



# Arm® CoreLink™ DMA-350 Controller

Revision: r0p0

## Technical Reference Manual

**Non-Confidential**

**Issue 04**

Copyright © 2021–2023 Arm Limited (or its affiliates). 102482\_0000\_04\_en  
All rights reserved.



## Arm® CoreLink™ DMA-350 Controller Technical Reference Manual

Copyright © 2021–2023 Arm Limited (or its affiliates). All rights reserved.

### Release information

#### Document history

Issue	Date	Confidentiality	Change
0000-01	6 July 2021	Confidential	Initial release
0000-02	14 January 2022	Non-Confidential	EAC for r0p0
0000-03	15 December 2022	Non-Confidential	REL for r0p0
0000-04	18 January 2023	Non-Confidential	Second release of REL for r0p0

### Proprietary Notice

This document is protected by copyright and other related rights and the practice 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. 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 implementations infringe any third party patents.

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, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

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.

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

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

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. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2021–2023 Arm Limited (or its affiliates). All rights reserved.

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.

Previous issues of this document included language that can be offensive. We have replaced this language. See [B. Revisions](#) on page 194.

To report offensive language in this document, email [terms@arm.com](mailto:terms@arm.com).

# Contents

<b>1. Introduction.....</b>	<b>9</b>
1.1 Product revision status.....	9
1.2 Intended audience.....	9
1.3 Conventions.....	9
1.4 Useful resources.....	11
<b>2. Terms and abbreviations.....</b>	<b>12</b>
<b>3. DMA-350 overview.....</b>	<b>13</b>
3.1 Component overview.....	15
3.2 Compliance.....	16
3.3 Key features.....	17
3.4 Configurable options.....	17
3.5 Product documentation and design flow.....	18
3.5.1 Documentation.....	18
3.5.2 Design flow.....	18
3.6 Product revisions.....	19
<b>4. DMAC interfaces.....</b>	<b>20</b>
4.1 Clock and reset.....	20
4.2 APB4 subordinate interface.....	20
4.2.1 APB4 protection.....	21
4.2.2 APB4 PWAKEUP.....	21
4.2.3 APB4 PDEBUG.....	21
4.3 AXI5 manager interfaces.....	21
4.3.1 Burst type (FIXED / INCR only).....	22
4.3.2 ID generation.....	22
4.3.3 W transfer generation.....	22
4.3.4 Issuing capabilities.....	22
4.3.5 Response handling.....	23
4.3.6 AXI bandwidth utilization.....	23
4.3.7 Example of an unaligned start address.....	24
4.3.8 Example of an unaligned end address.....	25

4.3.9 Transfer attributes.....	27
4.3.10 Additional AXI signaling.....	27
4.3.11 Configurable secondary AXI5 port.....	27
4.4 Trigger Interface.....	28
4.4.1 Trigger input interface.....	28
4.4.2 Trigger input interface signals.....	29
4.4.3 Trigger output interface.....	29
4.4.4 Trigger output interface signals.....	29
4.5 AXI4 stream interface.....	29
4.6 LPI interfaces.....	30
4.6.1 LPI power P-Channel.....	30
4.6.2 LPI clock Q-Channel.....	30
4.7 DMA interrupts.....	30
4.8 Control and status interface.....	31
4.8.1 General purpose outputs.....	31
4.8.2 Stop and pause control.....	31
4.8.3 Cross Trigger Interface.....	32
4.8.4 Status signals.....	32
4.9 Configuration interface.....	33
4.9.1 Automatic boot interface.....	33
<b>5. DMAC operation.....</b>	<b>34</b>
5.1 DMAC operation overview.....	34
5.2 DMAC operation basic commands.....	34
5.2.1 1D operation modes.....	36
5.2.2 List of cases for 1D WRAP.....	37
5.2.3 1D transfers with increments.....	39
5.3 DMAC operation extended commands.....	41
5.3.1 Transfer type 2D.....	41
5.3.2 WRAP for 2D.....	45
5.3.3 Templated transfers.....	54
5.4 DMAC operation triggers.....	55
5.4.1 Trigger inputs.....	55
5.4.2 Trigger outputs.....	62
5.4.3 Trigger matrix for selectable sources.....	62
5.4.4 Internal trigger connection.....	63

5.4.5 Software triggers.....	64
5.5 AXI4 stream operation.....	65
5.5.1 Packet boundaries on stream in interface.....	65
5.5.2 Packet boundaries on stream out interface.....	66
5.5.3 Stream interworking with other modes.....	66
5.6 DMA Channel lifecycle.....	69
5.6.1 Command execution states.....	71
5.6.2 Automatic restart of commands.....	74
5.6.3 Error handling.....	74
5.6.4 Command execution status reporting.....	77
5.7 Command linking.....	81
5.7.1 Command structure.....	82
5.7.2 Loading commands.....	85
5.7.3 Automatic boot feature.....	86
5.8 Arbitration.....	87
5.8.1 Arbitration requests.....	87
5.8.2 Arbitration scheme.....	88
5.9 DMAC power management and DMAC control.....	90
5.9.1 Power management.....	90
5.9.2 Configuration.....	92
5.9.3 Halting and restarting the DMA with CTI.....	99
5.10 Initialization.....	99
5.11 Interrupt operation.....	100
<b>6. Programmers model.....</b>	<b>102</b>
6.1 About the programmers model.....	102
6.2 Programming considerations.....	102
6.3 Memory map.....	104
6.4 Register summary.....	105
6.4.1 DMACH<n> summary.....	105
6.4.2 DMANSECCTRL summary.....	106
6.4.3 DMASECCTRL summary.....	107
6.4.4 DMASECCFG summary.....	108
6.4.5 DMAINFO summary.....	108
6.5 Register descriptions.....	109
6.5.1 DMACH<n> description.....	109

6.5.2 DMANSECCTRL description.....	153
6.5.3 DMASECCTRL description.....	161
6.5.4 DMASEC CFG description.....	170
6.5.5 DMAINFO description.....	174
<b>A. Signal descriptions.....</b>	<b>186</b>
<b>B. Revisions.....</b>	<b>194</b>



# 1. Introduction

## 1.1 Product revision status

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

<b><math>r_x</math></b>	Identifies the major revision of the product, for example, $r1$ .
<b><math>p_y</math></b>	Identifies the minor revision or modification status of the product, for example, $p2$ .

## 1.2 Intended audience

This book is written for system designers, system integrators, and programmers who are designing or programming a System-on-Chip (SoC) that uses the CoreLink DMA-350.

## 1.3 Conventions

The following subsections describe conventions used in Arm documents.

### Glossary







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

See the Arm® Glossary for more information: [developer.arm.com/glossary](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.

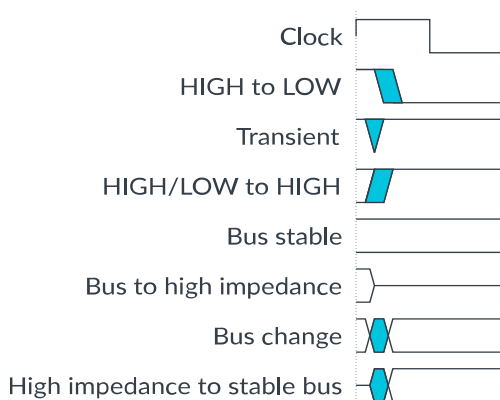
Convention	Use
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments.  For example:  <pre>MRC p15, 0, &lt;Rd&gt;, &lt;CRn&gt;, &lt;CRm&gt;, &lt;Opcode_2&gt;</pre>
<b>SMALL CAPITALS</b>	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, <b>IMPLEMENTATION DEFINED</b> , <b>IMPLEMENTATION SPECIFIC</b> , <b>UNKNOWN</b> , and <b>UNPREDICTABLE</b> .
 Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
 Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
 Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
 Note	An important piece of information that needs your attention.
 Tip	A useful tip that might make it easier, better or faster to perform a task.
 Remember	A reminder of something important that relates to the information you are reading.

## Timing diagrams

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

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

**Figure 1-1: Key to timing diagram conventions**



## Signals

The signal conventions are:

### Signal level

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

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

### Lowercase n

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

## 1.4 Useful resources

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

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at [developer.arm.com/documentation](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™ DMA-350 Controller Configuration and Integration Manual</a>	102483	Confidential
<a href="#">Arm® CoreLink™ DMA-350 Controller Release Note</a>	107750	Confidential

Arm architecture and specifications	Document ID	Confidentiality
<a href="#">AMBA® 5 AHB Protocol Specification</a>	IHI 0033	Non-Confidential
<a href="#">AMBA® APB Protocol Specification</a>	IHI 0024	Non-Confidential
<a href="#">AMBA® AXI and ACE Protocol Specification</a>	IHI 0022	Non-Confidential
<a href="#">AMBA® AXI-Stream Protocol Specification</a>	IHI 0051	Non-Confidential
<a href="#">AMBA® Low Power Interface Specification</a>	IHI 0068	Non-Confidential
<a href="#">Arm® Power Control System Architecture Specification</a>	DEN 0050	Confidential



Note

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

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

## 2. Terms and abbreviations

This document uses the following terms and abbreviations.

