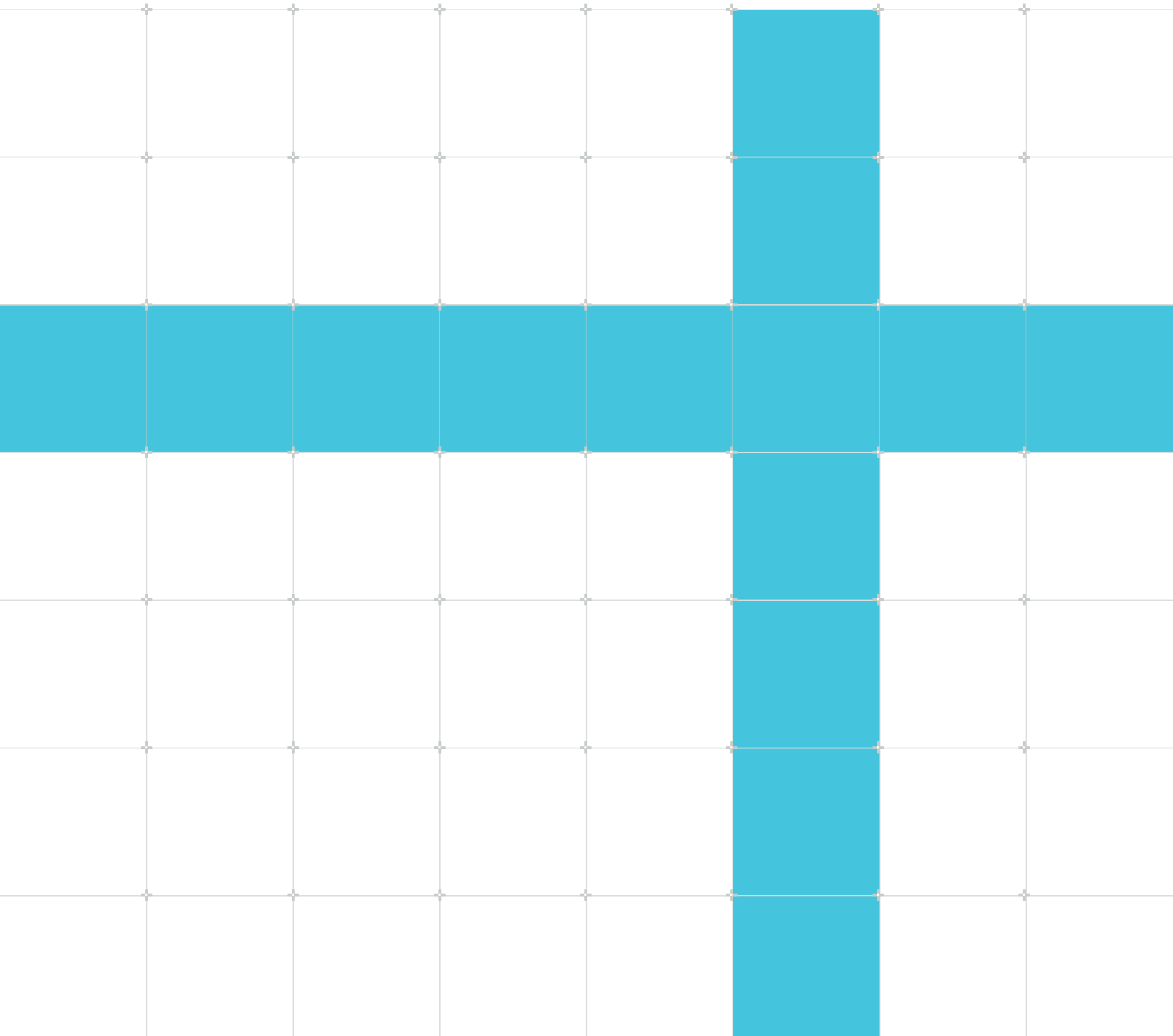
TrustZone Technology Microcontroller System Hardware Design Concepts

Version 1.0

User Guide

Issue 01
107779_0100_01_en



TrustZone Technology Microcontroller System Hardware Design Concepts
User Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	1 January 2023	Non-Confidential	First release

Proprietary Notice

and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

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

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

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Scope

The Arm® v8-M Architecture adds support for the isolation of Secure and Non-secure software running on a single processor. In addition to using an Armv8-M based processor, creation of a TrustZone® platform requires further consideration within the system. This document focuses on some of the design considerations when building such a system with simple examples and highlights the possible implementation and its associated components.

