



# Arm<sup>®</sup> Address Translation Unit

Version 1.0

## Specification

**Non-Confidential**

Copyright © 2023–2024 Arm Limited (or its affiliates). All rights reserved.

**Issue 02**

107714\_0000\_02\_en



# Arm® Address Translation Unit Specification

This document is Non-Confidential.

Copyright © 2023–2024 Arm Limited (or its affiliates). All rights reserved.

This document is protected by copyright and other intellectual property rights.

Arm only permits use of this document if you have reviewed and accepted [Arm's Proprietary Notice](#) found at the end of this document.

This document (107714\_0000\_02\_en) was issued on 2024-06-30. There might be a later issue at <http://developer.arm.com/documentation/107714>

The product version is 1.0.

See also: [Proprietary notice](#) | [Product and document information](#) | [Useful resources](#)

## Start reading

If you prefer, you can skip to [the start of the content](#).

## Intended audience

This book is written for system designers, system integrators, and programmers who are designing or programming a System-on-Chip (SoC) that uses an Arm® Address Translation Unit.

## 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](mailto:terms@arm.com).

## 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>.

# Contents

|  |           |
|--|-----------|
| <b>1. ATU overview.....</b>  | <b>5</b>  |
| 1.1 Topology.....  | 5         |
| 1.2 Terms used for the address-translation scheme.....                 | 6         |
| 1.3 Address translation scheme.....                                    | 7         |
| 1.3.1 Sample address-translation hardware flow.....                    | 9         |
| 1.3.2 Sample software address-translation setup examples.....          | 10        |
| 1.4 Uninitialized ATU.....   | 10        |
| <b>2. Functional description of the ATU.....</b>                       | <b>12</b> |
| 2.1 ATU register block.....  | 12        |
| 2.2 Address translator.....  | 12        |
| 2.3 Hardware interfaces.....   | 12        |
| 2.3.1 APB4 subordinate programming port interface.....                 | 13        |
| 2.3.2 AXI subordinate interface.....                                   | 13        |
| 2.3.3 AXI manager interface.....                                       | 13        |
| 2.3.4 Interrupt and alarm interface.....                               | 13        |
| <b>3. Configuration options for the ATU.....</b>                       | <b>14</b> |
| <b>4. Programmers model for the ATU.....</b>                           | <b>16</b> |
| 4.1 ATU register summary.....  | 17        |
| 4.1.1 ATUBC, Build Configuration register.....                         | 18        |
| 4.1.2 ATUC, Configuration register.....                                | 19        |
| 4.1.3 ATUIS, Interrupt Status register.....                            | 20        |
| 4.1.4 ATUIE, Interrupt Enable register.....                            | 21        |
| 4.1.5 ATUIC, Interrupt Clear register.....                             | 21        |
| 4.1.6 ATUMA, Mismatched Address register.....                          | 22        |
| 4.1.7 ATURSSLA<n>, Right Shifted Start Logical Address n register..... | 23        |
| 4.1.8 ATURSELA<n>, Right Shifted End Logical Address n register.....   | 24        |
| 4.1.9 ATURAV_L<n>, Region AddValue Low n the register.....             | 24        |
| 4.1.10 ATURAV_H<n>, Region AddValue High n register.....               | 25        |
| 4.1.11 ATUROBA<n>, Region Output Bus Attributes n register.....        | 26        |
| 4.1.12 ATURGPV<n>, Region General Purpose n register.....              | 29        |

|   |           |
|---|-----------|
| 4.1.13 PIDR4, Peripheral ID 4 register.....                         | 29        |
| 4.1.14 PIDR0, Peripheral ID 0 register.....                         | 30        |
| 4.1.15 PIDR1, Peripheral ID 1 register.....                         | 31        |
| 4.1.16 PIDR2, Peripheral ID 2 register.....                         | 32        |
| 4.1.17 PIDR3, Peripheral ID 3 register.....                         | 32        |
| 4.1.18 CIDR0, Component ID 0 register.....                          | 33        |
| 4.1.19 CIDR1, Component ID 1 register.....                          | 34        |
| 4.1.20 CIDR2, Component ID 2 register.....                          | 34        |
| 4.1.21 CIDR3, Component ID 3 register.....                          | 35        |
| <b>5. Typical software flows for the ATU.....</b>                   | <b>37</b> |
| 5.1 Enable a translation region.....                                | 37        |
| 5.2 Disable a translation region.....                               | 37        |
| 5.3 Dynamically remap an enabled translation region.....            | 38        |
| 5.4 Block access to disabled address ranges.....                    | 38        |
| 5.4.1 Software analysis flow when SAM resets the subsystem.....     | 39        |
| 5.4.2 Software analysis flow when SAM interrupts the subsystem..... | 40        |
| 5.4.3 Software analysis flow when ATU interrupts the subsystem..... | 42        |
| 5.4.4 Software analysis flow when ATU responds with bus error.....  | 42        |
| <b>Proprietary notice.....</b>                                      | <b>43</b> |
| <b>Product and document information.....</b>                        | <b>45</b> |
| Product status.....   | 45        |
| Revision history.....   | 45        |
| Conventions.....  | 46        |
| <b>Useful resources.....</b>  | <b>49</b> |

# 1. ATU overview

The Arm® *Address Translation Unit* (ATU) allows you to convert the *Logical Address* (LA) of an outgoing transaction that is initiated in the subsystem memory space to a *Physical Address* (PA) residing in the SoC system memory.

The ATU gives you the flexibility to allocate for or share the compute environment memory regions with the subsystem. These regions are in any address in the SoC system memory space, and decoupled from the LA mapping in the subsystem.

The ATU is informed of the security attributes of the incoming transaction from the subsystem computer environment at its subordinate AXI interface. The subsystem computer environment supports secure, non-secure, user, and admin privileges. The SoC system memory can support four physical address spaces: Secure, Non-secure, Root, and Realm. To address that gap, the ATU provides a way to translate LA to PA. It also targets system addresses with rich security attributes according to the translation region as defined in the output bus attributes, see [ATUROBA<n>](#), [Region Output Bus Attributes n register](#).

The ATU serves as a firewall. Any attempt to access a LA which has no matching configured address region in the ATU results in an alarm signal which can be routed to an alarm manager entity, such as the *Arm® Security Alarm Manager* (SAM). You can configure the alarm manager entity to react to this alarm from the ATU. Such reaction can be to reset the subsystem. Alternatively the alarm can set an IRQ for the processor of the subsystem.

For more information on the SAM, see the [Arm® Security Alarm Manager Specification](#).

## 1.1 Topology

This topics provides information about the ATU components and interfaces.

The components of the ATU are as follows:

### ATU register block

Configures the ATU and checks its status

### Address translator

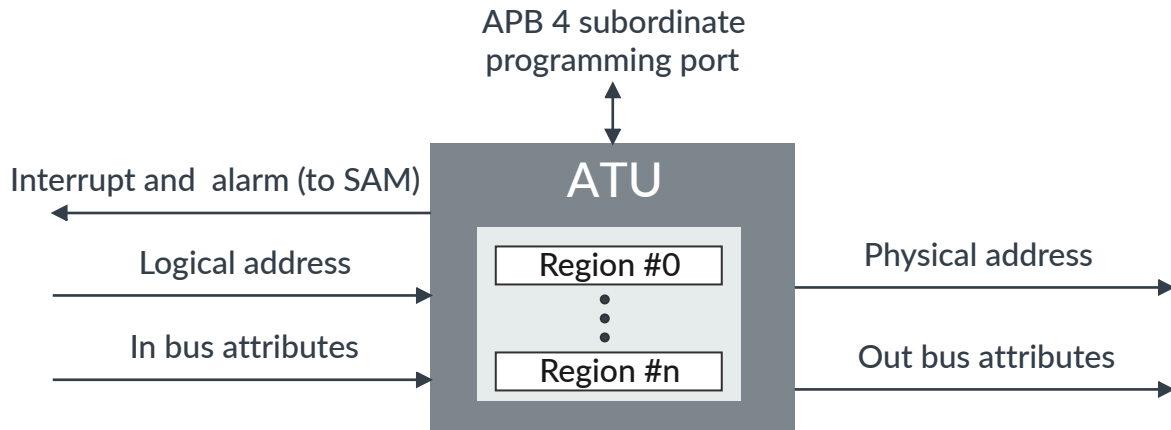
Scans the incoming logical address and attempts to find a match in all the enabled regions according to the defined algorithm

### Hardware interfaces

The ATU includes several hardware interfaces, such as the APB4 subordinate programming port interface, the AXI subordinate interface, the AXI manager interface, and the Interrupt and alarm interface

### ATU inputs and outputs

[Figure 1-1: ATU inputs and outputs](#) on page 6 shows the inputs and outputs connected to the operation of the ATU.