### **AHB**

Advanced High-performance Bus. A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol.

### **AMBA**

Advanced Microcontroller Bus Architecture

### **APB**

Advanced Peripheral Bus

### **AXI**

AMBA Advanced Extensible Interface

### **CTI**

Cross Trigger Interface

### **DMAC**

Direct Memory Access Controller

### **FSM**

Finite State Machine

### **LPI**

Low-power interface

### **RAZ/WI**

Read as Zero / Write Ignored

## 3. DMA-350 overview

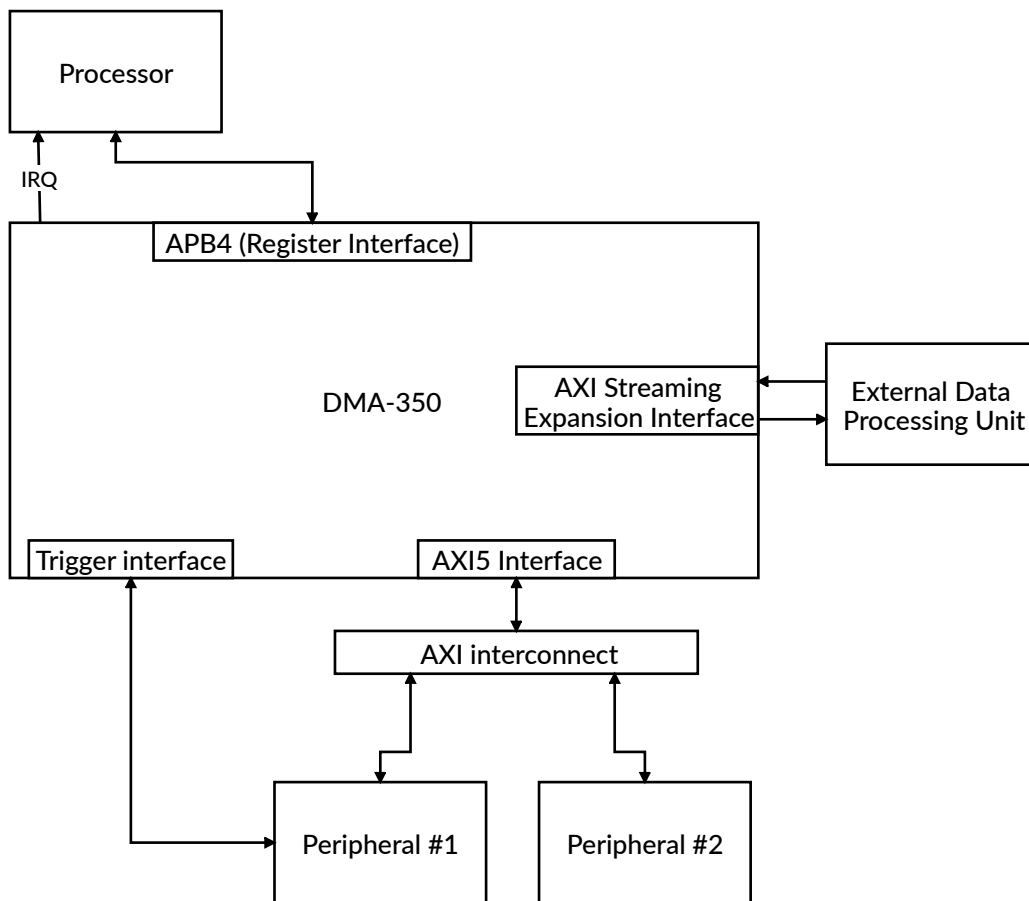
The CoreLink DMA-350 is a *Direct Memory Access Controller* (DMAC) with AMBA® AXI5 interfaces, which provides fast memory to memory, peripheral to memory, memory to peripheral copy, and peripheral to peripheral capabilities on multiple channels.

The processor in the system can control the DMA channel behavior over an APB4 interface and set security-related settings when Security Extension is supported. The DMAC has configurations for different types of copy, scatter-gather, increment, or two-dimensional image copy operations. It also supports trigger inputs and outputs for flow control and command sequencing capabilities. The DMA-350 adds support for low-power integration through the LPI interfaces for both clock and power.

The DMA-350 can have several channels that can have different properties, like FIFO size, to suit different requirements. Several DMA commands can be combined with the command linking feature. Using the command linking feature, a complex transfer can be set up and the DMAC can perform without the need for interaction with the processor. Each DMA channel can have a stream interface (AXI4-Stream) connected to an external Data Processing Unit. These channels can perform required data processing tasks on the fetched data before the DMAC writes the data to the destination.

The following figure shows DMA-350 in a system:

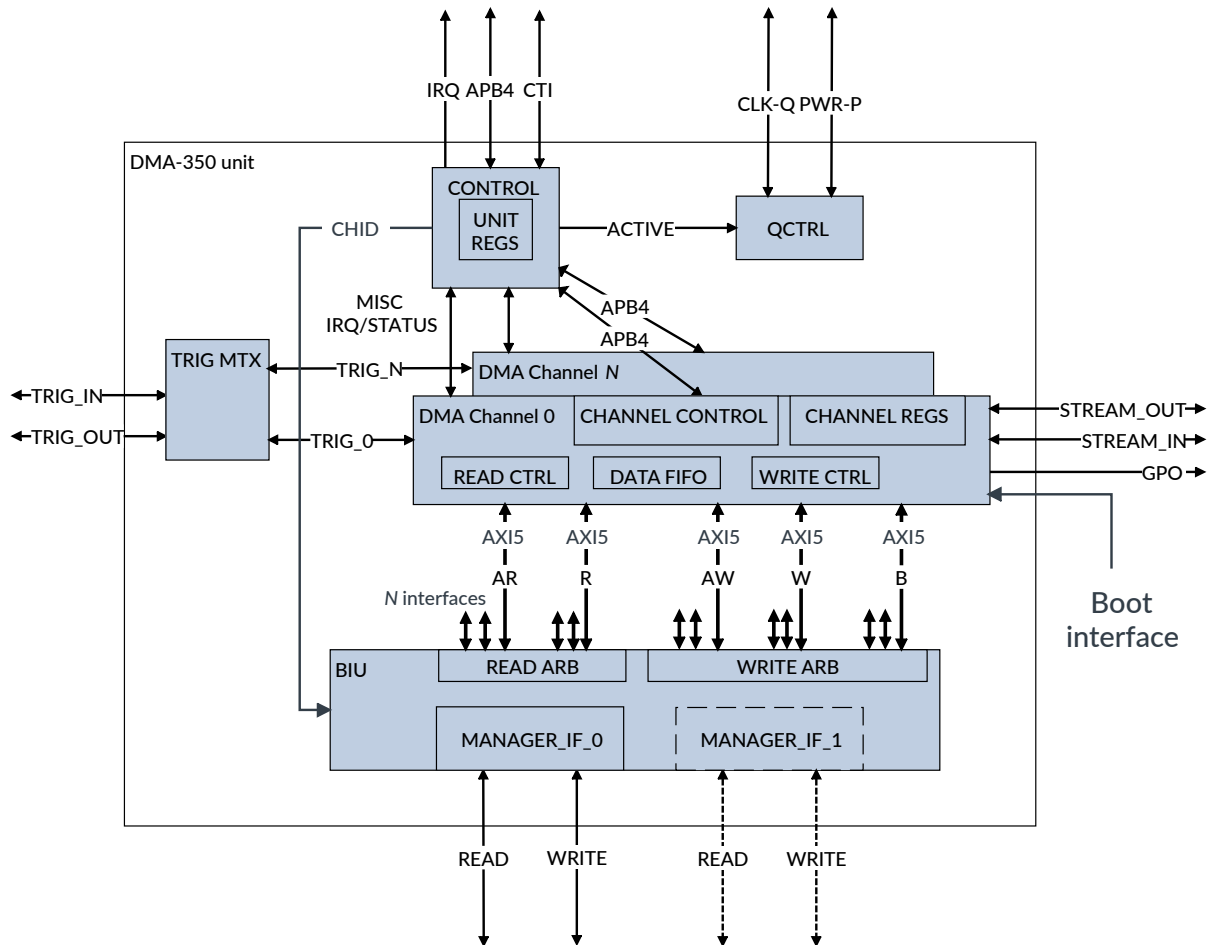
**Figure 3-1: DMA-350 in a system**



## 3.1 Component overview

The following block diagram shows the top-level components of DMA-350:

**Figure 3-2: DMA-350 top-level block diagram**



DMA-350 has the following main blocks:

### Control

#### Control Block

- Receives and terminates APB4 bus from the processor
- Forwards register accesses to each channel
- Implements the common control functions of the DMAC
- Handles security and privilege access right settings of the channels
- Generates interrupts for high-level DMAC events

## DMA Channel

DMA Channel block

- Implements one functional unit of the DMA controller that is capable of generating memory transfers
- Generates AXI transfers that match the settings of the current command
- Terminates the Trigger Interface from peripherals
- Generates interrupts and trigger out indications for internal channel events
- Implements an internal FIFO to separate read and write sides of the channel
- Implements support for an External Data Processing Unit over the AXI-4 stream interface
- Several DMA channels may exist in the DMA-350 determined by a configuration parameter

## BIU

Bus Interface Unit

- Responsible for routing and arbitrating traffic between the DMA channels and either one or two downstream AXI5 manager interfaces

## TRIG MTX

Trigger matrix block

- Implements a selectable connection between the trigger input and output ports of the DMA channels and the top-level trigger input and output ports
- Allows internal triggering between the channels

## QCTRL

Power and Clock management

- Implements the clock and power management features of DMA-350
- Terminates the LPI interfaces and allows quiescence based on the internal activity

## 3.2 Compliance

The DMA-350 interfaces are compliant with the following Arm specifications and protocols:

- AMBA AXI5 protocol, see the [AMBA® AXI and ACE Protocol Specification](#)
- AMBA APB4 protocol, see the [AMBA® APB Protocol Specification](#)
- AMBA AXI4-Stream protocol, see the [AMBA® AXI-Stream Protocol Specification](#)
- AMBA Low Power Interface protocol, see the [AMBA® Low Power Interface Specification](#)



## 3.3 Key features

The DMA-350 supports the following key features:

- Programmable APB4 configuration registers
- AXI5 data path
- *Low-power interface* (LPI) for clock and power
- 1-8 parallel DMA channels each with a configurable FIFO
- 1D memory copy including increments
- Interrupt capability for each channel and global events
- Flexible command linking capability
- Triggering capabilities for flow control
- Extended memory copy capabilities: 2D/wrap/template
- Expansion interface

## 3.4 Configurable options

The DMA-350 can be configured with the following options to meet specific design requirements:



The parentheses contain the relevant configuration parameter. These are described in the *Arm® CoreLink™ DMA-350 Controller Configuration and Integration Manual*.

- AXI5 address width 32-bit or 64-bit (ADDR\_WIDTH)
- AXI5 data width of 32-bit, 64-bit, or 128-bit (DATA\_WIDTH)
- Channel identification register width between 0-bit and 16-bit (CHID\_WIDTH)
- General Purpose Output (GPO) width of the channels between 0-bit and 32-bit (GPO\_WIDTH)
- GPO support of a channel (CH\_GPO\_MASK)
- Stream support of a channel (CH\_STREAM\_MASK)
- FIFO depth of every channel separately to 1,2,4,8,16,32,64 (CH\_<N>\_FIFO\_DEPTH)
- Extended feature support (2D, wrapping, template) of a channel (CH\_EXT\_FEAT\_MASK)
- Number of DMA channels between 1 and 8 (NUM\_CHANNELS)
- Number of trigger input ports between 0 and 32 (NUM\_TRIGGER\_IN)
- Number of trigger output ports between 0 and 32 (NUM\_TRIGGER\_OUT)
- Trigger input port synchronization (TRIG\_IN\_SYNC\_EN\_MASK)
- Trigger output port synchronization (TRIG\_OUT\_SYNC\_EN\_MASK)

- Presence of secondary AXI5 manager port (AXI5\_M1\_PRESENT)
- Presence of Security Extension for TrustZone® support (SECEXT\_PRESENT)

## 3.5 Product documentation and design flow

This section describes the DMA-350 documentation in relation to the design flow.

### 3.5.1 Documentation

The DMA-350 documentation is as follows:

#### Technical Reference Manual

The Technical Reference Manual (TRM) describes the functionality and the effects of functional options on the behavior of the DMA-350. It is required at all stages of the design flow. The choices made in the design flow can mean that some behaviors described in the TRM are not relevant.

If you are programming the DMA-350, contact:

- The implementer to determine:
  - The build configuration of the implementation.
  - The integration, if any, that was performed before implementing the DMA-350.
- The integrator to determine the pin configuration of the device that you are using.

#### Configuration and Integration Manual

The Configuration and Integration Manual (CIM) describes:

- The available build configuration options and related issues in selecting them.
- How to integrate the DMA-350 into an SoC. This section describes the considerations need to be taken into account by the integrator when connecting the interfaces in the system.
- The processes to sign off on the configuration, integration, and implementation of the design.

The CIM is a confidential document that is only available to licensees.

### 3.5.2 Design flow

The DMA-350 is delivered as synthesizable RTL. Before it can be used in a product, it must go through the following processes:

#### Implementation

The implementer configures and synthesizes the RTL to produce a soft netlist.

#### Integration

The integrator connects the implemented design into an SoC. Integration includes connecting the design to a memory system and peripherals.

## Programming

The system programmer develops the software to configure and initialize the DMA-350, and tests the required application software.

Each process is separate and can include implementation and integration choices that affect the behavior and features of the DMA-350.

The operation of the final device depends on:

### Build configuration

The implementer chooses the options that affect how the RTL source files are pre-processed.

These options usually include or exclude logic that affects one or more of the following:

- Area
- Maximum frequency
- Features of the resulting macrocell

### Configuration inputs

The integrator configures some features of the DMA-350 by tying inputs to specific values. These configurations affect the start-up behavior before any software configuration is made.

### Software configuration

The programmer configures the DMA-350 by programming particular values into registers. This configuration affects the behavior of the DMA-350.

## 3.6 Product revisions

This section describes the differences in functionality between product revisions:

r0p0 First release.

## 4. DMAC interfaces

This section contains an overview of the CoreLink DMA-350 interfaces.

### 4.1 Clock and reset

The DMA-350 is intended to be placed in a single clock and reset domain.

The APB4 configuration interface and the AXI5 manager interfaces (M0 and M1) use a clock enable signal to support these interfaces to run on a divided frequency. The APB configuration interface uses the `pclken` signal, the M0 interface uses the `aclken_m0` signal, and the M1 interface uses the `aclken_m1` signal.



The optional AXI4-Stream manager and subordinate interfaces have no clock enable signal, they use the module clock directly.

---

The DMA-350 uses a single, active-LOW reset, `resetrn`. This reset must be deasserted synchronously with the `clk` clock signal, but it can be asserted asynchronously.

There are no special powerup requirements defined for DMA-350.

### 4.2 APB4 subordinate interface

The APB4 subordinate interface is for accessing the internal configuration registers of the DMA-350.

The DMA-350 supports the AMBA APB4 Protocol Specification. For more information, see the [AMBA® APB Protocol Specification](#) document.

#### Register access

Only 32-bit wide accesses are supported. When an APB4 Write-Access uses `pstrb` other than 4'b1 it results in an error response. Unaligned accesses are treated as normal accesses by having `paddr[1:0]` tied to 0 internally.

## 4.2.1 APB4 protection

Certain registers in the APB4 register space are protected for higher privilege and security software modules. These registers configure the overall behavior of the DMAC and are protected internally within the DMAC by checking the privilege and security attribute of the access.

### Privilege protection

The DMAC checks pprot[0] bit when accessing registers with higher privilege rights. The DMAC returns with **RAZ/WI** when the access is denied.

### Security protection

The DMAC supports the Security Extension by checking pprot[1] when accessing registers with higher security rights. When accesses are violating security the DMAC returns **RAZ/WI** or error response depending on the SW configurable register and it might also generate an interrupt if enabled.

## 4.2.2 APB4 PWAKEUP

The APB4 interface is extended with a clock wakeup signaling through the pwakeup port. This allows faster clock request mechanism when the pwakeup is generated from a registered source.

This signal is used to indicate that there is ongoing activity that is associated with the APB4 interface. If the signal is not part of the APB4 interface, the psel signal can be registered to generate the pwakeup signal.

## 4.2.3 APB4 PDEBUG

The APB4 interface has a sideband signal that allows a debugger to access the DMAC without causing side-effects when sweeping the memory range in Non-secure mode. When the debugger creates the APB4 access this signal can be set HIGH, which disables the security violation error and interrupt generation while still protecting the Secure registers by responding with **RAZ/WI**. This signal must be stable throughout the 2 cycles of the access.

## 4.3 AXI5 manager interfaces

The DMAC is an AXI5 manager that complies with AMBA AXI5 protocol with a reduced set of AXI5 features. For details on the protocol, see [AMBA® AXI and ACE Protocol Specification](#).