2. Introduction

TrustZone technology can be used to implement a PSA Certified root of trust. The PSA Certified scheme is a common industry framework and methodology for built-in security, enabling silicon manufacturers, system software providers, and OEMs to ensure the security of connected products using a proven security architecture and corresponding open-source implementations. For more information see https://www.arm.com/architecture/psa-certified

Adding and implementing the optional Security extension in Armv8-M places additional burdens on a microcontroller system. This document describes a simple scheme suitable for use in such a system as a means of aiding understanding the requirements of TrustZone technology enabled microcontroller system design.

This section describes some of fundamental concepts from Armv8-M:

2.1 Attribution of security based on memory address

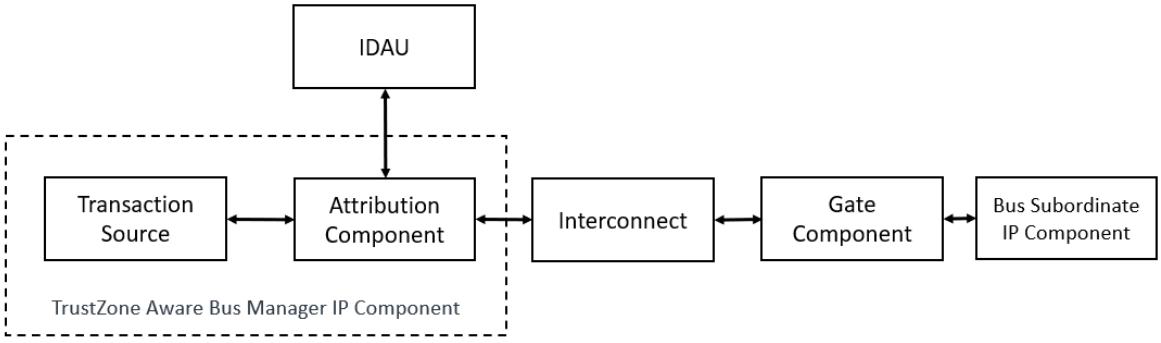
Armv8-M's TrustZone technology determines security based on address, any particular address is considered either Secure or Non-secure. From an Armv8-M processor point of view, this attribution can be performed either: - From within the processor via a private Security Attribution Unit (SAU) - At a system level via the Implementation Defined Attribution Unit (IDAU).

Within a microcontroller context, this document assumes that the primary mechanism for determining the security of any given address is via the IDAU.

2.2 Example of transaction flow in an Armv8-M TrustZone system

The figure below shows the flow that a transaction follows through key component types in the system.

Figure 2-1: Figure-1



In Figure 2-1: Figure-1 on page 9 the bus manager IP component could be an Armv8-M processor or some other bus manager designed to support TrustZone system security features. This figure shows that the bus manager component is partitioned to be independent from system-specific Secure or Non-secure memory mappings. This is achieved by positioning the IDAU outside the bus manager IP component.

Transactions are generated within the bus manager and then pass through a security attribution process before leaving the bus manager. The transaction is then routed to a specific bus subordinate according to the address map implemented in the interconnect. In this example the address decoder, within the interconnect, ignores the transaction secure-attribute. The interconnect merely transmits the security attribute along with the rest of the transaction information. Before the transaction can reach a subordinate it must past through a security gate component.

When the transaction is passed into the attribution component the request type and the transaction address are used to determine the secure-attribute of the transaction that will be passed to the interconnect. Once the transaction has been routed by the interconnect to the target subordinate it is the responsibility of the gate component to guarantee that the secure-attribute of the transaction is compatible with the memory type of the subordinate. If a security violation is detected then the gate must not allow the transaction to proceed.