**Figure 1-1: ATU inputs and outputs**

The ATU in bus attributes input is the incoming security and caching attributes at the AXI subordinate interface. The ATU out bus attributes are programmable by the subsystem software for each address region according to the expected bus attributes of the target SoC.

The in bus and out bus attributes are as follows:

#### In bus attributes

AxProt[2:0], and AxCache[3:0] attributes.

#### Out bus attributes

AxProt[2:0], AxCache[3:0], and AxNSE attributes. The current ATU implementation assumes that the subsystem does not support AxNSE. However, the ATU supports AxNSE towards the external system.

The in bus attributes are evaluated by the matching address region in the AXI subordinate interface, and are forwarded or overridden as the out bus attributes at the AXI manager interface.

## 1.2 Terms used for the address-translation scheme

An overview of the relevant terms used in address-translation hardware flow.

The following terms are used for both the hardware address-translation flow and the software address-translation setup computation:

#### PS

The page size that the ATU supports. For example, for 4KB, PS for the following computation is 0xC.

#### LA

The subsystem's Logical Address (LA) which is input to the ATU. LA is a 32-bit address.

**PAW**

The SoC's Physical Address Width (PAW) configuration option.

**PA**

The system's Physical Address (PA) which is the output of the ATU. Its width is per ATUBC.PAW. PA width in bits =  $32 + 4 * \text{PAW}$ .

**CRSLA**

The *Computed Right Shifted (according to the ATU page size) Logical Address* (CRSLA). The address bits within the page are removed. The size of CRSLA is 32-PS bits.

**CRSPA**

The *Computed Right Shifted (according to the ATU page size) Physical Address* (CRSPA). The address bits within the page are removed. The size of CRSPA is  $(32 + 4 * \text{PAW})$ -PS bits.

**NR**

The number of regions that the ATU supports.

The following terms are only used for the software address-translation setup computation:

**RSLA**

The Right Shifted Logical Address.

**RSPA**

The Right Shifted Physical Address.

**ATURAV\_L**

Region Add Value Low n register.

**ATURAV\_H**

Region Add Value High n register.

## 1.3 Address translation scheme

The ATU compares the incoming 32-bit LA and the transaction size to the programmed regions setting. It produces the outgoing ATUPAW-bits wide PA using the address configuration of the matching region.

If the transaction LA and the transaction size do not match any of the programmed and enabled regions, the ATU blocks the transaction, responds with a bus error, and sets the ATUIS.ME register bit.

The outgoing PA size, ATUPAW, is a build-time configuration option.

The granularity of the ATU regions is by pages. ATU page size, ATUPS, is a build-time configuration option.

## Requirements for address translation

- An AXI burst must not pass the 4KB address boundary. ATU page sizes of less than 4KB are not supported. Otherwise, a single AXI burst can pass the ATU address boundary which would require a more complex ATU end-region check.
- The ATU validates that there is a match between the incoming address and an enabled region, <n>, if ((ATURSSLA<n>.RSSLA field <= CRSLA) and (CRSLA<= ATURSELA<n>.RSELA field)).

For more information about the ATURSSLA<n> register, see [4.1.7 ATURSSLA<n>, Right Shifted Start Logical Address n register](#) on page 23.

For more information about the ATURSELA<n> register, see [4.1.8 ATURSELA<n>, Right Shifted End Logical Address n register](#) on page 23.

- Software must make sure that there is no LA overlap in the enabled regions.
- ATURSELA<n>.RSELA field is the LA of the last page within the region.

Software can set two different LA regions to map into same PA region. For example, to allow access to system memory content as data from one LA region, then access this content as code from another logical address region.

## Process

The ATU finds the accessed region using the LA and the transaction size. Then it constructs the PA as follows:

1. The LA is right shifted by the value of ATUBC.PS bits, and the least significant address bits within a page are dropped. The result is the *Computed Right Shifted Logical Address* (CRSLA).
2. The CRSLA is checked to ensure that:
  - It is higher than or equal to ATURSSLA<n>.RSSLA bits
  - It is lower than or equal to ATURSELA<n>.RSELA bits in all enabled regions

If there is a match in only one enabled address region <n>:

- The CRSLA is added to the concatenation of ATURAV\_H<n>.AddValue\_M and ATURAV\_L<n>.AddValue\_L. This result is known as the adder result.
- The adder result is left shifted by the bit value of ATUBC.PS. Then it is ORed with the least significant address bits from the LA. The number of these bits are  $\log_2^{\text{page size}}$ . The result is the output PA.



For more information about the ATURAV\_H<n>.AddValue\_M, see [4.1.10 ATURAV\\_H<n>, Region AddValue High n register](#) on page 25.

For more information about the ATURAV\_L<n>.AddValue\_L, see [4.1.9 ATURAV\\_L<n>, Region AddValue Low n the register](#) on page 24.

If there is a mismatch or if there is more than one enabled address region <n>, the ATU:

- Blocks the access and responds to the originator with a bus error



- Sets the ATUIS.ME register bit
- Captures the mismatched LA in the ATUMA register

For more information about the ATUIS register, see [4.1.3 ATUIS, Interrupt Status register](#) on page 20.

### 1.3.1 Sample address-translation hardware flow

We provide an example for the sample hardware flow.

The snippet shows a sample hardware address-translation.

```

1. CRSLA = LA >> PS
2. MatchesFound = 0
3. On all regions (from n = 0 to NR-1)
   {
   The following operation is implemented in all regions that are parallel to
   one another, to complete the operation in minimal time.
   If (ATUC.n & (ATURSSLA<n>.RSSLA <= CRSLA) & (CRSLA <= ATURSELA<n>.RSELA)),
   then:

   a. MatchesFound += 1
   b. AddValue = (ATURAV_H<n>.AddValue_M<32> | ATURAV_L<n>.AddValue_L
      AddValue size is (32 + 4*PAW)-PS bits.
   c. PA = ((AddValue + CRSLA) <&& PS) + (LA & (2<sup>PS</sup> - 1))

   Notes:
   - AddValue is the right shifted (according to the ATU page size)
     value to add to the CRSLA. It is a (32 + 4*PAW)-PS bits value.
   - If the PA result is less than the LA then the AddValue is a
     negative number, that is its MS bits are FFs.
   - The adder of each region is of \[(32 + 4*PAW)-PS\] bits wide.
     It adds AddValue of \[(32 + 4*PAW)-PS\] bits to CRSLA of \[32-PS\] bits
   }

Notes:
- Software must assure that there is no overlap in the ATU regions,
  such that only one match will be found by the hardware.
- If an overlap of more than one region is found or considered as a software
  bug, then the ATU must block the transaction to the SoC and respond with
  a bus error.
Then it raise an interrupt bit in the ATUIS register.

4. If (MatchesFound == 1)
   a. Forward the transaction using the translated address.
   b. Set the out bus attributes according to the setup in the ATUROBA<n>
      register.

5. Else (Either MatchesFound == 0 or MatchesFound > 1)
   It is either software bug or an attack. Then, do the following:

   a. Block the access to system memory
   b. Respond with a bus error
   c. Set the ATUIS.ME field
   d. Latch the mismatched LA in the ATUMA.MA bit value

```

The coalesced interrupt output from the ATU can be connected to the SAM or similar alarm manager entity. When the ATUIS.ME and the ATUIE.ME bits are set, the coalesce interrupt output

is set, and the ATUERR alarm event can be registered by the alarm manager entity (for example SAM).

### 1.3.2 Sample software address-translation setup examples

We provide examples for for the sample software address-translation setup.

