE.DREGION.
- Setting MPU_RBAR.SH to 1 is **UNPREDICTABLE**. This encoding is treated as Non-shareable.
- The Attribute fields (MPU_ATTR) of the MPU_MAIR0 and MPU_MAIR1 registers include some encodings which are **UNPREDICTABLE**.
    - If MPU_ATTR[7:4]!=0 and MPU_ATTR[3:0]==0 is **UNPREDICTABLE**, the attributes are treated as Normal memory, Outer non-cacheable, Inner non-cacheable.
    - If MPU_ATTR[7:4]==0 and MPU_ATTR[1:0]!=0 is **UNPREDICTABLE**, the attributes are treated as Device-nGnRE.
- The external AMBA AHB5 interface signals cannot distinguish between some of the memory attribute encodings defined by the Arm®v8-M PMSA:
    - Normal transient memory is treated the same as Normal non-transient memory.
    - Device memory with gathering or Reordering attributes (G, R) are always treated as non-Gathering and non-Reordering. Early Write Acknowledgment attributes (E, nE) are supported on the Cortex®-M33 AHB5 interfaces.

C.11 Miscellaneous UNPREDICTABLE instruction behavior

This section documents the behavior of the Cortex®-M33 processor in a number of miscellaneous **UNPREDICTABLE** instruction scenarios:

- Load instructions which specify writeback of the base register are **UNPREDICTABLE** if the base register to be written back matches the register to be loaded (Rn==Rt). In the Cortex®-M33 processor, the base register is updated to the loaded value.

- Store instructions which specify writeback of the base register are **UNPREDICTABLE** if the base register to be written back matches the register to be stored (Rn==Rt). In the Cortex®-M33 processor, the value stored is the initial base register value. The base register is then written back with the expected updated value.
- Multiply and Multiply accumulate instructions which write a 64-bit result using two registers, `SMULL`, `SMLAL`, `SMLALBB`, `SMLALBT`, `SMLALTB`, `SMLALTT`, `SMLALD`, `SMLALDX`, `SMLSLD`, `SMLSLDX`, `UMULL`, and `UMAAL` are **UNPREDICTABLE** if the two registers are the same (RdHi==RdLo). In the Cortex®-M33 processor, these cases generate an UNDEFINSTR UsageFault exception.
- Floating-point instructions which transfer between two registers and either two single precision registers or one double precision register, `VMOV Rt, Rt2, Dm` and `VMOV Rt, Rt2, Sm, Sm1` are **UNPREDICTABLE** if the two registers are the same (Rt==Rt2). In the Cortex®-M33 processor, these cases generate an UNDEFINSTR UsageFault exception.

Appendix D Revisions

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

D.1 Revisions

The following tables show the technical changes between released issues of this book.

Table D-1: Issue 0000-00

Change	Location
First release	-

Table D-2: Differences between issue 0000-00 and issue 0001-00

Change	Location
Updated CPUID reset value	4.1 Identification register summary on page 364.3 CPUID Base Register on page 42
Revised the functional block diagram and associated note	2.4 Component blocks on page 17
Revised the memory model description	3.4 Memory model on page 30
Revised the exception handling and prioritization in Secure and Non-secure state description	3.7.1 Exception handling and prioritization on page 34
Removed a redundant sentence 'Registers not described here are described in the Arm®v8-M Architecture Reference Manual'	6.1 NVIC programmers model on page 49
Revised the usage restrictions description	8. External coprocessors on page 56
Removed footnote in the ITM register summary table	11.1.1 ITM register summary table on page 73
Clarified that the functionality of the INT_ATVALID and INT_ATREADY Registers is only present in integration mode	11.1.5 Integration Mode Read ATB Ready Register on page 7611.1.4 Integration Mode Write ATB Valid Register on page 76