2.3 Secure Gateway (SG) instruction

For an Armv8-M processor function calls can switch from Non-secure to Secure state using the execution of a Secure Gateway (SG) instruction. Secure code uses SG instructions to mark legal

entry points into Secure state. For example, the start of a Secure function, and Non-secure code might call Secure functions by branching to the address of an SG instruction. Additional restrictions apply to the SG instruction (See Non-secure Callable (NSC) memory type below).

2.4 Memory types

Armv8-M defines three types of memory, as defined in the following sections.

An exempt memory type also exists. See section ROM table exemptions

2.4.1 Non-secure (NS)

Non-secure addresses are used for memory and peripherals accessible by all software running on the device. Non-secure transactions originate from manager operating as Non-secure or from Secure managers accessing a Non-secure address. Non-secure transactions are only permitted to access NS addresses. The system must ensure that NS transactions are denied access to Secure addresses.

Armv8-M provides additional memory protection mechanisms and privileged / unprivileged access concepts. You can use these to restrict access of certain addresses to certain software. This document does not describe these concepts. See Armv8-M Memory model and MPU User Guide for more details.

2.4.2 Secure (S)

Secure addresses are used for memory and peripherals accessible only by Secure software / managers. Secure transactions are those that originate from managers operating as Secure when targeting a Secure address, or an exempt memory type. See section ROM table exemptions.

2.4.3 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 an SG instruction that can allow software to transition from Non-secure to Secure state. The inclusion of NSC locations removes the need for Secure software creators to police for the accidental inclusion of SG instructions (or data sharing the same encoding value) in normal Secure memory by restricting the functionality of the SG instruction to NSC memory only.

3. Memory map

There are many possible approaches to mapping Secure and Non-secure regions across a memory map. One possible obvious approach is to take an existing device memory map and allow each memory/subordinate to have Secure and Non-secure locations scattered across them in a contiguous address range. Although this approach would be possible to implement with gate and IDAU components, there are potential challenges (see section Region numbers) in terms of the logic required and region granularity. The approach proposed in this guide enables the use of a very simple IDAU design. We employ extensive address region aliasing (duplication of all memory mapped components in both secure and non-secure address regions) within the interconnect address decoders and then rely on gate components to completely remove this duplication so that each physical memory location is only visible in either a secure region or a non-secure region

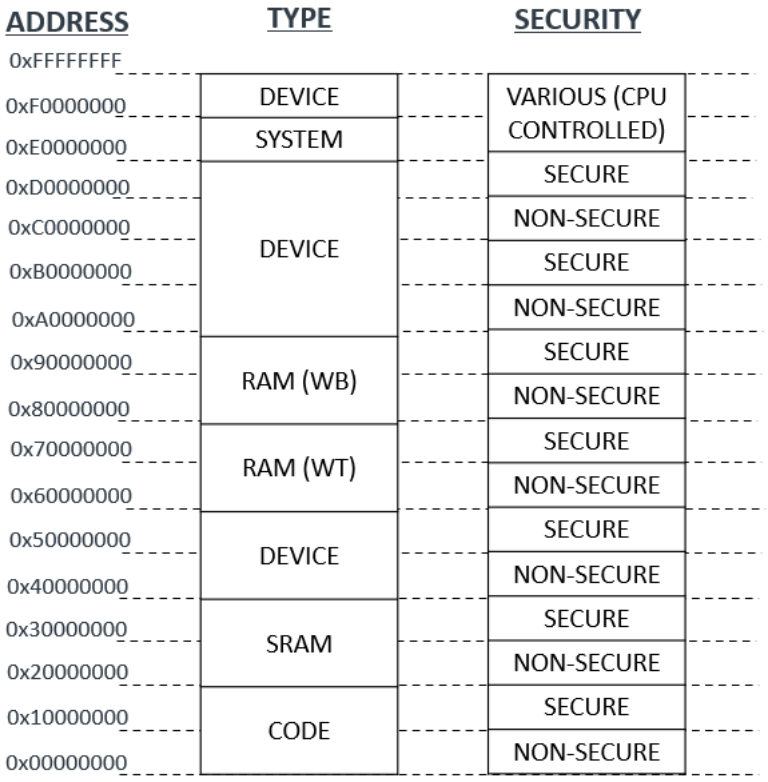
The term “aliasing” can be confusing in this context. To be clear, we are not making a physical memory location or peripheral register accessible simultaneously at two separate addresses. Instead it will only be accessible in either a secure or a non-secure region at any point in time. This guide will consistently use the term “aliasing”.