**Table 4-1: Supported AXI5 features**

AXI5 feature	DMA-350 support	Comment
Poison	Yes	Only on R-channel to raise an error condition
Wakeup_signals	Yes	Wakeup signal present on both manager interfaces
Max_Transaction_Bytes	Yes	1024 bytes is the maximum in one burst
Regular_Transactions_Only	No	Burst sizes can take other than power of 2 values

AXI5 feature	DMA-350 support	Comment
Exclusive_Accesses	No	No exclusive transfers sent from the DMAC

AXI5 manager interfaces are used for the read and write data transfers generated by the DMA channels.

A single AXI5 manager interface is always present on the DMAC. An extra AXI5 interface can also be added by using configuration parameter *AXI5\_M1\_PRESENT*. For a detailed description on parameters, see [Configuration](#).

### 4.3.1 Burst type (FIXED / INCR only)

The DMA-350 reduces complexity by only sending FIXED and INCR bursts.

FIXED bursts are used when targeting peripherals with FIFOs and the same address is accessed several times. This can be done by setting the increment value to 0.

INCR bursts are used in all other cases and the length of a burst is limited by the \*MAXBURSTLEN register setting. This also reduces the complexity of converting AXI5 transfers to AHB in case the DMAC is connected to an AHB interconnect at some point.

### 4.3.2 ID generation

The AxID generation is automatically done by the DMAC on channel basis. Read and write transfers use an AxID that identifies the channel. The command linking also uses the channel's own ID for fetching the commands from memory but with an extra sideband signal indicating the command access. SW-based channel ID setting can be added by using the CHID registers to aid memory mapping outside the DMAC. These registers are present on the bus as User signals, the DMAC does not use them internally.

### 4.3.3 W transfer generation

The WDATA is sent on the bus at the same time or after the AW channel transfer. The WDATA does not overtake the AW control information as it adds complexity to the arbitration logic.

Strobe signals are used if the transfer size is set to less than the bus data width.

### 4.3.4 Issuing capabilities

Read issuing capability is limited by the FIFO depth of every channel separately to 1, 2, 4, 8, 16, 32, 64. The channels issue reads when they have enough space to store the complete read response.

Write issuing capability depends on the transfer size and the data stored in the internal FIFO of the channel. A write is issued when the FIFO contains the full burst with all its beats.

The number of transfers issued depends on the command type and the address alignment.

The maximum issuing capability is limited by the DMA unit to avoid overloading the AXI infrastructure with too many outstanding transfers. The software can control the number of transfers outstanding on the bus. This enables debug and bus utilization shaping if the bus is still overloaded.

### 4.3.5 Response handling

AXI accesses can result in success or different error responses. Error responses are also routed to the particular channel and it goes into error state, while the others continue their operation.

### 4.3.6 AXI bandwidth utilization

This section provides a definition of terms used and optimization requirements.

#### Definition of terms used

##### Burst breakpoint

Boundary that AXI5 bursts cannot cross.

These boundaries can be the following:

- 1KB address boundary: DMA-350 never crosses 1KB address boundaries (AHB requirement support).
- Trigger block end: Flow trigger requests enable a trigger block to be accessed. Once the access is complete, a new trigger request is required to access any further data so AXI5 bursts cannot go beyond trigger blocks.
- FIFO size/2 limits the amount of data transferred by a burst.
- MAXBURST register setting limits the burst length (axlen): the burst must be shorter than this user-specified value

##### Unaligned start

First accessed byte of an AXI5 transaction is not the LSB byte lane of the bus.

##### Unaligned end

Last accessed byte of an AXI5 transaction is not the MSB byte lane of the bus.

DMA-350 has two modes to determine the transfer size of its AXI5 burst beats (axsize):

##### Unoptimized

Beat size is the same as set in DMA command regardless of bus width.

##### Optimized

To optimize bus utilization for narrow transfers, they are grouped into wider accesses, when possible. The AXI5 protocol specification offers different possibilities for read and write side.

- For write side, beat size is the same as bus width, write strobes are used to exclude unaccessed byte lanes from write.

- For read side, beat size is the same as bus width for an AXI5 transaction with aligned end. AXI5 transaction with unaligned end may only be partially optimized. See the examples in [Example of an unaligned start address](#) and [Example of an unaligned end address](#).

## Optimization requirements

- AXI5 subordinate targeted by AXI5 bursts must be of normal memory type (not device).
- Template feature must be disabled for current DMA command on current side.
- Address increment must be 1 for current DMA command on current side.
- Read:
  - AXI transaction either contains enough data to reach bus alignment or has an aligned start.
  - If start is aligned, but not enough data to access all bytes on the bus, largest possible arsize is used that stays within the address space defined by the DMA command.

### 4.3.7 Example of an unaligned start address

Assuming the following settings:

- DATA\_WIDTH = 64
- CH\_CTRL.TRANSIZE = 0 (8-bit), CH\_XADDRINC.SRC/DESXADDRINC = 1
- CH\_XSIZE.SRC/DESXSIZE = 0xD (13 bytes)
- CH\_SRC/DESADDR.SRC/DESXADDR[2:0] = 3

#### Unoptimized read/write

AXI5 transaction uses the TRANSIZE value for axsize. This means the unoptimized bandwidth utilization is constant for all beats.

#### Utilization

Transaction size in bytes/bus size in bytes, 1/8 in this example.

The following figure shows the AXI5 transactions for this scenario. Each row in the figure represents a transfer and the shaded cells indicate bytes that are not transferred.



**Figure 4-1: AXI5 transactions for unoptimized bandwidth utilization**

AXI transaction parameters				Data byte lanes							
Address	Transfer size	Burst length	Transfer number	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
0x03	8-bits	13 transfers									
			1st	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			2nd	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			3rd	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			4th	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			5th	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			6th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08
			7th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08
			8th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08
			9th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08
			10th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08
			11th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08
			12th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08
			13th	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08

## Optimized read/write

When bandwidth utilization is optimized, read and write side behave similarly in case the start address is unaligned. The axaddr signal uses the unaligned address. For reads it implicitly means for the AXI5 subordinate to ignore lower byte lanes. For writes, additionally the wstrb signals specify the used byte lanes.

### Utilization

Bytes until first aligned address in AXI-access/bus size in bytes, (8-3)/8 in this example.

The following figure shows the AXI5 transactions for this scenario. Each row in the figure represents a transfer and the shaded cells indicate bytes that are not transferred.

**Figure 4-2: AXI5 transactions for optimized bandwidth utilization**

AXI transaction parameters				Data byte lanes							
Address	Transfer size	Burst length	Transfer number	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
0x03	64-bits	2 transfers									
			1st	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			2nd	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08

## 4.3.8 Example of an unaligned end address

Assume the following settings:

- DATA\_WIDTH = 64
- CH\_CTRL.TRANSIZE = 0 (8-bit), CH\_XADDRINC.SRC/DESXADDRINC = 1
- CH\_XSIZE.SRC/DESXSIZE = 0x3 (3 bytes)

- CH\_SRC/DESADDR.SRC/DESXADDR[2:0] = 0

### Unoptimized read/write

AXI5 transaction uses the TRANSIZE value for axsize. This means the unoptimized bandwidth utilization is constant for all beats.

#### Utilization

Transaction size in bytes / bus size in bytes, 1/8 in this example.

The following figure shows the AXI5 transactions for this scenario. Each row in the figure represents a transfer and the shaded cells indicate bytes that are not transferred.

**Figure 4-3: AXI transactions for unoptimized read/write**

AXI transaction parameters				Data byte lanes							
Address	Transfer size	Burst length	Transfer number	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
0x00	8-bits	3 transfers									
			1st	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			2nd	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
			3rd	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00

### Optimized read

The first burst uses 16-bit access, while the second only 8-bit.

#### Utilization

Transaction size in bytes/bus size in bytes,

- 2/8 – for the first transaction
- 1/8 – for the second transaction.

The following figure shows the AXI5 transactions for this scenario. Each row in the figure represents a transfer and the shaded cells indicate bytes that are not transferred.

**Figure 4-4: AXI transactions for optimized read**

AXI transaction parameters				Data byte lanes							
Address	Transfer size	Burst length	Transfer number	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
0x00	16-bits	1 transfer									
			1st	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
0x02	8-bits	1 transfer									
			1st	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00

### Optimized write

For the write bursts the bus width can be used, wstrb signals specify the used byte lanes for the AXI5 subordinate.

## Utilization

Transaction size in bytes/bus size in bytes, 3/8 for this transaction.

The following figure shows the AXI5 transactions for this scenario. Each row in the figure represents a transfer and the shaded cells indicate bytes that are not transferred.

**Figure 4-5: AXI transactions for optimized write**

AXI transaction parameters				Data byte lanes							
Address	Transfer size	Burst length	Transfer number	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
0x00	64-bits	1 transfer									
			1st	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00

## 4.3.9 Transfer attributes

The memory attributes of the transfers can be adjusted from SW configurable registers for each command. Security and privilege settings can be adjusted for Secure or privileged channels. Device or memory types can be selected for read and write sides separately. Cacheability and Shareability values can also be set to match the settings of the memory location the DMA transfer is targeting. These settings must match the values defined in other components of the system and can affect the performance of each transfer.

