System Design with ARMv8-M

Version 1.0

ARM

ARM 100767_0100_0100_en

System Design with ARMv8-M

Release Information

Document History

Issue	Date	Confidentiality	Change
0100	02 September 2016	Non-Confidential	First release

Non-Confidential Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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.

Web Address

http://www.arm.com

Contents
System Design with ARMv8-M

Preface

This preface introduces the *System Design with ARMv8-M* .

It contains the following:

About this book

The ARM®v8-M C Language Extensions (ACLE) for ARMv8-M enable you to use the ARMv8-M Security Extension to build a Secure image, and to enable a Non-secure image to call a Secure image.

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

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

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

Intended audience

Using this book

This book is organized into the following chapters:

Chapter 1 System Design for ARM®v8-M

This system design illustrates a simple system with the key extra components and logic that are required to support an ARM®v8-M-based microcontroller with ARM TrustZone® technology for ARMv8-M.

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.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace`

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

`<and>`

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

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.

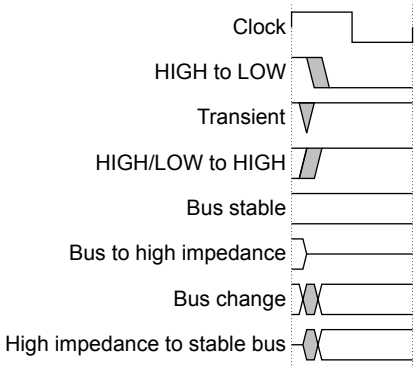


Figure 1 Key to timing diagram conventions

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 denotes an active-LOW signal.

Feedback

Feedback on this product

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

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

Feedback on content

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

- The title *System Design with ARMv8-M* .
- The number ARM 100767_0100_0100_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

——— **Note** ———

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

———————————

Chapter 1
System Design for ARM®v8-M

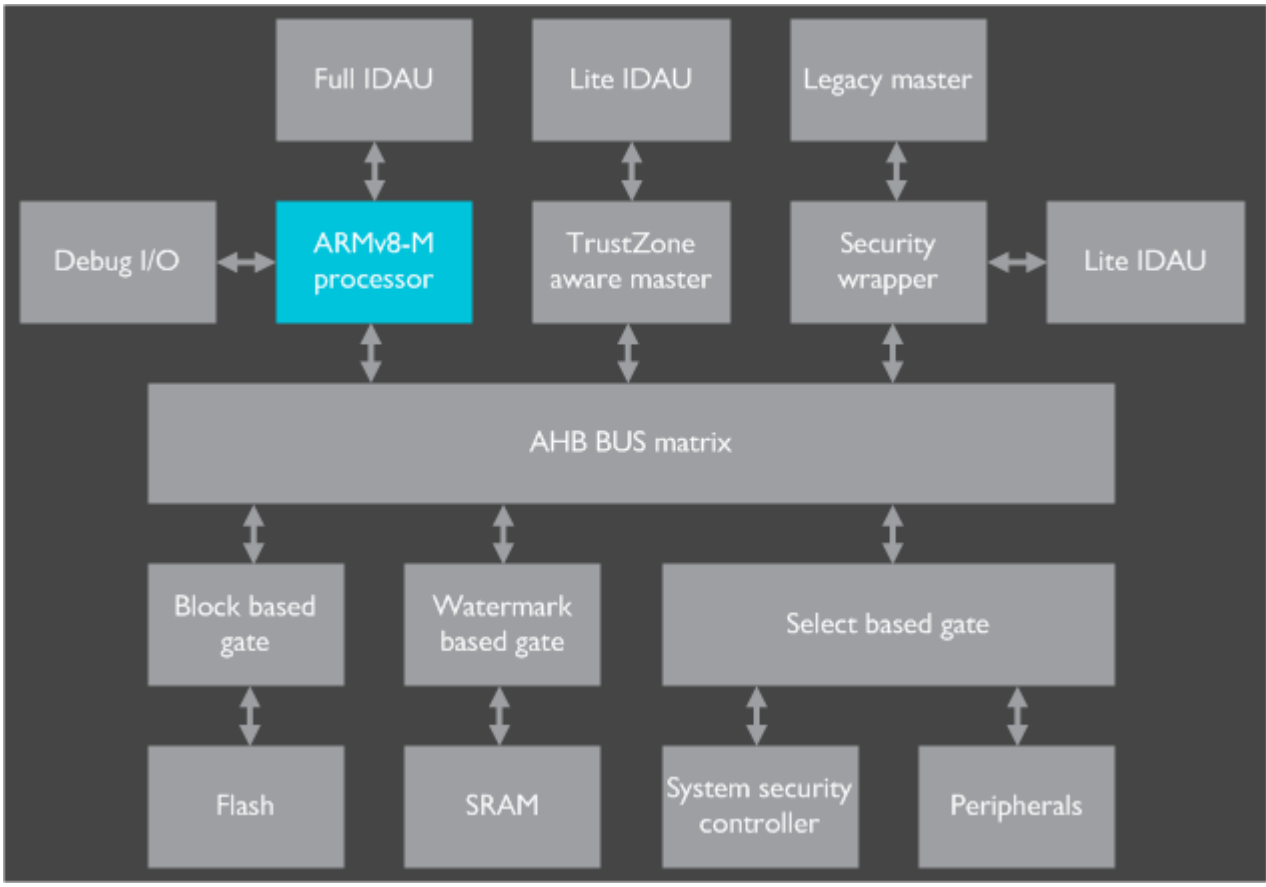
This system design illustrates a simple system with the key extra components and logic that are required to support an ARM®v8-M-based microcontroller with ARM TrustZone® technology for ARMv8-M.