If the value of the PA is lower than the value of the LA, then the SW must compute a negative add value with top bits set to 1 such that the sum of add value and the RSLA results with the required PA value. The overflow top bits of the add operation are ignored.

#### Example 1 - Step-down address translation

```
PS = 0xC (4KB)
LA = 0x30000000
PAW = 6 (56 bits)
PA = 0x00000000_00D00000

SW computation:

RSLA = 0x30000
RSPA = 0x000_00000D00

Add value = RSPA - RSLA = 0x000_00000D00 - 0x30000 = 0xFFFF_FFFD0D00

ATURAV_L = 0xFFFFD0D00
ATURAV_H = 0x00000FFF
```

#### Example 2 - Step-up address translation (most significant supported bit of the PA (bit 55) is set)

```
PS = 0xC (4KB)
LA = 0x30000000
PAW = 6 (56 bits)
PA = 0x00800000_30D00000

SW computation:

RSLA = 0x30000
RSPA = 0x800_00030D00

Add value = RSPA - RSLA = 0x800_00030D00 - 0x30000 = 0x00000800_00000D00

ATURAV_L = 0x00000D00
ATURAV_H = 0x00000800
```

## 1.4 Uninitialized ATU

When the ATU is uninitialized, its regions are not enabled, and any attempt to access any system memory address fails.

This failure results in the following:

- A bus error occurs.
- The transaction to the SoC is blocked.
- The ATUIS.ME (Mismatch Error) bit is set.
- The mismatched Logical Address is latched in the ATUMA.

By default, the mismatch interrupt is enabled as it assumed that such an event is a security breach and therefore also conveyed to the SAM.

## 2. Functional description of the ATU

The ATU has several components that provide different functionality.

### 2.1 ATU register block

Software uses the ATU register block to configure the ATU and check its status.

For more information on the ATU registers and their properties, see the [Programmers model for the ATU](#).

### 2.2 Address translator

The ATU address translator operation starts when a read or write transaction is detected at its AMBA® AXI subordinate interface.

The address translator scans the incoming logical address and attempts to find a match in all the enabled regions, according to the algorithm that the [Address translation scheme](#) defines. If there is no match, the interrupt, ATUIRQ and alarm signal, ATUERR, are set.

### 2.3 Hardware interfaces

The ATU has several hardware interfaces, each with its own role.

The list of hardware interfaces are as follows:

#### **APB4 subordinate programming port interface**

The ATU includes an AMBA APB4 subordinate programming port interface that the subsystem software uses to control the ATU operation and read its status.

#### **AXI subordinate interface**

The ATU includes an AXI subordinate interface that the subsystem software or hardware uses to access the system address space for reads and writes.

#### **AXI manager interface**

The ATU includes an AXI manager interface used by the ATU to generate the outgoing AXI transaction to the system memory.

#### **Interrupt and alarm interface**

The ATU includes an interrupt and alarm interface.

### 2.3.1 APB4 subordinate programming port interface

The ATU includes an AMBA APB4 subordinate programming port interface that the subsystem software uses to control the ATU operation and read its status.

The APB4 subordinate programming port interface does not filter transactions according to their security attributes, Secure or Non-secure, or their privilege level attributes. To filter these transactions, we recommend controlling the ATU APB4 subordinate programming port interface using an Arm TrustZone® *Peripheral Protection Controller* (PPC). For more details about the APB4 TrustZone PPC, see the [Arm® CoreLink™ SIE-200 System IP for Embedded Technical Reference Manual](#).

For more information about the memory map and registers exposed in the APB4 subordinate programming port interface, see the [Programmers model for the ATU](#).

### 2.3.2 AXI subordinate interface

The ATU includes an AXI subordinate interface that the subsystem software or hardware uses to access the system address space for reads and writes.

The ATU uses the incoming address as its logical address input, and uses the incoming security and caching attributes as its in bus attributes input in the AXI subordinate interface.

### 2.3.3 AXI manager interface

The ATU includes an AXI manager interface that is used by the ATU to generate the outgoing AXI transaction to the system memory.

The ATU relates to the address at this interface as its Physical Address output and relates to the security and caching attributes at this interface as its Bus Attributes output. These addresses replace those coming from the AXI subordinate interface according to the programmed ATU translation scheme in the [ATUROBA<n>](#) register.

### 2.3.4 Interrupt and alarm interface

The ATU includes an interrupt and alarm interface.

The following table lists the interrupt and alarms used by the interface.

**Table 2-1: Interrupt and alarm interface signals**

| Signal name | Type      | Trigger                                   |
|-------------|-----------|---|
| ATUIRQ      | Interrupt | A non-matching incoming address           |
| ATUERR      | Alarm     | A non-matching incoming address           |
| ATUPARERR   | Alarm     | A parity error in the ATU register values |

### 3. Configuration options for the ATU

The ATU provides configuration options to configure its features and components.

The following table describes the configuration options of the ATU.

**Table 3-1: ATU configuration options**

| Configuration parameter name | Allowed values  | Default values | Description   |
|------------------------------|---|----------------|---|
| ATUNTR                       | 1 = 2 regions<br><br>2 = 4 regions<br><br>3 = 8 regions<br><br>4 = 16 regions<br><br>5 = 32 regions | NA             | Selects the number of translation regions that the ATU supports. This value is reflected in the ATUBC.NTR field.  |
| ATUPS                        | 0xC = 4096 bytes<br><br>0xD = 8192 bytes<br><br>0xE = 16384 bytes                                   | NA             | Selects the page size granularity that the ATU supports. This value is reflected in the ATUBC.PS field.<br><br><b>Note:</b><br>An AXI burst must not pass the 4KB address boundary. The ATU page sizes of less than 4KB are not supported, otherwise a single AXI transaction can pass the ATU page boundary which would require a more complex ATU end-region check. |

| Configuration parameter name | Allowed values   | Default values | Description  |
|------------------------------|--|----------------|--|
| ATUPAW                       | 0 = 32 bits<br><br>1 = 36 bits<br><br>2 = 40 bits<br><br>3 = 44 bits<br><br>4 = 48 bits<br><br>5 = 52 bits<br><br>6 = 56 bits<br><br>7 = 60 bits | NA             | Selects the SoC's physical address width. This value affects the adders of the ATU and ATURAV_H<n> registers, and is reflected in the ATUBC.PAW field. |

## 4. Programmers model for the ATU

This chapter provides general information about the ATU register properties.

The following information applies to ATU registers:

- The base address is not fixed and can be different for any particular system implementation. The offset of each register from the base address is fixed
- Do not attempt to access reserved or unused address locations. Attempting to access these locations can results in unexpected behavior
- Unless otherwise stated in the accompanying text:
  - Do not modify undefined register bits
  - Ignore undefined register bits on reads
  - All register bits are reset to the reset value specified in the register summary table of each block

Access type is described as follows:

**RW**

Read/write

**RO**

Read-only

**WO**

Write-only

**WI**

Write ignore

**RAZ/WI**

Read as zero, write ignore

**RAZW1C**

Read as zero, write 1 to clear the respective bit in the register



## 4.1 ATU register summary

The ATU register summary table lists the registers in the ATU register block, where n is 0 to (ATUBC.NTR - 1).