3.1 Aliasing

Armv8-M defines a memory map divided on 512MB boundaries.The example system adds support for Security by aliasing each of these at their halfway point such that: - The lower half provides access to a 256MB Non-secure window - The upper half provides a view of the same 256MB region but this time as Secure.

Control points elsewhere in the system (see Implementation Defined Attribution Unit (IDAU)) will determine what is accessible in each of the S and NS windows.

Figure 3-1: Figure-2



We recommend that you use a memory map similar to Figure 3-1: Figure-2 on page 12 for practical systems.(See section Summary of recommendations). For this simple system, the result is that bit[28] of the address is a proxy for Secure vs Non-Secure.

For systems with additional constraints on addressing (for example those requiring to support a legacy address map), or where larger contiguous portions of memory are required, the implementer is free to adapt this scheme appropriately (while meeting certain requirements, see section Region numbers). Though aliasing concept employed by this example system is useful, it is not essential. Also, the choice of whether the upper or lower alias is S or NS, is arbitrary.


Note

Using bit[28] for both aliasing and defining Secure vs Non-secure is an example only and must not be relied upon by software.

3.2 Attribute transmission

AMBA AXI and AHB5 include support for transmitting the security attribute of a transaction.

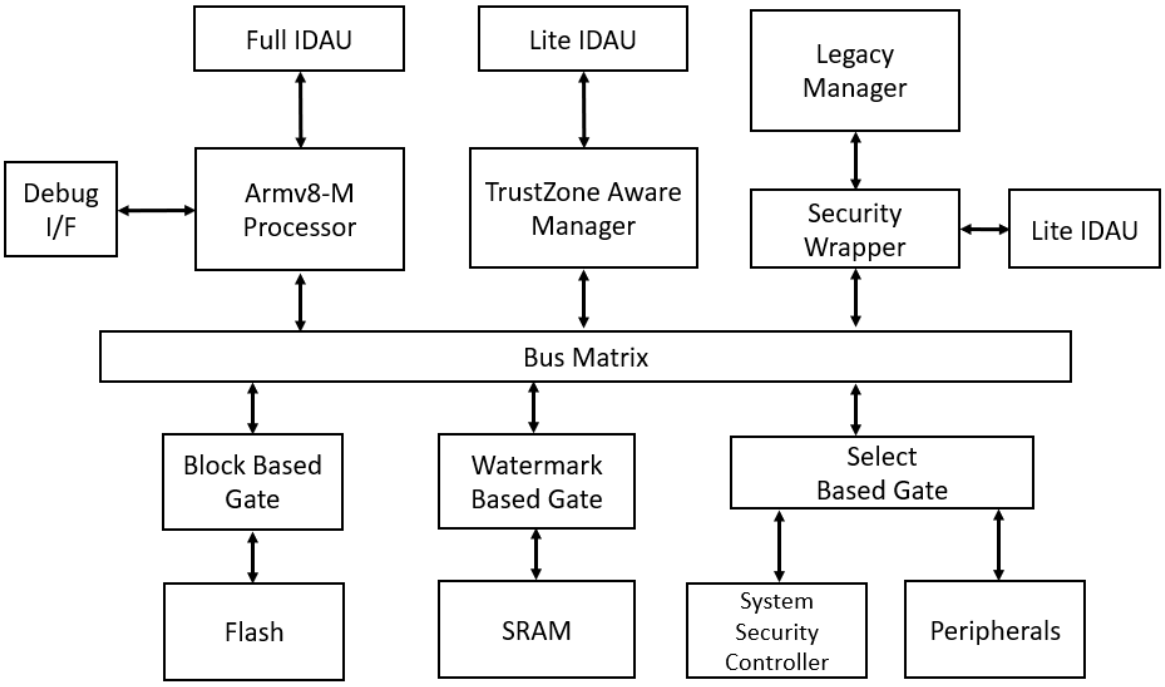

Note

Some existing interconnect and matrix products include the ability to specify additional “user” fields. You can propagate these fields to transfer the NS-attribute of a transaction.

4. Example System

The Figure 4-1: Figure-3 on page 14 shows a simple system with the key additional components required to support Armv8-M based microcontroller utilising TrustZone technology:

Figure 4-1: Figure-3



Component	Description	Section
Full IDAU	Informs a Armv8-M manager of system specific security address map, along with region identification numbers and whether a particular address is Non-secure callable	Full IDAU
Lite IDAU	Informs a TrustZone technology aware manager or security wrapper of the system specific security address map	Region numbers
Security wrapper	Maps a legacy manager as either Secure or Non-Secure and applies secure attribution based on address map	Security wrapper
Block-based gate	Conditionally rejects transactions for an address region divided into blocks / pages	Block-based Gate
Watermark-based gate	Conditionally rejects transactions for an address region divided into two via a movable watermark value	Watermark-based Gate
Select-based gate	Conditionally rejects transactions for an address region containing multiple subordinates based on device select lines	Select-based Gate
System security controller	System controller containing the control fields that drive the rest of the security components	System Security Controller