Command link related reads are always sent out as instruction type transfers, that is, arprot[2] is driven 1. Command link accesses inherit the security and privilege settings of the channel.

### 4.3.10 Additional AXI signaling

AXI supports multiple features that are optional for the DMAC to operate:

- The axsize signal is maximized by the data bus width so the MSB might not be used if 64-bit or narrower bus is driven by the DMAC.
- The axlock signals are not used as the DMAC does not generate exclusive or locked transfers.
- The axinner signal indicates inner domain cache attributes for read and write transactions.
- The axchid and axchidvalid signals indicate the software configurable channel ID.
- The arcmdlink signal indicates the current read operation is a command link read.
- The awakeup signals are driven from a register on both manager interfaces to wake the devices in the AXI data path.
- The axdomain signals add minimal ACE5-Lite compatibility.

### 4.3.11 Configurable secondary AXI5 port

The DMA-350 supports an extra complete AXI5 manager port when enabled through a configuration parameter. This allows easier connection to other parts of the system without the

need of an interconnect. The secondary port also allows parallel read and write operations on both interfaces.

Both AXI5 manager ports use the same attributes, for example data width and issuing capability. Both ports are synchronous to the module clock while they have their own clock enable signals.

When using two AXI5 manager ports, the DMA-350 must be able to decide based on a given address whether it should be forwarded to AXI5\_M0 or AXI5\_M1 manager port. A SystemVerilog function is provided for this in a implementer-editable file. The delivered IP bundle contains an example file for this purpose.

For details, see the AXI5 manager port address mapping section of the *Arm® CoreLink™ DMA-350 Controller Configuration and Integration Manual*.

## 4.4 Trigger Interface

The Trigger Interfaces can be used to control the interaction of the DMA channel operation with other peripherals.

An external peripheral or device can control the execution of DMA commands through the trigger input interface. The DMAC can also control external peripherals through its trigger output interface.

The trigger input and trigger output interfaces are compatible with each other. Because of this compatibility, it is also possible to connect two DMA channels together internally, inside the DMAC or by chaining multiple DMA controllers together.

In addition to the hardware Trigger Interfaces, software Trigger Interfaces are provided through channel control registers that enable the software to interact with triggering.

### 4.4.1 Trigger input interface

The trigger input interface can synchronize the operation of a DMA command and a peripheral. The peripheral can signal the DMA-350 when a command or part of a command can be started.

This interface is built up from a simple 4-phase handshake with additional qualifier bus signals that define the type of the current request and acknowledge pair. The interface is defined to be easily compatible with existing Trigger Interface types. The 4-phase handshake also allows simple clock domain crossing structures when the interface is used across different asynchronous clock domains.

## 4.4.2 Trigger input interface signals

When the req signal is pulled HIGH by the peripheral, it signals that the DMAC can start its command or part of its command. The DMAC acknowledges the receipt of the req by asserting the ack HIGH.

There are additional qualifier signals accompanying both the req and ack signals. These signals give additional information to the receiving entity.

Using the reqtype[1:0] signals, the peripheral can supply more detailed information to the DMA-350. The DMAC uses the acktype[1:0] signals to give the peripheral more information along with the ack.

## 4.4.3 Trigger output interface

The trigger output interface can signal a connected peripheral that DMA-350 finished a command.

This interface is built up from a simple 4-phase handshake. The interface is defined to be easily compatible with existing Trigger Interface types. The 4-phase handshake also allows simple clock domain crossing structures when the interface is used over different asynchronous clock domains.

## 4.4.4 Trigger output interface signals

The DMA-350 signals to the external peripheral that a command reached its final state by pulling req signal HIGH. The peripheral must acknowledge receiving this information by pulling the ack signal HIGH. The execution of the DMA command is not completed until this acknowledge is received, which gives a synchronization point between the DMA-350 and the peripheral.

## 4.5 AXI4 stream interface

Each DMA channel can have a dedicated AXI4 stream interface to enable an external engine to do manipulation on the data being transferred through the DMAC. When enabled, all the data read on the AXI is pushed out to the stream interface. The external engine is expected to do the data manipulation which can result in either consuming, keeping the same amount or even generating more data elements. After the conversion is done, the data is sent back to the DMAC on the stream in interface and it is ultimately written out on the AXI write side.

The stream interface is expected to be simply connected to the external engine so no interconnect related sideband signals or null transfers are expected. The stream interface complies with the [AMBA® AXI-Stream Protocol Specification](#) and uses a reduced set of AXI4 stream signals that are necessary for the data transfers.

### Limitations and extensions

The interface data width matches the AXI read and write channel data width, which means that conversion can be done outside the DMAC.

The stream interface also supports continuous aligned and unaligned streams. The DMAC does not send sparse streams and they are not accepted when received from the external engine. `tstrb` signals must be all `1`, except when `tlast` is asserted, which restricts unaligned streams to only have strobes at the end.

Null bytes are not supported, the `tkeep` signal is not present on the interface.

An additional flush sideband output signal is present on the `stream_in` interface. This signal serves as a hint to ask for a `TLAST` from the external engine because of early command finish.

## 4.6 LPI interfaces

The DMAC adds support for low-power integration through the LPI interfaces for both clock and power. The Q-Channel interface provides quiescence capability for the clock and the P-Channel interface allows power management when the DMAC is IDLE and has no activity ongoing. The DMAC can request for power and clock over activity indication signals. The DMAC either accepts or denies the power and clock controller requests based on its current internal state.

### 4.6.1 LPI power P-Channel

The power P-Channel interface is used to request power quiescence from the DMAC. A power controller drives the request while the DMAC either accepts or denies the request based on its current internal state. The DMAC can also request power for an activity over the `pactive` signal.

`pactive` supports a 10-bit wide activity indication from Off to Warm reset. However, the DMAC only asserts Off, Full retention mode and On indications.

The DMAC is not aimed to support forceful power shutdown. When a request occurs and activity is ongoing, the DMAC simply denies the request and continues its operation. The request has no effect on the operation except the P-Channel handshake.

### 4.6.2 LPI clock Q-Channel

The clock Q-Channel interface is used to request clock quiescence from the DMAC. A clock controller drives the request while the DMAC either accepts or denies the request based on its current internal state. The DMAC can also request clock for an activity over the `qactive` signal.

The DMAC is not aimed to support forceful clock shutdown. When a request occurs and activity is ongoing, the DMAC simply denies the request and continues its operation. The request has no effect on the operation except the Q-Channel handshake.

## 4.7 DMA interrupts

Interrupts provide indication of internal state changes of the DMA channel and the DMA unit as well. Each channel has its own separate interrupt. One global Non-secure interrupt is always

present and, in addition, one global Secure interrupt and one Secure violation interrupt also appear when TrustZone support is enabled. Interrupts are level-based signals.

## 4.8 Control and status interface

The control and status interface makes system control of the DMAC operation (stop and pause) possible and provides status information of the DMAC operation.

### 4.8.1 General purpose outputs

The *General Purpose Output* (GPO) ports provide extra bits that you can set to a stable value throughout a DMAC operation.

You can use the GPO ports for multiple purposes:

- Selecting memory banks
- Indicating a channel operation on a debug pin
- Controlling the external data manipulation engine, and more.

GPOs are driven per channel so each channel has its own ports.

The value of the GPO is SW controlled. The value on the output becomes active when the ENABLE register bit is set and remains stable while the DMA channel is active. The GPO stays on the last value set by the command when the channel stops driving the GPO.



The GPO can be set back to the initial value by running an empty command that selects all GPOs and clears them to 0.

---

### 4.8.2 Stop and pause control

The allch\_stop signals can be used to stop the operation of all the active channels of the DMAC by an external hardware unit. The stop function can be useful when dealing with error scenarios in the system and immediate action must clear the DMAC tasks.

The stop waits for all the outstanding responses from read and write transactions but it tries to finish the channel operation as soon as possible by not sending more requests out and not asserting triggers.

These signals are built up from a simple 4-phase handshake. When the allch\_stop\_req\_nonsec is asserted, all Non-secure channels that are not in IDLE are stopped. The software can enable channels but they are immediately stopped when this request is asserted. The assertion of allch\_stop\_ack\_nonsec signals that all Non-secure channels have been stopped or are inactive.

When the security option is enabled (SECEXT\_PRESENT=1) separate signals, allch\_stop\_req\_sec and allch\_stop\_ack\_sec, exist for the Secure channels that operate on the channels in the Secure domain only.