**Table 4-1: ATU register summary**

| Offset              | Name        | Access | Reset value | Width  | Description   |
|---------------------|-------------|--------|-------------|--------|---|
| 0x000               | ATUBC       | RO     | CFG_DEF     | 32-bit | ATU Build Configuration register. This register reflects the build configuration for software usage.<br><br>Its reset value depends on the selected configuration parameter in <a href="#">Configuration options for the ATU</a> <ul style="list-style-type: none"> <li>ATUNTR is reflected in the ATUBC.NTR</li> <li>ATUPS is reflected in the ATUBC.PS</li> <li>ATUPAW is reflected in the ATUBC.PAW</li> </ul> |
| 0x004               | ATUC        | RW     | 0x0000_0000 | 32-bit | ATU Configuration register. This register controls each ATU configuration.  |
| 0x008               | ATUIS       | RO     | 0x0000_0000 | 32-bit | ATU Interrupt Status register. This register indicates the current interrupt events.  |
| 0x00C               | ATUIE       | RW     | 0x0000_0001 | 32-bit | ATU Interrupt Enable register. This register is used to enable the corresponding interrupt events.  |
| 0x010               | ATUIC       | RW     | 0x0000_0000 | 32-bit | ATU Interrupt Clear register. This register is used to clear the corresponding interrupt events.  |
| 0x014               | ATUMA       | RO     | 0x0000_0000 | 32-bit | ATU Mismatched Address register. This register holds the last mismatched Logical Address which caused bus error and interrupt event.  |
| 0x018               | Reserved    | RAZ/WI | 0x0000_0000 | 32-bit | Reserved  |
| 0x01C               | Reserved    | RAZ/WI | 0x0000_0000 | 32-bit | Reserved  |
| 0x020               | ATURSSLA<n> | RW     | 0x0000_0000 | 32-bit | ATU Right Shifted Start Logical Address n register. This register defines the starting Logical Address for region n. The number of regions n depends on the ATUNTR configuration parameter.   |
| 0x0A0               | ATURSELA<n> | RW     | 0x0000_0000 | 32-bit | ATU Right Shift End Logical Address n register. This register defines the End Logical Address for region n. The number of regions n depends on the ATUNTR configuration parameter.  |
| 0x120               | ATURAV_L<n> | RW     | 0x0000_0000 | 32-bit | ATU Region Add Value Low n register. This register, along with the ATURAV_H register, defines the offset to be added to the incoming Logical Address, to compute the result, the Physical Address for region n. The number of regions n depends on the ATUNTR configuration parameter.  |
| 0x1A0               | ATURAV_H<n> | RW     | 0x0000_0000 | 32-bit | ATU Region Add Value High n register. This register, along with the ATURAV_L register, defines the offset to be added to the incoming Logical Address, to compute the result, the Physical Address for region n. The number of regions n depends on the ATUNTR configuration parameter.   |
| 0x220               | ATUROBA<n>  | RW     | 0x0000_8000 | 32-bit | ATU Region Output Bus Attributes n register. This register defines how the input bus attributes are transferred to the output bus attributes.   |
| 0x2A0               | ATURGPV<n>  | RW     | 0x0000_0000 | 32-bit | ATU Region General Purpose n register. This register serves the software and is not used by the ATU hardware.   |
| 0x320<br>–<br>0xFCC | Reserved    | RAZ/WI | 0x0000_0000 | 32-bit | Reserved  |
| 0xFD0               | PIDR4       | RO     | 0x0000_0004 | 32-bit | The Peripheral ID4 register returns byte[4] of the peripheral ID.   |

| Offset              | Name                  | Access | Reset value | Width  | Description  |
|---------------------|-----------------------|--------|-------------|--------|--|
| 0xFD4<br>–<br>0xFDC | Reserved              | RAZ/WI | 0x0000_0000 | 32-bit | Reserved   |
| 0xFE0               | <a href="#">PIDR0</a> | RO     | 0x0000_00C0 | 32-bit | The Peripheral ID0 register. Returns byte[0] of the peripheral ID. |
| 0xFE4               | <a href="#">PIDR1</a> | RO     | 0x0000_00B3 | 32-bit | The Peripheral ID1 register. Returns byte[1] of the peripheral ID. |
| 0xFE8               | <a href="#">PIDR2</a> | RO     | 0x0000_000B | 32-bit | The Peripheral ID2 register. Returns byte[2] of the peripheral ID. |
| 0xFEC               | <a href="#">PIDR3</a> | RO     | 0x0000_0000 | 32-bit | The Peripheral ID3 register. Returns byte[3] of the peripheral ID. |
| 0xFF0               | <a href="#">CIDR0</a> | RO     | 0x0000_000D | 32-bit | The Component ID0 register. Returns byte[0] of the component ID.   |
| 0xFF4               | <a href="#">CIDR1</a> | RO     | 0x0000_00F0 | 32-bit | The Component ID1 register. Returns byte[1] of the component ID.   |
| 0xFF8               | <a href="#">CIDR2</a> | RO     | 0x0000_0005 | 32-bit | The Component ID2 register. Returns byte[2] of the component ID.   |
| 0xFFC               | <a href="#">CIDR3</a> | RO     | 0x0000_00B1 | 32-bit | The Component ID3 register. Returns byte[3] of the component ID.   |

### 4.1.1 ATUBC, Build Configuration register

The ATUBC register reflects the ATU build configuration to the software.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

##### Address offset

0x000

##### Type

RO

##### Reset value

Configuration-dependent

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-2: ATUBC bit descriptions**

| Bits    | Name | Description | Type       | Reset |
|---------|------|-------------|------------|-------|
| [31:12] | -    | Reserved    | RAZ/<br>WI | 0x0   |

| Bits   | Name | Description  | Type       | Reset  |
|--------|------|--|------------|--------|
| [11:8] | PAW  | The physical address width of the SoC. This value affects the address of the ATU and the ATURAV_H<n> registers.<br><br>The ATUPAW configuration option defines the supported values in <a href="#">Configuration options for the ATU</a><br><br>Physical address width in bits = 32 + 4*PAW. | RO         | ATUPAW |
| [7:4]  | PS   | Page size. It is the region granularity and alignment in bytes.<br><br>The ATUPS configuration option defines the supported values in <a href="#">Configuration options for the ATU</a>  | RO         | ATUPS  |
| [3]    | -    | Reserved   | RAZ/<br>WI | 0x0    |
| [2:0]  | NTR  | Number of translation regions. Specifies the number of available regions. The ATUNTR configuration option defines the supported values in <a href="#">Configuration options for the ATU</a>  | RO         | ATUNTR |

### 4.1.2 ATUC, Configuration register

The ATUC register is used by software to control each ATU configuration.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

##### Address offset

0x004

##### Type

RW

##### Reset value

0x0000\_0000

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-3: ATUC bit descriptions**

| Bits   | Name | Description | Type       | Reset  |
|--------|------|-------------|------------|--------|
| [31:n] | -    | Reserved    | RAZ/<br>WI | 0x0000 |

| Bits      | Name | Description  | Type | Reset       |
|-----------|------|--|------|-------------|
| [(n-1):0] | RE   | <p>ATU regions enablement. The bit that is set enables its corresponding region, where n=the ATUBC.NTR. Then the number of implemented bits [(n-1):0] is according to the ATUBC.NTR field value:</p> <p><b>Bit 0</b><br/>Enable Region 0</p> <p><b>Bit 1</b><br/>Enable Region 1</p> <p><b>Bit n-1</b><br/>Enable Region n-1</p> | RW   | 0x0000_0000 |

### 4.1.3 ATUIS, Interrupt Status register

The ATUIS register indicates the status of the current interrupt events.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

##### Address offset

0x008

##### Type

RO

##### Reset value

0x0000\_0000

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-4: ATUIS bit descriptions**

| Bits   | Name | Description  | Type       | Reset |
|--------|------|--|------------|-------|
| [31:1] | -    | Reserved   | RAZ/<br>WI | 0x0   |
| [0]    | ME   | Mismatch Error. Indicates an address mismatch error. When set, the ATUMA register holds the offending Logical Address. | RO         | 0x0   |

### 4.1.4 ATUIE, Interrupt Enable register

The ATUIE register is used to enable corresponding interrupt events. When the interrupt enable field is set and the corresponding interrupt event in the status register is set, the coalesced interrupt output is asserted.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

##### Address offset

0x00C

##### Type

RW

##### Reset value

0x0000\_0001

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-5: ATUIE bit descriptions**

| Bits   | Name | Description                     | Type   | Reset |
|--------|------|---------------------------------|--------|-------|
| [31:1] | -    | Reserved                        | RAZ/WI | 0x0   |
| [0]    | ME   | Enable Mismatch Error interrupt | RW     | 0x1   |

### 4.1.5 ATUIC, Interrupt Clear register

The ATUIC register clears the corresponding interrupt events.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

**Address offset**

0x010

**Type**

RW

**Reset value**

0x0000\_0000

**Bit descriptions**

The following table shows the register bit assignments.