It contains the following sections:

1.1 Example system design

The example system design describes a simple system with the key extra components and logic that are required to support an ARMv8-M-based microcontroller with ARM TrustZone technology for ARMv8-M. In addition to the components shown, Secure access-only components at addresses exempt from attribution can also exist in the system.

The following figure shows the example system design:



The following table describes the components of the example system:

Table 1-1 Example system components

Component	Description
Full IDAU	*Implementation Defined Attribution Unit* (IDAU) informs an ARMv8-M master of the system-specific security address map, along with region identification numbers and whether a particular address is Non-secure callable.
Lite IDAU	Informs a TrustZone technology aware master or security wrapper of the system-specific security address map.
Security wrapper	Maps a legacy master as either Secure or Non-secure and applies Secure attribution that is based on the address map.
Block-based gate	Conditionally rejects transactions for an address region that is divided into blocks or pages.
Watermark-based gate	Conditionally rejects transactions for an address region that is divided into two using a movable watermark value.

Table 1-1 Example system components (continued)

Component	Description
Select-based gate	Conditionally rejects transactions for an address region containing multiple slaves that are based on device select lines.
System security controller	System controller containing the control fields that drive the rest of the security components.

Related concepts

1.4.1 Full IDAU on page 1-14.
1.4.3 Lite IDAU on page 1-15.
1.5 Security wrappers on page 1-16.
1.5 Security wrappers on page 1-16.
1.6 Block-based gate on page 1-17.
1.7 Watermark-based gate on page 1-18.
1.8 Select-based gate on page 1-19.
1.9 System initialization considerations on page 1-20.

1.2 Memory system and memory partitioning

If the ARMv8-M Security Extension is implemented the 4GB memory space is partitioned into Non-secure and Secure memory regions.

Non-secure (NS)

Non-secure transactions are those that originate from masters operating as, or deemed to be, Non-secure or from Secure masters accessing a Non-secure address. Non-secure transactions are only permitted to access NS addresses, and the system must ensure that NS transactions are denied access to Secure addresses.

The Secure memory space is further divided into two types, Non-secure Callable and Secure.

Non-secure Callable (NSC)

NSC is a special type of Secure location. This type of memory is the only type which an ARMv8-M processor permits to hold a Secure Gateway (SG) instruction that enables software to transition from Non-secure to Secure state. The inclusion of NSC memory locations removes the need for Secure software creators to allow for the accidental inclusion of SG instructions, or data sharing encoding values, in normal Secure memory by restricting the functionality of the SG instruction to NSC memory only.

Secure

Secure addresses are used for memory and peripherals that are only accessible by Secure software or Secure masters.Secure transactions are those that originate from masters operating as, or deemed to be, Secure when targeting a Secure address.