Table D-3: Differences between issue 0001-00 and issue 0002-00

Change	Location
Updated CPUID reset value	4.1 Identification register summary on page 364.3 CPUID Base Register on page 42
Updated AHB-AP Identification value	A.4.1.7 AHB-AP Identification Register, IDR, 0xFC on page 96

Table D-4: Differences between issue 0002-00 and issue 0003-00

Change	Location
Updated CPUID reset value.	4.1 Identification register summary on page 364.3 CPUID Base Register on page 42
In the third paragraph, changed 'associated external memory access is marked as Non-secure' to 'associated memory access is marked as Non-secure'. Clarified use of the register SAU_CTRL.EN and SAU_CTRL.ALLNS bit fields.	5.1 About security attribution and memory protection on page 45/>
Corrected the regions that show in the example of highest security level region	Table 5-1: Examples of Highest Security Level Region on page 45

Change	Location
Corrected the NVIC short description register names	6.1.1 NVIC register summary on page 49
Corrected the FPU exception flags names	7.2.4 Teal Exceptions on page 53
In the note, changed CPACR[2n+1:2n] to CPACR[2n+1:2n]	8.5 Configuring which coprocessors are included in Secure and Non-secure states on page 57
Corrected the ROM table value for when the ETM is not implemented. Changed `0xFFF42003` to `0xFFF42002`	10.1.3 Processor ROM table identification and entries on page 68

Table D-5: Differences between issue 0003-00 and issue 0004-00

Change	Location
Updated CPUID reset value	Table 4-1: Identification register summary on page 36 4.3 CPUID Base Register on page 42
Corrected the INVSTATE UsageFault exception statement at the end of the Exception-continuable instruction description	C.5 Exception-continuable instructions on page 124

Table D-6: Differences between issue 0004-00 and issue 0100-03

Change	Location
Updated CPUID reset value	• Table 4-1: Identification register summary on page 36 • 4.3 CPUID Base Register on page 42
Added content for new support for the *Custom Datapath Extension* (CDE) with *Arm Custom Instructions*(ACIs)	• 2.2 About the processor architecture on page 15 • 2.3 Processor configuration options on page 16 • 2.4 Component blocks on page 17 • 2.4.1 Processor core on page 19 • 2.4.3 Floating-Point Unit on page 19 • 2.6 Compliance on page 22 • 2.9 Product revisions on page 26 • 3.3 Instruction set summary on page 29 • 8.1 About external coprocessors on page 56 • 9. Arm Custom Instructions on page 60
Added register description	4.4 Auxiliary Feature Register 0 on page 43
Added SBIST bit description	4.2 Auxiliary Control Register on page 41
Fixed bit description	11.1.4 Integration Mode Write ATB Valid Register on page 76
Added statement about the number of breakpoints supported	14.1 About the Breakpoint Unit on page 84
Fixed PPB address range	3.4 Memory model on page 30
Fixed typo in architecture version	• 3.1 About the programmers model on page 28 • 3.5 Exclusive monitor on page 32 • 7.1 About the FPU on page 52

Change	Location
Unmerged Secure and Non-secure information for Identification registers	4.1 Identification register summary on page 36
Unmerged Secure and Non-secure information for NVIC registers	6.1.1 NVIC register summary on page 49
Fixed register descriptions	• 11.1.5 Integration Mode Read ATB Ready Register on page 76 • 11.1.4 Integration Mode Write ATB Valid Register on page 76

Table D-7: Differences between issue 0100-03 and issue 0100-07

Change	Location
Updated the address information for DWT_DEVARCH and DWT_DEVTYPE	12.2 DWT programmers model on page 78
Updated the function description for bit [1]	B.3.8 Integration Test ATB Control 0 Register on page 119
Updated the register names for 0xE000EFBC and 0xE002EFBC addresses	4.1 Identification register summary on page 36
Added a new statement on DWT usage	12.1 DWT functional description on page 78
Updated register names	10.1.4.1 SCS CoreSight identification on page 70