In addition to the components listed, additional Secure-access-only, or components at addresses exempt from attribution might exist in the system. To make this example system a complete illustration of all the different gate types, this guide shows the use of a watermark based gate for the SRAM component.


Note

It is recommended to use of a block based gate for on-chip SRAMs (see section Summary of recommendations).

5. System Security Controller

This section describes System security controller and its configuration.

5.1 Secure Configuration

The system security controller is a peripheral component accessible only by Secure transactions. The controller holds all of the register fields used to modify the configurable aspects of the system security. For example, for a full IDAU whether an address region is NSC, which peripherals are accessible by NS transactions, where the boundary between Secure and Non-secure memory is in RAM. The system security controller might include logic for monitoring violations detected by the security gates, and consolidating any events into a secure interrupt targeting a secure processor within the system. This guide does not describe the peripheral register map and bit assignments. However, all references to “control” within this guide 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 might also include the capability to be locked.For example, when Secure software has programmed the controller you might want to prevent further, possibly accidental, modification until the system is reset. The system security controller shares some similarities with existing power and clock controllers found in MCUs. Therefore, you might like to include this functionality in the same, or neighbouring, block might make sense.

The implementer can chose the number and location of system security controllers.

6. Implementation Defined Attribution Unit (IDAU)

This section explains some of the options possible in an Implementation Defined Attribution Unit (IDAU).

6.1 Armv8-M Processors

The Armv8-M Architecture defines a mechanism whereby an implementation defined mechanism (for example, a component in the surrounding system) can define whether any particular address is exempt from checking, is NSC or Secure. When the processor does not implement an internal Security Attribution Unit (SAU), the processor uses this information to determine the security attributes for any given transaction. When a SAU is present and enabled, the IDAU attributes determine the most permissive behaviour. For example, if the IDAU marks an address as Secure, then the SAU cannot make it Non-Secure, but if the IDAU marks it as Non-secure, then the SAU can mark the region as Secure (for the purposes of software running on that processor).

6.2 Full IDAU

Arm's 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 defined in section Aliasing maps simply into bit[28] of the address being used to drive the Secure attribution signal.However, you need to consider NSC and also exempt regions for CoreSight ROM tables (and potentially similar).