1.3 System security controller

The system security controller is a peripheral component accessible only by Secure transactions. The controller holds all the register fields that are used to modify the configurable aspects of the system security. For example, for a full *Implementation Defined Attribution Unit* (IDAU), whether an address region is NSC, which peripherals are accessible by NS transactions, and where the boundary between Secure and Non-secure memory is in RAM.

The system security controller might include logic for monitoring violations that are detected by the security gates, and consolidating any events into a Secure interrupt, that targets a Secure processor within the system.

The peripheral register map and bit assignments are not described further. All references to control here imply the existence of an associated software modifiable field within the system security controller.

In addition to only being accessible by Secure software, the system security controller can also be locked. For example, when Secure software has programmed the controller, it can be locked to prevent further accidental modification until the system is reset.

The system security controller shares some similarities with existing power and clock controllers found in microcontrollers (MCUs), and as such inclusion of this functionality in the same or neighboring block might make sense.

Note

The number and location of system security controllers is left to the implementer.

1.4 Implementation Defined Attribution Unit (IDAU)

The ARMv8-M architecture defines a mechanism whereby an implementation can define whether any particular address is exempt from checking, is NSC or Secure. If the ARMv8-M Security Extension is included, then the internal *Secure Attribution Unit* (SAU) or an external *Implementation Defined Attribution Unit* (IDAU) determines the Security state attributed to each address.

The number of SAU regions is defined during the implementation of the processor. If no SAU regions are defined, or the SAU is disabled, and no IDAU is included in the system then the entire memory address space is Secure and the processor is not able to switch to Non-secure state.

This section contains the following subsections:

1.4.1 Full IDAU

ARMv8-M implementations contain a common IDAU interface, where an address is presented and a combinatorial response is required for the IDAU attributes corresponding to this address.

The scheme that is defined for the example system maps into bit[28] of the address being used to drive the Secure attribution signal, however, extra consideration is required regarding NSC and exempt regions for CoreSight™ ROM tables, and similar.

Non-secure Callable regions

ARMv8-M requires any Secure address which is to be used as a branch target for switching between Non-secure to Secure state to contain an `SG` instruction and be marked as NSC.

If a *Secure Attribution Unit* (SAU) is implemented, the SAU can attempt to mark a region as NSC. If the *Implementation Defined Attribution Unit* (IDAU) has marked the same region as Secure only, it would be the most restrictive and so prevent the region being treated as NSC. Similarly if the IDAU marks a region as *Non-secure Callable* (NSC), an implemented SAU could restrict the NSC region to Secure only.

For a system containing an ARMv8-M processor with an SAU implementing a suitable number of regions, one option might be for the IDAU to mark all locations that can contain Secure code as NSC. Alternatively, if an SAU is not implemented, or if extra flexibility is wanted, extra controls can be added to mark which IDAU regions are Secure or NSC.

ROM table exemptions

The use of ARM CoreSight technology for debug requires the inclusion of system level identification and ROM tables to enable an external debugger to automatically identify the devices it is connecting to.

The *Implementation Defined Attribution Unit* (IDAU) supports the concept of exempt regions, where a transaction is allowed to propagate regardless of its security attribute. The IDAU must therefore support some mechanism (likely synthesis time parameters) for specifying the location of these 4KB exemption windows for the ROM tables.

The exemption windows might also be used to provide access to an agent that authorizes Secure debug.

Note

Take care with the use of exempt regions. In particular, ARM recommends that any exempt region must fall within address spaces that are marked as execute-never (XN), or which prevent instruction execution by another mechanism.

Related concepts

1.1 Example system design on page 1-10.

1.4.2 Region numbers

ARMv8-M processors implement the TT instruction which is designed to enable interrogation of the internal memory protection units of the processor, the SAU, and the IDAU. A full IDAU must provide a region number for any address query. A full IDAU must also ensure that the region number is different between addresses which have differing security attributes, or whose addresses span areas of non-contiguous security attribution.