**Table 4-6: ATUIC bit descriptions**

| Bits   | Name | Description   | Type   | Reset |
|--------|------|---|--------|-------|
| [31:1] | -    | Reserved  | RAZ/WI | 0x0   |
| [0]    | ME   | When written as 1, clears the ME interrupt status in the ATUIS register | RAZW1C | 0x0   |

## 4.1.6 ATUMA, Mismatched Address register

The ATUMA register indicates the offending mismatched logical address that caused the ATU to reject the access to the system memory and caused the bus error or the interrupt.

**Configurations**

This register is available in all configurations.

**Attributes****Width**

32

**Functional group**[ATU register summary](#)**Address offset**

0x014

**Type**

RO

**Reset value**

0x0000\_0000

**Bit descriptions**

The following table shows the register bit assignments.

**Table 4-7: ATUMA bit descriptions**

| Bits   | Name | Description  | Type | Reset |
|--------|------|--|------|-------|
| [31:0] | MA   | Mismatch Address. Indicates the offending mismatched logical address that caused the mismatch error. | RO   | 0x0   |

### 4.1.7 ATURSSLA<n>, Right Shifted Start Logical Address n register

The ATURSSLA<n> register indicates the right shifted start logical address of region <n>. The access to addresses between the ATURSSLA<n> and the ATURSELA<n> registers is allowed by the ATU.

If all the enabled regions do not allow access, then the transaction is blocked.

#### Configurations

The number of actual supported translation regions (ATUNTR) is a configuration option.

#### Attributes

**Width**

32

**Functional group**

[ATU register summary](#)

**Address offset**

0x020

**Type**

RW. Unimplemented region registers are implemented as **RAZ/WI**.

**Reset value**

0x0000\_0000

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-8: ATURSSLA<n> bit descriptions**

| Bits           | Name  | Description  | Type          | Reset       |
|----------------|-------|--|---------------|-------------|
| [31:(31-PS+1)] | -     | Reserved   | <b>RAZ/WI</b> | 0x0         |
| [(31-PS):0]    | RSSLA | Right Shifted Start LA used for the corresponding region. Page Size (PS) refers to the ATUBC.PS bit.<br><br>The value stored in this field is the start LA, on the last page right shifted by the value of the PS. | RW            | 0x0000_0000 |

4.1.8 ATURSELA<n>, Right Shifted End Logical Address n register

The ATURSELA<n> register indicates the right shifted end logical address of region <n>. ATU allows access to the address between the ATURSSLA<n> and the ATURSELA<n> registers. If all of the enabled regions do not allow access, then the transaction is blocked.

Configurations

The number of actual supported translation regions (ATUNTR) is a configuration option.

Attributes

Width

32

Functional group

ATU register summary

Address offset

0x0A0

Type

RW. Unimplemented region registers are implemented as **RAZ/WI**

Reset value

0x0000\_0000

Bit descriptions

The following table shows the register bit assignments.

Table 4-9: ATURSELA<n> bit descriptions

| Bits           | Name  | Description  | Type   | Reset       |
|----------------|-------|--|--------|-------------|
| [31:(31-PS+1)] | -     | Reserved   | RAZ/WI | 0x0         |
| [31-PS:0]      | RSELA | Right Shifted End logical address used for the corresponding region. Page Size (PS) refers to the ATUBC.PS bit.<br><br>The value stored in this field is the end logical address within the last page, right shifted by the value of the PS. | RW     | 0x0000_0000 |

4.1.9 ATURAV\_L<n>, Region AddValue Low n the register

The ATURAV\_L<n> register indicates the least significant bits of AddValue of region <n>.

AddValue is the concatenation of ATURAV\_H<n> and ATURAV\_L<n> registers to a (32 + 4\*PAW)-PS bits value.

AddValue = (ATURAV\_H<n>.AddValue\_M<<32) | ATURAV\_L<n>.AddValue\_L



## Configurations

The number of actual supported translation regions (ATUNTR) is a configuration option.

## Attributes

### Width

32

### Functional group

[ATU register summary](#)

### Address offset

0x120

### Type

RW. Unimplemented region registers are implemented as **RAZ/WI**.

### Reset value

0x0000\_0000

## Bit descriptions

The following table shows the register bit assignments.

**Table 4-10: ATURAV\_L<n> bit descriptions**

| Bits   | Name       | Description  | Type          | Reset       |
|--|------------|--|---------------|-------------|
| <b>If (4*PAW – PS) &lt; 0</b><br>Reserved field consumes (PS-4*PAW) bits.<br><br><b>If (4*PAW – PS) &gt;= 0</b><br>Reserved field is of 0 bits (does not exist). | -          | Reserved   | <b>RAZ/WI</b> | 0x0         |
| <b>If (4*PAW – PS) &lt; 0</b><br>AddValue_L field consumes 32-(PS-4*PAW) bits.<br><br><b>If (4*PAW – PS) &gt;= 0</b><br>AddValue_L field consumes 32 bits.       | AddValue_L | LS bits of the AddValue used for the corresponding region. Page Size (PS) refers to the ATUBC.PS bit.<br><br>The value stored in this field is the LS bits of AddValue right shifted by the value of the PS. The MS bits are stored in the ATURAV_H<n> register. | RW            | 0x0000_0000 |

### 4.1.10 ATURAV\_H<n>, Region AddValue High n register

The ATURAV\_H<n> register indicates the most significant bits of AddValue of region <n>.

AddValue is the concatenation of ATURAV\_H<n> and ATURAV\_L<n> registers to a (32 + 4\*PAW)-PS bits value.

$$\text{AddValue} = (\text{ATURAV\_H< n >}. \text{AddValue\_M< < 32 >}) \mid \text{ATURAV\_L< n >}. \text{AddValue\_L}$$

## Configurations

The number of actual supported translation regions (ATUNTR) is a configuration option.

## Attributes

### Width

32

### Functional group

[ATU register summary](#)

### Address offset

0x1A0

### Type

RW. Unimplemented region registers are implemented as **RAZ/WI**.

### Reset value

0x0000\_0000

## Bit descriptions

The following table shows the register bit assignments.

**Table 4-11: ATURAV\_H<n> bit descriptions**

| Bits  | Name       | Description   | Type          | Reset       |
|---|------------|---|---------------|-------------|
| <b>If (4*PAW-PS) &lt; 0</b><br>Reserved field consumes 32 bits.<br><br><b>If (4*PAW-PS) &gt;= 0</b><br>Reserved field consumes 32-(4*PAW-PS) bits.                  | -          | Reserved  | <b>RAZ/WI</b> | 0x0         |
| <b>If (4*PAW-PS) &lt; 0</b><br>AddValue_M field consumes 0 bits (does not exist).<br><br><b>If (4*PAW-PS) &gt;= 0</b><br>AddValue_M field consumes (4*PAW-PS) bits. | AddValue_M | MS bits of the AddValue used for the corresponding region. Page Size (PS) refers to ATUBC.PS bit. Physical Address Width (PAW) refers to the ATUBC.PAW bit.<br><br>The value stored in this field is the MS 32 bits of AddValue right shifted by the value of the PS. | RW            | 0x0000_0000 |

### 4.1.11 ATUROBA<n>, Region Output Bus Attributes n register

The ATUROBA<n> register indicates the output bus attributes of region <n>.

The AxCACHE3.. AxCACHE0 must be programmed so that the resulting AxCACHE[3:0] values comply with the values provided in the [AMBA® AXI Protocol Specification](#). AxCACHE[3:0] must not be 0b0100, 0b0101, 0b1100, or 0b1101.

## Configurations

The number of actual supported translation regions is configurable using the ATUNTR configuration option.

## Attributes

### Width

32

### Functional group

[ATU register summary](#)

### Address offset

0x220

### Type

RW. Unimplemented region registers are implemented as **RAZ/WI**

### Reset value

0x0000\_8000

## Bit descriptions

The following table shows the register bit assignments.