6.2.1 Non-secure Callable regions

As noted in section Non-secure Callable (NSC) , Armv8-M requires any Secure address that is to be used as a branch target for switching between Non-secure to Secure state, must contain an SG instruction and be marked as NSC.

If an SAU is implemented, this could attempt to mark a region as NSC. However, if the IDAU marks the same region as Secure only, then this would be the most restrictive, and so prevent the region being treated as NSC. Similarly, if the IDAU marks a region as NSC, an implemented SAU could restrict this down 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 could contain Secure code as NSC. Alternatively, if an SAU is not implemented, or if additional flexibility is required, you can include additional controls to mark IDAU regions as Secure Non-secure Callable.

6.2.2 ROM table exemptions

Using Arm's CoreSight technology for debug requires the inclusion of system-level identification and ROM tables for the purpose of allowing an external debugger to automatically identify the devices it is connecting to. The 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.

You can use the exemption windows to provide access to an agent that authorizes Secure debug, see section Debug Security

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

6.3 Region numbers

Armv8-M processors implement a "TT" instruction designed to enable interrogation of the processor's internal memory protection units, the SAU and the IDAU. The full IDAU must provide a region ID number for any address query. The full IDAU must ensure that the region ID is different between addresses which either themselves have differing security attributes, or whose addresses span areas of non-contiguous security attribution.

The example system stripes Secure vs Non-secure at bit[28], thus the requirements can be met by assigning the region ID directly from address bits [31:28].

In this example system less than 16 region IDs are used. For Armv8-M implementations planned at the time of writing the region ID on the IDAU interface is defined as an 8-bit quantity limiting the maximum number of regions in the memory to 256. This means that the address map MUST NOT be fragmented into more than 256 transitions between secure and non-secure regions. This is a key advantage for the address map proposed in section Aliasing above as this strictly limits ID usage to only 16 no matter how fine grained memory/peripheral mapping is between secure and non-secure regions.

We strongly recommend that the proposal outlined in section Aliasing is followed for large parts of the address map. There can be exceptions to this but the granularity of secure and non-secure regions must be strictly limited in this case to avoid exceeding the maximum limit of 256 regions (see section Summary of recommendations).

6.4 Lite IDAU

TrustZone technology aware managers other than ARMv8-M processors, and any security wrappers, also require knowledge of the security address map. This is performed by a lite version of the IDAU, which provides Secure vs Non-secure attribution, but does not need to implement the NSC or region number aspects.

7. Security Wrapper

This section gives an overview of security wrappers including: Legacy managers and Programmable managers

7.1 Legacy managers

Transactions from legacy or non-security aware managers require appropriate attribution when connected into a TrustZone system. The example system uses security-wrapper gaskets to perform this function.

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

- If the request type is Secure, it uses the system address map (address bit[28] in the example system) to determine if the secure-attribute of the transaction should 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 subordinate components. However, it is permitted (though 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 would record an event in the system security controller, which in turn would generate a Secure interrupt to the ARMv8-M processor.

Broadly speaking, the mechanism 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. However, in practical systems we strongly recommend that ‘generating an abort’ or ‘ignoring-writes and returning a zero-on-a-read’ are selectable at run time under software control. The mechanism used to record an event, and generation of an interrupt is entirely within the implementer’s freedom (see section Summary of recommendations).

7.2 Programmable managers

Take care with programmable managers, for example DMAs, display controllers, USB and the like. If the manager is permitted to generate Secure transactions, then it must be ensured that this does not provide a mechanism for Non-secure software to control transactions in Secure memory space. For example, permit 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 this to ensure that DMA actions initiated by a Non-secure programmer always produce Non-secure DMA operations.

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

A non-security aware solution might be for the system to be configured by Secure software either to:

- Force all DMA manager transactions to Non-secure and permit Non-secure programming access, or
- Permit Secure DMA manager transactions but not permit programming from Non-secure.

In such a system, allowing Secure DMA transactions and Non-secure DMA programming would never be permitted. This requires co-ordination between the configuration of the security wrapper and the associated select-based gate for the subordinate side of the DMA.