The operation of the DMA channels can be paused immediately at a point in time by the allch\_pause signals operated by an external hardware unit. The pause function can be useful to freeze the channel in a state and check the current values of its registers, or to pause the DMA unit for a while to free up bus infrastructure resources. The enable bit remains asserted and every transfer and trigger state are kept, no information is lost. These signals are built up from a simple 4-phase handshake. When the allch\_pause\_req\_nonsec asserted, all Non-secure channels that are not in IDLE are paused.

The software can enable channels but they are immediately paused when this request is asserted. When the request is deasserted, the operation continues. When the allch\_pause\_ack\_nonsec asserted, all Non-secure channels are paused or inactive. When the security option is enabled (SECEXT\_PRESENT=1) separate signals, allch\_pause\_req\_sec and allch\_pause\_ack\_sec, exist for the Secure channels that operate on the channels in the Secure domain only.

### 4.8.3 Cross Trigger Interface

The DMAC provides a Cross Trigger Interface (CTI) that allows pausing and resuming all channels at once. The CTI is required in a system where a processor is halted for debug purposes and the debugger must save the actual memory contents so the DMAC can also be paused to avoid corrupting the current state of the system.

The interface consists of a halt request, halt\_req, a restart request, restart\_req, and a status signal showing whether the DMAC has halted, halted.

### 4.8.4 Status signals

These signals indicate different status of the individual channels:

- ch\_enabled shows when a channel is active
- ch\_err shows when a channel has encountered an error
- ch\_stopped shows when the channel is stopped
- ch\_paused shows when the channel is paused
- ch\_priv shows the privilege setting of the channel
- ch\_nonsec shows the security setting of the channel



When SECEXT\_PRESENT is set to 1 the ch\_enabled, ch\_err, ch\_stopped, ch\_paused and ch\_priv signals must be separated based on the ch\_nonsec signals.

The ch\_nonsec signals are not present when SECEXT\_PRESENT is set to 0.

---



## 4.9 Configuration interface

The behavior of the DMAC can be configured through static input ports. Static input ports come from either tie-off signals or driven by configurable registers that are stable and contain a valid value when the DMAC is released from reset.

### 4.9.1 Automatic boot interface

The boot interface consists of the following signals:

- `boot_en` - to enable automatic booting,
- `boot_addr` - address of boot command descriptor,
- `boot_memattr` - AXI memory attributes to be used during fetching the boot command descriptor,
- `boot_shareattr` - AXI memory Shareability attributes to be used during fetching the boot command descriptor.

Assertion of the `boot_en` indicates that automatic booting is enabled and the other signals are valid. If `boot_en` is not asserted, automatic booting is disabled and the other boot signals are ignored. All boot signals must be stable when deasserting the reset and remain stable until fetching of the boot command is started.

For more detail on automatic booting, see [Automatic boot feature](#).

## 5. DMAC operation

This section contains an overview of CoreLink DMA-350 operations.

### 5.1 DMAC operation overview

The DMAC can be configured by register writes to execute a vast variety of commands. When the software finalizes the register settings and the command link elements, the channel can be enabled by an additional register write. This step makes all configuration registers become read-only from the software side. The DMAC then starts checking the validity of the command and executes the transfers based on the register settings.

The commands can contain simple memory transfers, scatter-gather type transfers, handling of triggers, general purpose outputs, 2D memory operations, and many other combinations. When a command is finished the DMAC either returns to idle, reloads the command or jumps to a next item in a command list. When the DMAC returns to an IDLE state it can generate interrupts and trigger output signals to synchronize this event with SW or other HW elements in the system. When the DMAC is in IDLE, the registers can be configured again.

The DMAC is also prepared to detect errors during the memory transfer. When a fault is received, it returns an error and stops its operation.

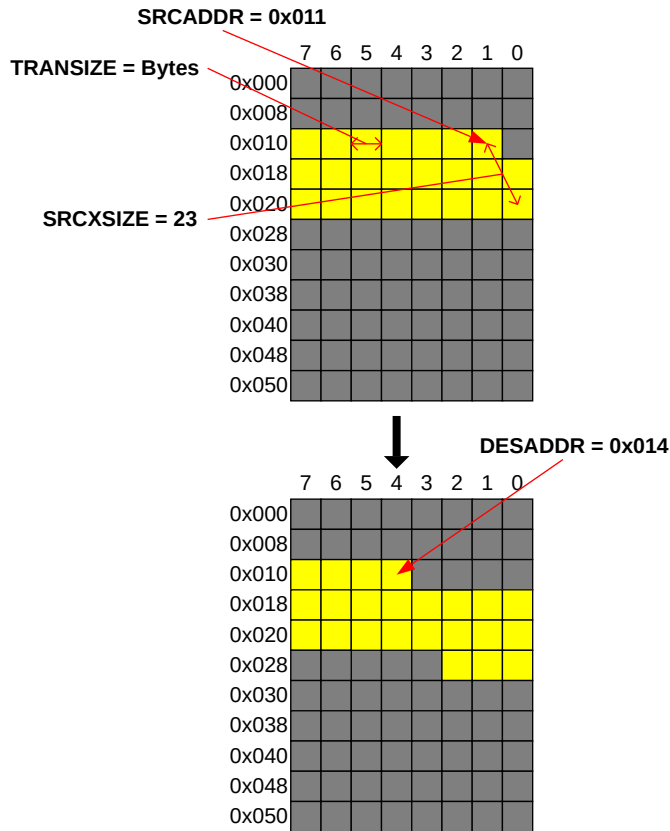
### 5.2 DMAC operation basic commands

This section contains an overview of DMA-350 operation basic commands.

#### Transfer type 1D

Transfer type 1D is a one-dimensional block transfer. It transfers data from the source address to the destination address continuously in configured transfer size. This mode is the typical use-case for bulk data transfer where no data formatting is necessary during DMAC transfer. Transfer size setting range from byte to bus width in power of 2 increments. The transfer size is the same on both source and destination sides. The source and destination address are aligned with the transfer size as the lower address bits are ignored according to the TRANSIZE value. The number of transfers to copy are specified in transfer size increments. The SW can also limit the maximum burst length the DMAC can send to the bus to allow arbitrating others on the interconnect.

**Figure 5-1: 1D transfer example**



**Table 5-1: 1D transfer attributes**

Parameter name	Register map entry	Description
Source Address	CH<x>_SRCADDR	The starting source address to transfer data from. The source address must be aligned with the transfer size.
Destination Address	CH<x>_DESADDR	The starting destination address to transfer data to. The destination address must be aligned with the transfer size.
Source X-size	CH<x>_XSIZE.SRCXSIZE	The source number of transfers in the X dimension. The width of the source transfers is equal to the TRANSIZE value of the source transfers.
Destination X-size	CH<x>_XSIZE.DESXSIZE	The destination number of transfers in the X dimension. The width of the destination transfers is equal to the TRANSIZE value of the destination transfers.
Transfer size	CH<x>_CTRL.TRANSIZE	The data width that is utilized by both the source and the destination of the DMAC operation.
Priority	CH<x>_CTRL.PRIORITY	This signal is passed on to the AXI as AxQOS and is used for arbitration between the channels.
Source maximum burst size	CH<x>_SRCTRANSCFG.SRCMAXBURSTLEN	The maximum burst length that is supported on the source side of the DMAC operation.
Destination maximum burst size	CH<x>_DESTRANSCFG.DESMAXBURSTLEN	The maximum burst length that is supported on the destination side of the DMAC operation.