**Table 4-12: ATUROBA<n> bit descriptions**

| Bits    | Name     | Description  | Type          | Reset |
|---------|----------|--|---------------|-------|
| [31:16] | -        | Reserved   | <b>RAZ/WI</b> | 0x0   |
| [15:14] | AxNSE    | <p>Affects the AxNSE signal at the main AXI Interconnect that goes to the SoC. Supported values are as follows:</p> <ul style="list-style-type: none"> <li>0b00 – Reserved</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxNSE of the output bus is set to 0</li> <li>0b11 – Set 1. AxNSE of the output bus is set to 1</li> </ul> <p>When the RME is supported in the SoC, the AxNSE signal is used with the AxPROT[1] signal to allow access to the Realm physical address space of the target SoC.</p> <p>For information about the AxNSE signal, see <a href="#">AMBA® AXI Protocol Specification</a>.</p> | RW            | 0b10  |
| [13:12] | AxCACHE3 | <p>Affects the AxCACHE[3] signal (Other Allocate) at the output bus which goes to the SoC. Supported values are as follows:</p> <ul style="list-style-type: none"> <li>0b00 – Passthrough. AxCACHE[3] of the input bus is transferred as is to the output bus</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxCACHE[3] of the output bus is set to 0</li> <li>0b11 – Set 1. AxCACHE[3] of the output bus is set to 1</li> </ul>  | RW            | 0b00  |

| Bits    | Name     | Description  | Type | Reset |
|---------|----------|--|------|-------|
| [11:10] | AxCACHE2 | Affects the AxCACHE[2] signal (Allocate) at the output bus which goes to the SoC. Supported values are as follows: <ul style="list-style-type: none"> <li>0b00 – Passthrough. AxCACHE[2] of the input bus is transferred as is to the output bus</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxCACHE[2] of the output bus is set to 0</li> <li>0b11 – Set 1. AxCACHE[2] of the output bus is set to 1</li> </ul>       | RW   | 0b00  |
| [9:8]   | AxCACHE1 | Affects the AxCACHE[1] signal (modifiable) at the output bus which goes to the SoC. Supported values are as follows: <ul style="list-style-type: none"> <li>0b00 – Passthrough. AxCACHE[1] of the input bus is transferred as is to the output bus</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxCACHE[1] of the output bus is set to 0</li> <li>0b11 – Set 1. AxCACHE[1] of the output bus is set to 1</li> </ul>     | RW   | 0b00  |
| [7:6]   | AxCACHE0 | Affects the AxCACHE[0] signal (Bufferable) at the output bus which goes to the SoC. Supported values are as follows: <ul style="list-style-type: none"> <li>0b00 – Passthrough. AxCACHE[0] of the input bus is transferred as is to the output bus</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxCACHE[0] of the output bus is set to 0</li> <li>0b11 – Set 1. AxCACHE[0] of the output bus is set to 1</li> </ul>     | RW   | 0b00  |
| [5:4]   | AxPROT2  | Affects the AxPROT[2] signal (Instruction access) at the output bus which goes to the SoC. Supported values are as follows: <ul style="list-style-type: none"> <li>0b00 – Passthrough. AxPROT[2] of the input bus is transferred as is to the output bus</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxPROT[2] of the output bus is set to 0</li> <li>0b11 – Set 1. AxPROT[2] of the output bus is set to 1</li> </ul> | RW   | 0b00  |
| [3:2]   | AxPROT1  | Affects the AxPROT[1] signal (Non-secure access) at the output bus which goes to the SoC. Supported values are as follows: <ul style="list-style-type: none"> <li>0b00 – Passthrough. AxPROT[1] of the input bus is transferred as is to the output bus</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxPROT[1] of the output bus is set to 0</li> <li>0b11 – Set 1. AxPROT[1] of the output bus is set to 1</li> </ul>  | RW   | 0b00  |
| [1:0]   | AxPROT0  | Affects the AxPROT[0] signal (Privileged access) at the output bus which goes to the SoC. Supported values are as follows: <ul style="list-style-type: none"> <li>0b00 – Passthrough. AxPROT[0] of the input bus is transferred as is to the output bus</li> <li>0b01 – Reserved</li> <li>0b10 – Set 0. AxPROT[0] of the output bus is set to 0</li> <li>0b11 – Set 1. AxPROT[0] of the output bus is set to 1</li> </ul>  | RW   | 0b00  |

### 4.1.12 ATURGPV<n>, Region General Purpose n register

The ATURGPV<n> register is intended for general-purpose use of the software by storing software related values for region <n>. The hardware does not use the values stored in this register.

The idea is to put one software flag per region or attribute in this register, which describes the region to the software. An example use case is to mark which entity owns the destined physical address of this region, for example, a system-provided region versus a subsystem owned and managed region. Software can use the flags as a security measure when the address translation API is called.

#### Configurations

The number of actual supported translation regions (ATUNTR) is a configuration option. For more information, see [Configuration options for the ATU](#).

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

##### Address offset

0x2A0

##### Type

RW. Unimplemented region registers are implemented as **RAZ/WI**.

##### Reset value

0x0000\_0000

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-13: ATUGPV<n> bit descriptions**

| Bits   | Name  | Description             | Type          | Reset |
|--------|-------|-------------------------|---------------|-------|
| [31:8] | -     | Reserved                | <b>RAZ/WI</b> | 0x0   |
| [7:0]  | Value | Value. Used by software | RW            | 0x00  |

### 4.1.13 PIDR4, Peripheral ID 4 register

The Peripheral ID4 register returns byte[4] of the peripheral ID.

#### Configurations

This register is available in all configurations.

## Attributes

### Width

32

### Functional group

[ATU register summary](#)

### Address offset

0xFD0

### Type

RO

### Reset value

0x0000\_0004

## Bit descriptions

The following table shows the register bit assignments.

**Table 4-14: Peripheral ID 4 bit descriptions**

| Bits   | Name  | Description   | Type | Reset    |
|--------|-------|---|------|----------|
| [31:8] | -     | Reserved  | -    | 0x000000 |
| [7:4]  | SIZE  | 4KB Count, the number of 4K pages used.<br><ul style="list-style-type: none"> <li>0x00: 4K</li> <li>0x01: 8K</li> <li>0x02: 16K</li> <li>0x03: 32K</li> </ul> | RO   | 0x0      |
| [3:0]  | DES_2 | JEP106 Continuation Code  | RO   | 0x4      |

### 4.1.14 PIDR0, Peripheral ID 0 register

The Peripheral ID0 register returns byte[0] of the peripheral ID.

## Configurations

This register is available in all configurations.

## Attributes

### Width

32

### Functional group

[ATU register summary](#)

### Address offset

0xFE0

**Type**

RO

**Reset value**

0x0000\_00C0

**Bit descriptions**

The following table shows the register bit assignments.

**Table 4-15: Peripheral ID 0 bit descriptions**

| Bits   | Name            | Description  | Type | Reset |
|--------|-----------------|--|------|-------|
| [31:8] | -               | Reserved   | -    | 0x00  |
| [7:0]  | Peripheral ID 0 | PART_0, Identification register part number, bits[7:0] | RO   | 0xC0  |

**4.1.15 PIDR1, Peripheral ID 1 register**

The Peripheral ID1 register returns byte[1] of the peripheral ID.

**Configurations**

This register is available in all configurations.

**Attributes****Width**

32

**Functional group**
[ATU register summary](#)
**Address offset**

0xFFE4

**Type**

RO

**Reset value**

0x0000\_00B3

**Bit descriptions**

The following table shows the register bit assignments.

**Table 4-16: Peripheral ID 1 bit descriptions**

| Bits   | Name   | Description                                     | Type | Reset    |
|--------|--------|---|------|----------|
| [31:8] | -      | Reserved  | -    | 0x000000 |
| [7:4]  | DES_0  | JEP106 identification code, bits[3:0]           | RO   | 0xB      |
| [3:0]  | PART_1 | Identification register part number, bits[11:8] | RO   | 0x3      |

### 4.1.16 PIDR2, Peripheral ID 2 register

The Peripheral ID2 register returns byte[2] of the peripheral ID.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

##### Address offset

0xFE8

##### Type

RO

##### Reset value

0x0000\_000B

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-17: Peripheral ID 2 bit descriptions**

| Bits   | Name     | Description                           | Type | Reset    |
|--------|----------|---------------------------------------|------|----------|
| [31:8] | _        | Reserved                              | _    | 0x000000 |
| [7:4]  | REVISION | Revision Code                         | RO   | 0x0      |
| [3]    | JEDEC    | JEDEC                                 | RO   | 0x1      |
| [2:0]  | DES_1    | JEP106 identification code, bits[6:4] | RO   | 0x3      |

### 4.1.17 PIDR3, Peripheral ID 3 register

The Peripheral ID3 register returns byte[3] of the peripheral ID.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32



Functional group

[ATU register summary](#)

Address offset

0xFEC

Type

RO

Reset value

0x0000\_0000

Bit descriptions

The following table shows the register bit assignments.