8. Block-based Gate

This section gives an overview of Block-based gate.

8.1 Block assignment

Within the example system, a block-based gate protects and allocates regions of memory which are sub-divided into blocks. A typical example might be flash, where the use of a block-erase architecture results in a coarse-grained assignment to either Secure or Non-secure that mirrors the flash block size.

A control is implemented for each block which identifies whether the block is treated as Secure or Non-secure. At reset (or unless configured via alternative means, e.g. fuse), all blocks revert to Secure access only, and rely on Secure software to assign blocks as Non-secure.

In the example system, the control determines in which alias a particular block appears.

The gate has two requirements:

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

In addition to other less obvious attack vectors, requirement 2 prevents 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. This might be a sensible choice in practical systems as it will catch programming errors: The IDAU or SAU mapping of secure memory (which generates the secure attribute on an access) should match the block assignment so that a secure write should never occur to a non-secure block in a correctly configured system.

Within a pratical system we recommend a run time SW configurable rejection method used for non-permitted transactions to select between ‘ignore writes and to return zero on reads’ and ‘generating bus aborts’. The block based gate should also provide event information to the system security controller. This then generates an interrupt to the Armv8-M processor. Alternative mechanisms are also feasible, however, if an alternative rejected read default value is chosen, it must never be the “SG” instruction (`0xE97F`).

Although this mechanism is suitable for protecting the read interface of a flash controller, additional care must be taken with the programming interface, see section [Flash/NVM programming]

In practical systems we strongly recommend that fine-grained block based gates are used for on-chip SRAM. Ideally this would provide a 256-byte block granularity. This might prove to be impractical for larger on-chip SRAM sizes. As general guidance we recommend that 256-byte granularity is provided for SRAMs up to 128KBytes (total on-chip SRAM memory). For larger SRAM sizes granularity might scale with memory size. For example with 256KBytes of SRAM a granularity

of 512-bytes and for 512KBytes of SRAM a granularity of 1KBytes. In systems which provide multiple on-chip SRAM regions we recommend that a consistent block granularity is used across each on-chip SRAM region. (see section Summary of recommendations)

Larger memory types (particularly off-chip commodity memories) might use watermark based gates. Is may also make sense to remove aliasing for memory regions containing large off-chip memories.

The simple example in this guide uses aliasing, and so assigns a block to a particular security domain results in a “hole” in the other alias. 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.

9. Watermark-based Gate

This section gives an overview of Watermark-based gate.

9.1 Watermark assignment

In addition to being used for flash protection, the block based scheme can also be used to protect RAM. For completeness, the example system in this guide describes 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 will be 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) such that all of SRAM is marked as Secure; during initialization the Secure software sets the watermark to a value to provide the SRAM it does not require to the Non-Secure side.

Because SRAM can be used to hold code as well as 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, see section Block-based Gate.

Implementers can 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 is preferable to use a block based scheme for both flash and on-chip RAM in practical systems (see section Summary of recommendations).

10. Select-based Gate

This section gives an overview of Select-based gate.

10.1 Peripheral assignment

Typically, interconnects shall perform address decoding, and use this to select which peripheral a transaction is targeted to. This typically results in the generation of a “select” signal per peripheral. The example system implements a select-based gate that has a control per select line which identifies whether the peripheral is accessible to Non-secure code via the Non-secure address alias. It is recommended that an additional per-peripheral gate function be included to select privileged accesses only (see section 14.1 for a summary of recommendations).

When a Non-secure transaction targets a peripheral not assigned to Non-secure, the transaction shall be rejected. The rejection might be performed by the gate itself, or via the selection of a default-subordinate like component to handle the transaction.

From reset, the select-based gate is set to configure all peripherals as Secure; during initialization the Secure software can assign any peripherals it does not require to the Non-Secure side.