1.4.3 Lite IDAU

TrustZone technology-aware masters, other than ARMv8-M processors and any security wrappers, require knowledge of the security address map. A 'lite' version of the *Implementation Defined Attribution Unit* (IDAU) provides Secure or Non-secure attribution, but does not have to implement the *Non-secure Callable* (NSC) or region number aspects.

Related concepts

1.1 Example system design on page 1-10.

1.5 Security wrappers

Transactions from legacy or non-security aware masters require appropriate attribution when connected into a system with TrustZone technology for ARMv8-M. The example system utilizes security-wrappers to perform this function.

For each security wrapper, a control bit exists to determine whether the master must behave as either a Secure or a Non-secure master. The wrapper uses the control bit to determine a secure-request type and then the Secure attribute:

- If the request type is Secure, it uses the system address map to determine if the secure-attribute of the transaction must be Secure or Non-secure based on the address.
- If the request type is Non-secure, the secure-attribute is Non-secure.

The example system assumes that rejection of non-permitted transactions is performed at ingress to the slave components. However, it is permitted, but not required, for the security wrapper to reject a Non-secure transaction that is targeting a Secure address. In such a case the example system records an event in the system security controller, which in turn generates a Secure interrupt to the ARMv8-M processor.

Note

The mechanism that is used to reject the transaction is entirely up to the implementer and includes generating an abort or ignoring-writes and returning zero-on-a-read. The mechanism that is used to record an event, and generation of an interrupt is also entirely within the freedom of the implementer.

1.5.1 Programmable masters

Care must be taken with programmable masters, for example, DMAs, and display control. If the master is permitted to generate Secure transactions, ensure that doing so does not provide a mechanism for Non-secure software to control transactions in Secure memory space, such as permitting Non-secure software to configure a DMA transaction from Secure to Non-secure space.

At a first level, security aware DMAs might avoid this problem by recording the secure-attribute of the transaction that programmed the DMA and then using the secure-attribute of the transaction to ensure that DMA actions initiated by a Non-secure transaction always produce Non-secure DMA operations.

Note

Designers of such peripherals must thoroughly consider the implications of enabling two Security states to co-exist within a single master, including (but not limited to) any potential leakage resulting from partial programming in one state.

A possible non-security aware solution might be for the system to be configured by Secure software, either to force all DMA master transactions to Non-secure and permit Non-secure programming access, or to permit Secure DMA master transactions but not permit programming from Non-secure. However, in such a system allowing Secure DMA transactions and Non-secure DMA programming would not be permitted, and requires co-ordination between the configuration of the security wrapper and the (likely) associated select-based gate for the slave side of the DMA.

Related concepts

1.1 Example system design on page 1-10.
1.1 Example system design on page 1-10.

1.6 Block-based gate

In the example system design shown previously, a block-based gate is used to protect and allocate regions of memory which are subdivided into blocks. A control is implemented for each block which identifies whether the block is treated as Secure or Non-secure. At reset, or unless configured using alternative means, all blocks revert to Secure access only, and rely on Secure software to assign blocks as Non-secure. The control also determines in which alias a particular block appears.

The gate has two hard requirements:

1. It must not-permit Non-secure transactions to read/write blocks that the controls mark as Secure.
2. It must not-permit Secure transactions to read blocks that the controls mark as Non-secure.

In addition to other less obvious attack vectors, requirement 2 is designed to prevent Non-secure software using the aliasing to install code, including SG instructions into Secure memory and then calling into it.

In addition to the two hard requirements, it is also permissible to not permit Secure writes to Non-secure blocks.

Within the example system, the rejection method that is used for non-permitted transactions is to ignore writes and to return zero on reads, along with providing event information to the system security controller, which in turn would generate an interrupt to the ARMv8-M processor. Alternative mechanisms, such as generating bus aborts, are also feasible though potentially less desirable, in particular if an alternative rejected read default value is chosen it must never be the SG instruction.

Although this mechanism is suitable for protecting the read interface of a flash controller, extra care must be taken with the programming interface.

Note

There is no restriction on an implementer choosing a more complex decoding scheme. For example, one that produces contiguous memory at a constant base address for each security domain.

Related concepts

1.1 Example system design on page 1-10.

1.7 Watermark-based gate

In addition to its use for flash protection, the block-based scheme can also be used to protect RAM. For completeness, the example system in Figure 1 shows an alternative mechanism involving the use of a watermark scheme for the protection of RAM.