Table 4-18: Peripheral ID 3 bit descriptions

| Bits   | Name   | Description                  | Type | Reset    |
|--------|--------|------------------------------|------|----------|
| [31:8] | -      | Reserved                     | -    | 0x000000 |
| [7:4]  | REVAND | Manufacturer revision number | RO   | 0x0      |
| [3:0]  | CMOD   | Customer Modified            | RO   | 0x0      |

4.1.18 CIDR0, Component ID 0 register

The Component ID0 register returns byte[0] of the component ID.

Configurations

This register is available in all configurations.

Attributes

Width

32

Functional group

[ATU register summary](#)

Address offset

0xFF0

Type

RO

Reset value

0x0000\_000D

Bit descriptions

The following table shows the register bit assignments.

**Table 4-19: Component ID 0 bit descriptions**

| Bits   | Name    | Description | Type | Reset    |
|--------|---------|-------------|------|----------|
| [31:8] | -       | Reserved    | -    | 0x000000 |
| [7:0]  | PRMBL_0 | Preamble    | RO   | 0x0D     |

### 4.1.19 CIDR1, Component ID 1 register

The Component ID1 register returns byte[1] of the component ID.

#### Configurations

This register is available in all configurations.

#### Attributes

##### Width

32

##### Functional group

[ATU register summary](#)

##### Address offset

0xFF4

##### Type

RO

##### Reset value

0x0000\_00F0

#### Bit descriptions

The following table shows the register bit assignments.

**Table 4-20: Component ID 1 bit descriptions**

| Bits   | Name    | Description     | Type | Reset    |
|--------|---------|-----------------|------|----------|
| [31:8] | -       | Reserved        | -    | 0x000000 |
| [7:4]  | Class   | Component class | RO   | 0xF      |
| [3:0]  | PRMBL_1 | Preamble        | RO   | 0x0      |

### 4.1.20 CIDR2, Component ID 2 register

The Component ID2 register returns byte[2] of the component ID.

#### Configurations

This register is available in all configurations.

Attributes

Width

32

Functional group

[ATU register summary](#)

Address offset

0xFF8

Type

RO

Reset value

0x0000\_0005

Bit descriptions

The following table shows the register bit assignments.

Table 4-21: Component ID 2 bit descriptions

| Bits   | Name    | Description | Type | Reset    |
|--------|---------|-------------|------|----------|
| [31:8] | -       | Reserved    | -    | 0x000000 |
| [7:0]  | PRMBL_2 | Preamble    | RO   | 0x05     |

4.1.21 CIDR3, Component ID 3 register

The Component ID3 register returns byte[3] of the component ID.

Configurations

This register is available in all configurations.

Attributes

Width

32

Functional group

[ATU register summary](#)

Address offset

0xFFC

Type

RO

Reset value

0x0000\_00B1

## Bit descriptions

The following table shows the register bit assignments.

**Table 4-22: Component ID 3 bit descriptions**

| Bits   | Name    | Description | Type | Reset    |
|--------|---------|-------------|------|----------|
| [31:8] | -       | Reserved    | -    | 0x000000 |
| [7:0]  | PRMBL_3 | Preamble    | RO   | 0xB1     |

## 5. Typical software flows for the ATU

There are several typical software flows that occur under specific circumstances.

### 5.1 Enable a translation region

External system memories or devices can be accessed if a translation region is enabled.

Before the processor or a DMA device in the subsystem in which the ATU is installed attempts to access the external system memories or devices, its software must enable a translation region for these accesses. If the target address is already part of the enabled region, then such enablement is not required.

The flow for enablement of a translation region is as follows:

1. Select an unused and disabled region. For example, region <n>
2. Configure the selected region registers:
  - RSLA<n>
  - RSPA<n>
  - ATURAV\_L<n>
  - ATURAV\_H<n>
  - ATUROBA<n>
  - ATURGPV<n>
3. Set bit <n> in ATUC register to enable the region

### 5.2 Disable a translation region

When an ATU region is no longer used by the processor or a DMA device, the translation region should be disabled.

When the processor or a DMA device in the subsystem, in which the ATU is installed, stops using an ATU region, its software must disable the translation region for further accesses to:

- Free the region for other purposes
- Trap further mistaken accesses to the unused region as a security countermeasure

The flow for disabling translation regions:

1. Select the region to be disabled. For example, region <n>
2. Ensure that there are no outstanding transactions using region <n>
3. Clear bit <n> in ATUC register to disable the region

## 5.3 Dynamically remap an enabled translation region

You must either enable a translation region or to extend the already enabled region when accessing unmapped system memories or devices.

When the processor or a DMA device in the subsystem in which the ATU is installed must access partially-unmapped system memories or devices, its software must extend the enabled region to include the new system address range.

If the translation region is not yet enabled, see [Enable a translation region](#).

The flow for remapping an enabled translation region:

1. Select the active region to be changed. For example, region <n>
2. Ensure that there are no outstanding transactions using region <n>
3. Clear bit <n> in ATUC register to disable the region
4. Reconfigure the selected region registers:
  - RSLA<n>
  - RSPA<n>
  - ATURAV\_L<n>
  - ATURAV\_H<n>
  - ATUROBA<n>
  - ATURGPV<n>
5. Set bit <n> in ATUC register to enable the region

## 5.4 Block access to disabled address ranges

The ATU should block an address that is not enabled.

When the processor or a DMA device attempts to access a disabled address range, the ATU performs the following steps:

1. Blocks the access and returns a bus error. The response generated by the ATU is a SLVERR 0b10 error on a write response (BRESP) or read response (RRESP) signal.
2. Sets the ME bit in the ATUIS register, which if set generates an ATUIRQ interrupt to the processor or a DMA device.
3. Sets the ATUERR signal, which is typically connected to the SAM.
4. The ATUMA register captures the offending address to allow software to analyze the cause of the error.
5. If the SAM is installed and programmed to generate a reset response for the ATU reported error, then the subsystem is reset including the ATU.

6. Following the reset, the software can detect the cause of the reset by reading the RESET\_SYNDROME register of the subsystem, if available.

The RESET\_SYNDROME register indicates the cause for the reset. The SAMWRSTREQ or SAMCRSTREQ bit is set in the RESET\_SYNDROME register of the subsystem, but because the ATU was reset, the ATUMA loses its value. Therefore, software cannot analyze the exact cause of the error:

- If the error is sporadic, you can assume that the error is a response to an attack. The reset response is an effective countermeasure.
- However, if the error repeats, it is possible that a software error has occurred. In this case, it is important to know the exact offending address to fix the software bug.

### Using the ATU and SAM to analyze a software error

If the ATU alarm event is connected to the SAM input event 5 and a software error is being analyzed, there are two options:

- One option is that the software can replace the SAM reset response to one of the following interrupts:
  - *Non-Maskable Interrupt* (NMI), response 2
  - Critical Severity Fault Interrupt, response 3
  - Severity Fault Interrupt, response 4
- Another option is to disable the ATU input in the SAM and use the ATU interrupt. If the ATU input, the NMI, the Critical Severity Fault Interrupt, the Severity Fault Interrupt, and ATU interrupt are all disabled, the ATU still responds with a bus error to accesses to mismatch address. After one of these interrupts is serviced, software can read the ATUMA register to know the offending address.

For more information on the SAM, see the [Arm® Security Alarm Manager Specification](#).

#### 5.4.1 Software analysis flow when SAM resets the subsystem

After SAM resets the subsystem a specific analysis flow is used.