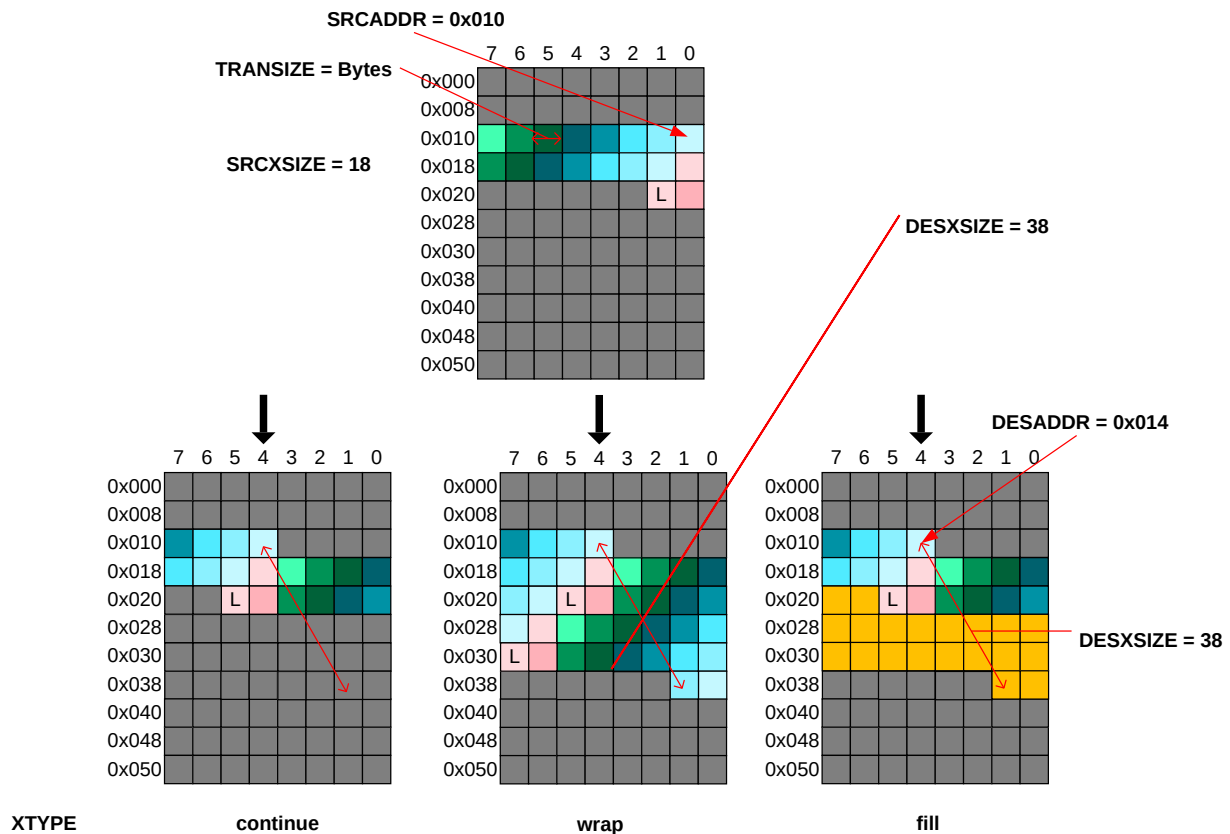
Parameter name	Register map entry	Description
Source AXI transfer properties	CH<x>_SRCTRANSCFG.SRCTRP	Source transfer properties  These properties include memory type, shareability attribute, secure attribute and privilege attribute, and thus determine the values on the arprot, arcache, ardomain and arinner signals.
Destination AXI transfer properties	CH<x>_DESTRANSCFG.DESTRP	Destination transfer properties  These properties include memory type, shareability attribute, secure attribute and privilege attribute, and thus determine the values on the awprot, awcache, awdomain and awinner signals.
1D wrap type	CH<x>_CTRL.XTYPE	Determines how to handle some cases of 1D, when the source and the destination X-sizes are not equal.
Fill value	CH<x>_FILLVAL	The fill value for special wrap cases, when the destination X-size is larger than the source X-size, and the remainder of the address range must be filled with predefined values.
Source address increment	CH<x>_XADDRINC.SRCXADDRINC	Increment used to calculate address on source side. The increment is: <i>TRANSIZE * SRCXADDRINC</i>
Destination address increment	CH<x>_XADDRINC.DESXADDRINC	Increment used to calculate address on destination side. The increment is: <i>TRANSIZE * DESXADDRINC</i>

### 5.2.1 1D operation modes

The WRAP functionality (1D wrap type) extends 1D transfers by allowing different source and destination memory sizes. This allows copying the same data multiple times to the destination location or filling an area with a default pattern.

The figure shows the different patterns, with the L indicating the last item of the source block.

**Figure 5-2: 1D wrap transfers of different types**



When WRAP modes are used for 1D operation the XTYPE setting defines the behavior of the DMAC when copying the source data to the destination. The options are the following:

#### **continue**

Wrapping within one line is disabled, used for 2D cases only. DESXSIZE is ignored. 1D operation type: 1Dbasic

#### **wrap**

Wrapping enabled, the source is copied multiple times to the destination. 1D operation type: 1Dwrap

#### **fill**

Filling enabled, the destination is filled with the predefined pattern when the source runs out of data. 1D operation type: 1Dfill

See the detailed descriptions of these modes in [List of cases for 1D WRAP](#).

## **5.2.2 List of cases for 1D WRAP**

The 1D WRAP transfers can result in the following cases:

### **SRCXSIZE == 0, DESXSIZE == 0**

The DMAC transfer does not start. Possibly only waiting for a trigger.

### **SRCXSIZE == 0, DESXSIZE > 0**

The following options can be selected for the wrap type:

- continue, wrap – Nothing happens, no read or write operation occurs.
- fill – The destination is filled with the predefined fill value, so only writes occur.

### **SRCXSIZE > 0, DESXSIZE == 0**

The following options can be selected for the wrap type:

continue, wrap, fill – no writes occur, only SRCXSIZE number of read transfers are sent by the DMAC. Possibly used with streaming interface to read data and pushed out on the stream output.

### **SRCXSIZE == DESXSIZE**

Essentially a normal 1D operation regardless of the XTYPE setting.

### **SRCXSIZE > DESXSIZE**

The following options can be selected for the wrap type:

continue, wrap, fill – The destination does not have enough room to contain the source memory content, so the copy is stopped when the destination range is filled. No overwrites happen. Excessive data is dropped within the DMAC.

### **SRCXSIZE < DESXSIZE**

The following options can be selected:

- continue – When the source runs out of data the DMAC stops copying data. SRCXSIZE number of reads and writes are executed.
- wrap – When the end of the source data is reached, the source address counter jumps back to the original start address and the starts copying the same source data to all necessary destination memory locations. The source memory location is read multiple times during the complete wrap operation, so the source data needs to remain the same throughout the operation. This is not true when using the stream interface. Essentially, DESXSIZE number of reads and writes are executed even though SRCXSIZE is smaller. The state of the SRCXSIZE and SRCADDR in itself cannot be used to check the number of reads already executed by the DMAC. It can only be calculated from the DESXSIZE and DESADDR where the operation stands.
- fill – A fill value is inserted to every remaining destination location. The fill value is defined in an extra register and the transfer size portion of it is used. The number of reads are equal to the SRCXSIZE, the fill value is generated within the DMAC and it creates DESXSIZE number of writes in total.

### **DESXADDRINC = 0**

Results in a special case of wrapping. This setting shows that the destination address is not incremented, possibly because of targeting a FIFO, so no real wrapping can occur.

- continue – When the source runs out of data the DMAC stops copying data. SRCXSIZE number of reads and writes are executed.
- wrap – Since the destination address is not incremented, the source data is copied to the destination FIFO multiple times depending on the ratio of the SRCXSIZE and DESXSIZE values. The source data is read multiple times and the data is repeated in the destination

FIFO. If SRCXSIZE is 3 and DESXSIZE is 8, the destination FIFO data looks like the following:

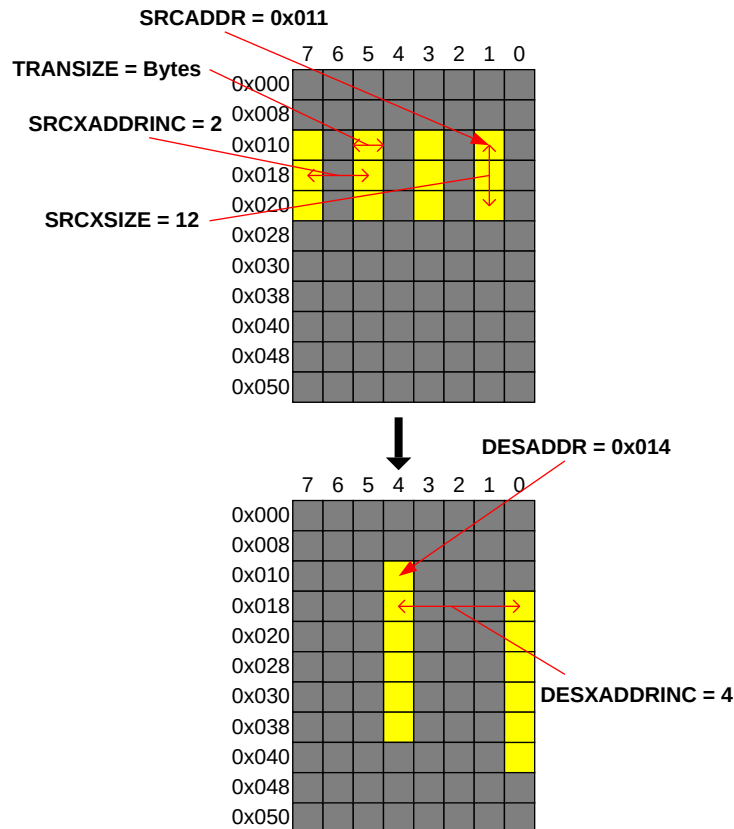
- des\_data[0]=src\_data[0]
  - des\_data[1]=src\_data[1]
  - des\_data[2]=src\_data[2]
  - des\_data[3]=src\_data[0]
  - des\_data[4]=src\_data[1]
  - des\_data[5]=src\_data[2]
  - des\_data[6]=src\_data[0]
  - des\_data[7]=src\_data[1]
- fill – The fill value is copied to the FIFO at the end of the transfer.