The example system assumes the use of a single SRAM component for system memory. Without prior knowledge of the Secure software that is installed on the device, one mechanism for partitioning the memory is a watermark control. This control specifies an address at which the SRAM transitions from being Non-secure to Secure.

From reset the watermark is set to a value, or disabled, so that all SRAM is marked as Secure. During initialization, the Secure software sets the watermark to a value to transfer the SRAM it does not require to the Non-secure side.

As SRAM can be used to hold code and data, the same rules apply for preventing SG instruction injection from the Non-secure side, along with the same rules for transaction rejection as per the block-based gate.

Note

Implementers are free to build schemes with both multiple regions and multiple RAMs. Also, there is no restriction on using the watermark-based scheme only for RAMs or the block-based scheme only for flash. In particular, it might be preferable to use a block-based scheme for both flash and RAM.

Related concepts

1.1 Example system design on page 1-10.

1.8 Select-based gate

Interconnects perform address decoding and use address decoding to select which peripheral a transaction is targeted to, which results in the generation of a select signal for each peripheral.

In the case where a Non-secure transaction targets a peripheral that is not assigned to Non-secure, the transaction is rejected. The rejection might be performed by the gate itself or by using the selection of a default-slave like component to handle the transaction.

At reset the select-based gate is set to configure all peripherals as Secure. During initialization, the Secure software can make Non-secure any peripherals that it does not require.

Note

The requirement that a peripheral register cannot be fetched by an instruction is intended to prevent storing an SG instruction in a peripheral and then branching to that location. This requirement can be met by having the peripheral, or interconnect block, reject fetch transactions to the peripheral.

Related concepts

1.1 Example system design on page 1-10.

1.9 System initialization considerations

You must consider processor reset, system security controller default values, and debug security when initializing the system.

1.9.1 Processor reset

An ARMv8-M processor with the ARMv8-M Security Extension comes out of reset in the Secure state at the address the *Secure Vector Table Offset Register* (VTOR_S) specifies.

ARM implementations provide a mechanism for specifying the default value of this register, using pins or a configuration parameter. The implementer must ensure that this mechanism is configured to point at a Secure memory location. Attempting to execute code from a Non-secure location while in the Secure state results in a fault condition within the processor.

1.9.2 System security controller default values

In the general case, the system security controller causes all configurable processors and memories to be inaccessible to Non-secure transactions out of reset. Subsequent release of components to the Non-secure state can then be performed using Secure software during the boot process.

1.9.3 Debug security

Debug security must factor in all stages of product deployment. One methodology is to produce devices where Secure debug is enabled from production. A subsequent programming phase can then disable Secure debug, restricting the device so that an authorization mechanism (with optional erase) would be required to enable the use of Secure debug.

Related concepts

1.1 Example system design on page 1-10.

1.10 Component-specific considerations

This section describes issues that implementers must also be aware of.

This section contains the following subsections:

1.10.1 Flash or NVM programming

Take care when implementing or instantiating a flash or other non-volatile memory controller that is intended to be shared by both Secure and Non-secure code.

In particular, if it is necessary to permit Non-secure code or a Non-secure debugger to erase code blocks that are allocated to Non-secure code, an implementer must consider how they permit erasure while simultaneously preventing Non-secure code or debug from erasing or modifying Secure blocks.

Sometimes it might not be necessary to permit Non-secure flash updates. An implementer might choose to implement extra controls to enable or disable Non-secure programming.

1.10.2 Security Attribution Unit

For the example system, it is assumed that the *Implentation Defined Attribution Unit* (IDAU) is used to configure the security of each address. Other models, particularly when deploying an ARMv8-M processor with a larger application processor style system-on-chip, might benefit from the flexibility that is offered by using the internal SAU as the primary mechanism.

1.10.3 General-purpose I/O

It can be desirable to allocate certain I/O pins to either Secure or Non-secure state. This allocation might require replication of the GPIO interface, and in particular might require the addition of per-pin Secure and Non-secure assignment bits, implemented to prevent Non-secure software from updating the Secure GPIO pins.