The following analysis flow is used after reset:

```

Read RESET_SYNDROME register
if (SAMWRSTREQ or SAMCRSTREQ bit is set)
{
    Read the SAMES0 register
    //Check and handle the ATUERR event:
    if (ATUERR bit (bit 5) in SAMES0 register is set)
    {
        //It is an ATU address mismatch event
        If of interest, log the ATU address mismatch event
        Clear the ATUERR bit (bit 5) in the SAMES0 register by writing 1
        to the ATUERR bit (bit 5) in the SAMECL0 register
    }
    else
    {
        Check and handle other SAM reported events.
    }
}

```

```

    Clear the AMWRSTREQ or SAMCRSTREQ bit in RESET_SYNDROME register
  }
  else - the subsystem was reset for other reasons (most likely reason is power on).
    Check the other RESET_SYNDROME register bits.

  Proceed with subsystem boot

```

## 5.4.2 Software analysis flow when SAM interrupts the subsystem

The SAM response to any input event, including ATUERR, can be programmed to generate an interrupt.

The generated interrupts are as follows:

- NMI
- Critical Severity Fault Interrupt
- Severity Fault Interrupt

Several issues can cause an NMI interrupt. The NMI exception handler must check for each possible cause. The ATU has its ATUIS register that indicates an address mismatch error.

### 5.4.2.1 NMI exception handler

The following analysis flow is used within the NMI exception handler:

```

Read the SAMES0 register
//Check and handle the ATUERR event:
if (ATUERR bit (bit 5) in SAMES0 register is set)
{
  //It is an ATU address mismatch event
  Read the ATUIS register
  if (ME bit is set)
  {
    Read the ATUMA register
    Log the offending mismatch address for analysis or debug
    Set the ME bit in the ATUIC register to clear the ME bit in the ATUIS
    register
  }
  Clear the ATUERR bit (bit 5) in the SAMES0 register by writing 1 to the ATUERR
  bit (bit 5) in the SAMECL0 register
}
else
{
  Check and handle other SAM reported events that could cause NMI, either in
  SAMES0 register or in SAMES1 register
}
Check for other reasons for NMI

// Consider resetting the subsystem
Exit from the NMI exception

```



### 5.4.2.2 Critical Severity Fault Interrupt exception handler

The following analysis flow is used within the Critical Severity Fault Interrupt exception handler:

```

Read the SAMES0 register
//Check and handle the ATUERR event:
if (ATUERR bit (bit 5) in SAMES0 register is set)
{
    //It is an ATU address mismatch event
    Read ATUIS register
    if (ME bit is set)
    {
        Read the ATUMA register
        Log the offending mismatch address for analysis or debug
        Set the ME bit in the ATUIC register to clear the ME bit in the ATUIS
        register
    }
    Clear the ATUERR bit (bit 5) in the SAMES0 register by writing 1 to the ATUERR
    bit (bit 5) in the SAMECL0 register
}
else
{
    Check and handle other SAM reported events that could cause a Critical Severity
    Fault Interrupt, either in the SAMES0 register
    or in the SAMES1 register
}

// Consider resetting the subsystem
Exit from the Critical Severity Fault exception

```

### 5.4.2.3 Severity Fault Interrupt exception handler

The following analysis flow is used within the Severity Fault Interrupt exception handler:

```

Read the SAMES0 register
//Check and handle the ATUERR event:
if (ATUERR bit (bit 5) in SAMES0 register is set)
{
    //It is ATU address mismatch event
    Read the ATUIS register
    if (ME bit is set)
    {
        Read the ATUMA register
        Log the offending mismatch address for analysis or debug
        Set the ME bit in the ATUIC register to clear the ME bit in the ATUIS
        register
    }
    Clear the ATUERR bit (bit 5) in the SAMES0 register by writing 1 to the ATUERR
    bit (bit 5) in the SAMECL0 register
}
else
{
    Check and handle other SAM reported events that could cause a Severity Fault
    Interrupt, either in the SAMES0 register or in
    the SAMES1 register
}

// Consider resetting the subsystem
Exit from the Severity Fault exception

```

### 5.4.3 Software analysis flow when ATU interrupts the subsystem

There is a specific software analysis flow for situations when ATU interrupts the subsystem by generating an ATUIRQ interrupt. This occurs when software programs SAM to mask out the ATUERR event, and enables the ATU mismatch error interrupt by setting the ME bit in the ATUIE register interrupt.

This analysis flow is used within the ATUIRQ exception handler.

The analysis flow is as follows:

```
Read the ATUIS register
if (ME bit is set)
{
    Read the ATUMA register
    Log the offending mismatch address for analysis or debug
    Set the ME bit in the ATUIC register to clear the ME bit in the ATUIS register
}

// Consider resetting the subsystem

Exit from the ATUIRQ exception
```

### 5.4.4 Software analysis flow when ATU responds with bus error

There is a specific software analysis flow for situations when ATU responds with a bus error. This analysis flow is used within the bus error exception handler.

The software analysis flow is as follows:

```
Read ATUIS register
if (ME bit is set)
{
    Read the ATUMA register
    Log the offending mismatch address for analysis or debug
    Set the ME bit in the ATUIC register to clear the ME bit in the ATUIS register
}
else
{
    Check and handle other sources that could cause bus error exception
}

// Consider resetting the subsystem

Exit from the bus error exception
```

# Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant

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

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

The validity, construction and performance of this notice shall be governed by English Law.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

# Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in Arm documents.

## Product status

All products and services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

## Revision history

These sections can help you understand how the document has changed over time.

### Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

#### Document history

| Issue   | Date         | Confidentiality  | Change          |
|---------|--------------|------------------|-----------------|
| 0000-02 | 30 June 2024 | Non-Confidential | Second release  |
| 0000-01 | 31 May 2023  | Non-Confidential | Initial release |

### Change history

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in [Document release information](#) on page 45.

**Table 2: Differences between issue 0000-01 and issue 0000-02**

| Change                               | Location  |
|--------------------------------------|---|
| Updated allowed bit value for ATUPAW | <a href="#">Configuration options for the ATU</a> |
| Updated bit values                   | <a href="#">ATURSSLA&lt;n&gt;</a>                 |
| Updated bit values                   | <a href="#">ATURSELA&lt;n&gt;</a>                 |

**Table 3: Issue 0000-01**

| Change                        | Location |
|-------------------------------|----------|
| Initial issue of the document | -        |

# 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](https://developer.arm.com/glossary).

## Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

| Convention                 | Use  |
|----------------------------|--|
| <i>italic</i>              | Citations.   |
| <b>bold</b>                | 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 <u>underline</u> | A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.  |
| <and>                      | Encloses replaceable terms for assembler syntax where they appear in code or code fragments.<br><br>For example:<br><div>MRC p15, 0, &lt;Rd&gt;, &lt;CRn&gt;, &lt;CRm&gt;, &lt;Opcode_2&gt;</div>              |
| SMALL CAPITALS             | Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, <b>IMPLEMENTATION DEFINED</b> , <b>IMPLEMENTATION SPECIFIC</b> , <b>UNKNOWN</b> , and <b>UNPREDICTABLE</b> . |



We recommend the following. If you do not follow these recommendations your system might not work.



Your system requires the following. If you do not follow these requirements your system will not work.



Danger

You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.



Note

This information is important and needs your attention.



Tip

This information might help you perform a task in an easier, better, or faster way.



Remember

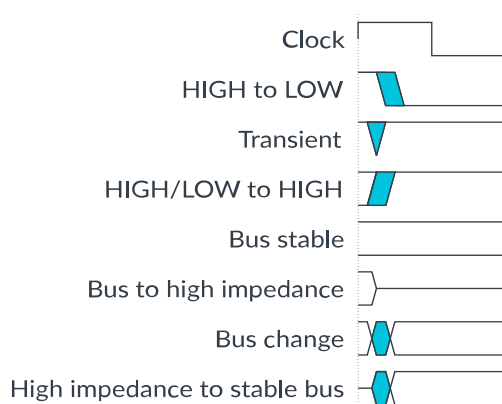
This information reminds you of something important relating to the current content.

## 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, n denotes an active-LOW signal.



## 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](https://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  |
|--|-------------|------------------|
| <a href="#">Arm® CoreLink™ SIE-200 System IP for Embedded Technical Reference Manual</a> | DDI 0571    | Non-Confidential |

| Arm architecture and specifications                       | Document ID | Confidentiality  |
|---|-------------|------------------|
| <a href="#">AMBA® AXI Protocol Specification</a>          | IHI 0022    | Non-Confidential |
| <a href="#">Arm® Security Alarm Manager Specification</a> | 107716      | Non-Confidential |