Where the select-based gate is used with peripherals, and it is never the case that the region could be fetched as an instruction by the Secure state of any processor, then it is not required that the peripheral be made un-readable by Secure transactions while assigned as a Non-secure peripheral. In general it might be preferable to reject Secure transactions while a peripheral is assigned as Non-secure. Where a transaction is rejected, the same mechanisms as used by the block-based gate are used. See section Block-based Gate

The requirement that a peripheral register cannot be fetched as an instruction by the processor is intended to prevent storing an “SG” instruction in a peripheral and subsequently branching to that location. You can meet this requiement by having the peripheral (or interconnect) block fetch transactions to the peripheral.

11. System Initialization Considerations

This section gives an overview of System Initialization considerations.

11.1 Processor reset

An Armv8-M processor with Security extension comes out of reset in Secure state at the address specified by the Secure Vector Table Offset Register (VTOR_S). ARM implementations provide a mechanism for specifying the default value of this register (via pins or configuration parameter). The implementer must ensure that this is configured to point to code at a Secure memory location; attempting to execute code from a Non-secure location while in Secure state would result in a fault condition within the processor.

11.2 System security controller default values

Usually, 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 world can subsequently be performed via Secure software during the boot process.

11.3 Debug security

Debug security needs to be factored in all stages of the product deployment. One methodology could be to produce devices where Secure debug is enabled from production; a subsequent programming phase is capable of disabling Secure debug, restricting the device so that an authorization mechanism (with optional erase) is required to subsequently enable the use of Secure debug.

12. Component-specific considerations

This section gives an overview of component specific considerations.

12.1 Introduction

This guide is not exhaustive. However, the following sections outline a few issues that implementers should also be aware of;

12.2 Flash or NVM programming

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

In particular, if you want to permit either Non-secure code or a Non-secure debugger to erase code blocks allocated to Non-secure code, an implementer must consider how they will permit this while simultaneously preventing Non-secure code/debug from erasing/modifying Secure blocks.

In some cases you might want to permit Non-secure flash updates; an implementer might choose to implement additional controls to enable/disable Non-secure programming.

12.3 Security Attribution Unit

This guide focuses on the use of TrustZone technology within the context of simple microcontrollers, making the assumption that the IDAU shall be used for configuring the security of each address. Additional models, particularly when deploying an Armv8-M processor within a larger application processor style system-on-chip, might benefit from the flexibility offered by using the internal SAU as the primary mechanism.

12.4 General-purpose IO

You might want to allocate certain IO pins to either Secure or Non-secure state. This could require replication of the GPIO interface, and in particular might require the addition of per-pin Secure vs Non-secure assignment bits, implemented so as to prevent Non-secure software from updating the Secure GPIO pins.

13. Recommendations for Practical Systems

This chapter gives the summary of recommendataions.

13.1 Summary of recommendations

In this section we summarise a set of recommendations for practical systems. Following these recommendations is strongly advised when a device will be accessible to a wide ecosystem of software developers. These recommendations will enable straightforward portability of secure software frameworks across these devices. In the case of ASICs which have a very targeted application with a very constrained developer base (for example, within a single OEM) then many of these recommendations might be relaxed to reduce the complexity of the system design:

The following basic level of system support is recommended:

- Configure the Armv8-M architecture processor to have 8 MPU Secure Regions, 8 MPU Non-secure regions, 8 SAU regions. It is possible to reduce SAU region requirement depending on system context.
- Broadly follow the memory map proposed in section Aliasing but when deviating from this it is vital to meet the restrictions described in section Region numbers.
- Use a block-based gate scheme for embedded NOR Flash and on-chip SRAM memories. Ideally with 256-byte block granularity for the SRAM. See section Block-based gate for more details.
- It is also ideal to provide a per-peripheral security selection and an additional per-peripheral selection allowing privileged accesses only. See section Peripheral assignment for more details.
- The mechanism used to reject non-permitted transactions should be ‘generating an abort’ or ‘ignoring-writes and returning a zero-on-a-read’ selectable at run time under software control. The system should record an event in the system security controller, which in turn should generate a Secure interrupt to the Armv8-M processor.

13.2 References

Here are some resources related to material in this guide:

- Armv8-M Architecture Reference Manual
- Cortex-M resources