### 5.2.3 1D transfers with increments

The 1D transfers also have increment capabilities to the source and destination addresses. It allows transferring memory ranges with gaps to other locations with different gaps. It also allows setting 0 increment to result in a peripheral like access targeting a single memory location with multiple

accesses. The number of transfers is the same on both sides. Increments are based on the *transfer size*.

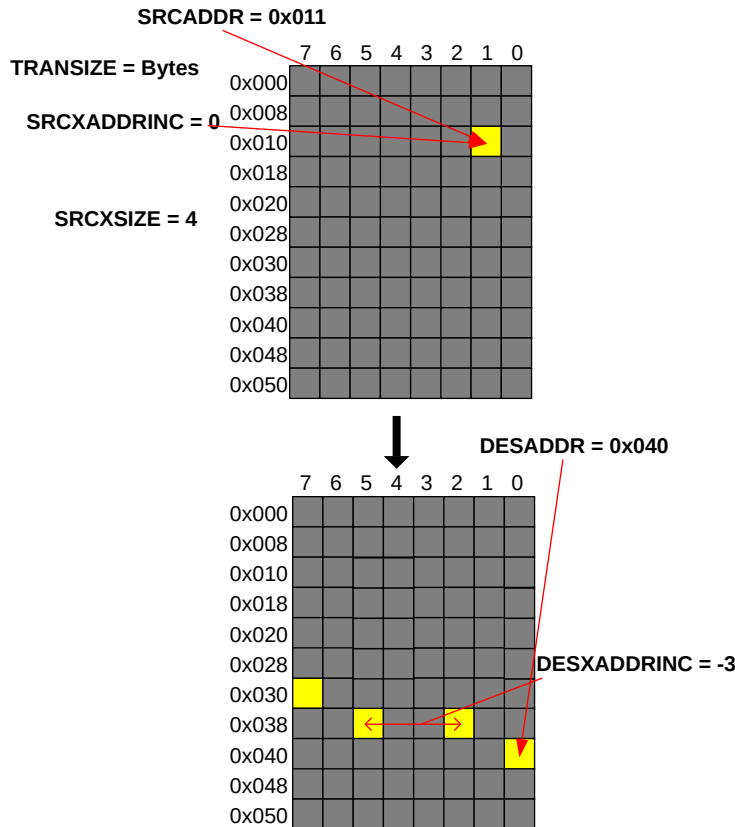
**Figure 5-3: 1D transfer with increments**



The increment value can be negative, too, in case a command must fetch or fill the memory in a different direction. The following example shows a transfer of 4 bytes from the same location to the destination where reverse addressing is used with decrements of 3. All bytes are read from the address 0x11, the first byte is stored at 0x40, the second at 0x3D, the third at 0x3A, and the final fourth at 0x37.



**Figure 5-4: 1D transfer with zero and negative increments**



The DMA-350 supports 16-bit wide increment registers. The range is interpreted a 2s complement where negative numbers range from -32768 (0x8000) to -1 (0xFFFF) and zero or positive numbers range from 0 (0x0000) to 32767 (0x7FFF).

## 5.3 DMAC operation extended commands

This section contains an overview of DMA-350 operation extended commands.

### 5.3.1 Transfer type 2D

The 2D transfers copies image-related data in the same format in the X and Y direction from one location to the other. The 2D operation contains extra registers for YSIZE, which defines the number of XSIZE-wide lines used in the copy.

1D transfers can still be created when the 2D mode is enabled by setting the YSIZE to 1. A YSIZE of 0 means no transfers in this case. When YSIZE is bigger than 1 then XSIZE reloads at the end of the line and jumps from 1 to the starting value. The transition from 1 to 0 means the end of the command and happens at the same time for both XSIZE and YSIZE registers.

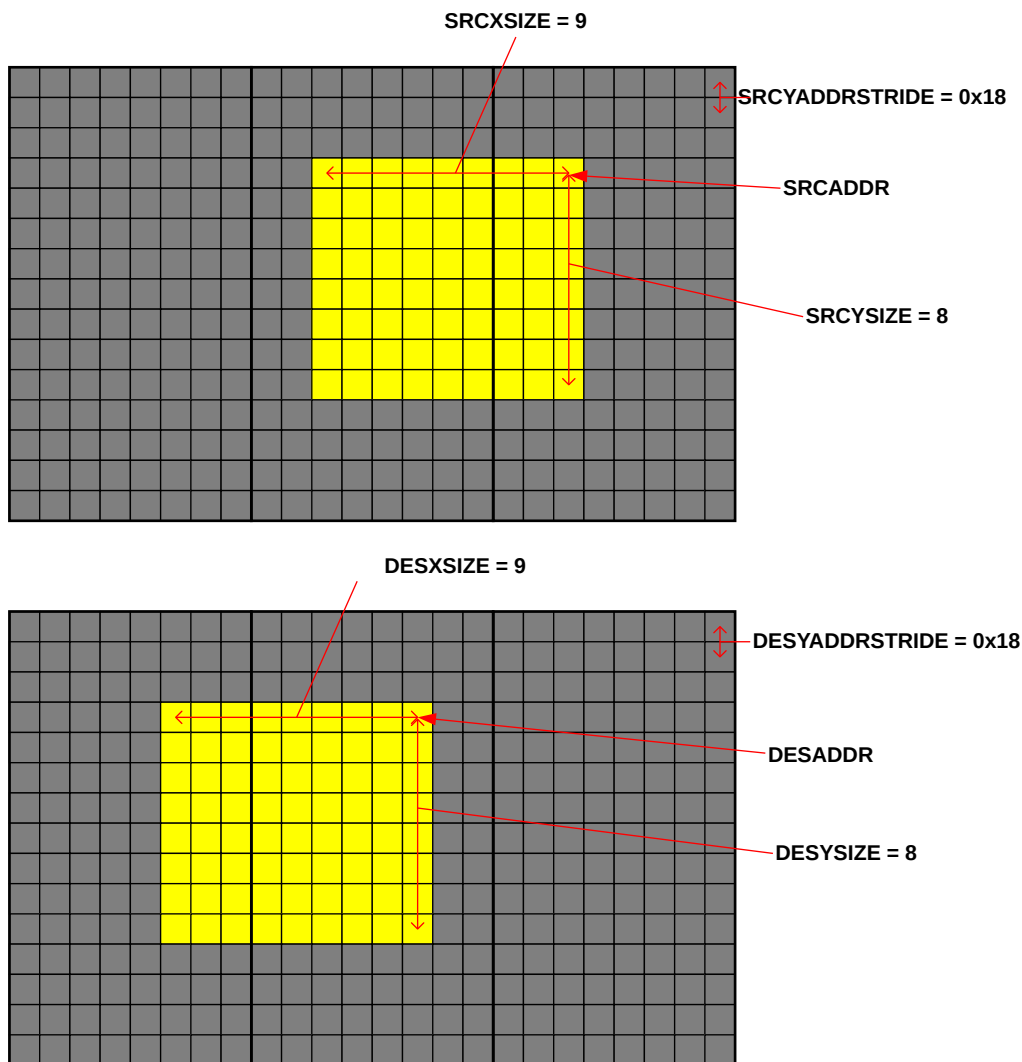
The YADDRSTRIDE register defines the address difference between the starting address of each XSIZE-wide line. The stride values can be positive or negative as well, similar to the XADDRINC.

**Table 5-2: Configurable parameters of the 2D operation**

Parameter name	Register map entry	Description
2D enable	CH<x>_CTRL.YTYPE	This configuration register enables and disables the 2D operation and determines its type.
Source address stride register	CH<x>_YADDRSTRIDE.SRCYADDRSTRIDE	The Y direction address stride at the source. This configuration register calculates the starting address of the next row by incrementing the starting address of the current row.
Destination address stride register	CH<x>_YADDRSTRIDE.DESYADDRSTRIDE	The Y direction address stride at the destination. This configuration register calculates the starting address of the next row by incrementing the starting address of the current row.
Source Y-size	CH<x>_YSIZE.SRCYSIZE	The number of rows of data to be transferred at the source.
Destination Y-size	CH<x>_YSIZE.DESYSIZE	The number of rows of data to be written at the destination.
1D wrap type	CH<x>_CTRL.XTYPE	This configuration register determines how to handle some cases of the 1D operations when the source and destination X-sizes are not equal. The configuration register has added importance at 2D operations, as some of the 1D wrap type operations only influence 2D operations.

In the following example the SRC and DES sizes are equal, but if they are set differently different kinds of wrapping scenarios might occur. These are supported when the WRAP option is enabled. When using 2D transfers, the XADDRINC values can also be used to enable minimal transformation of the copied images.

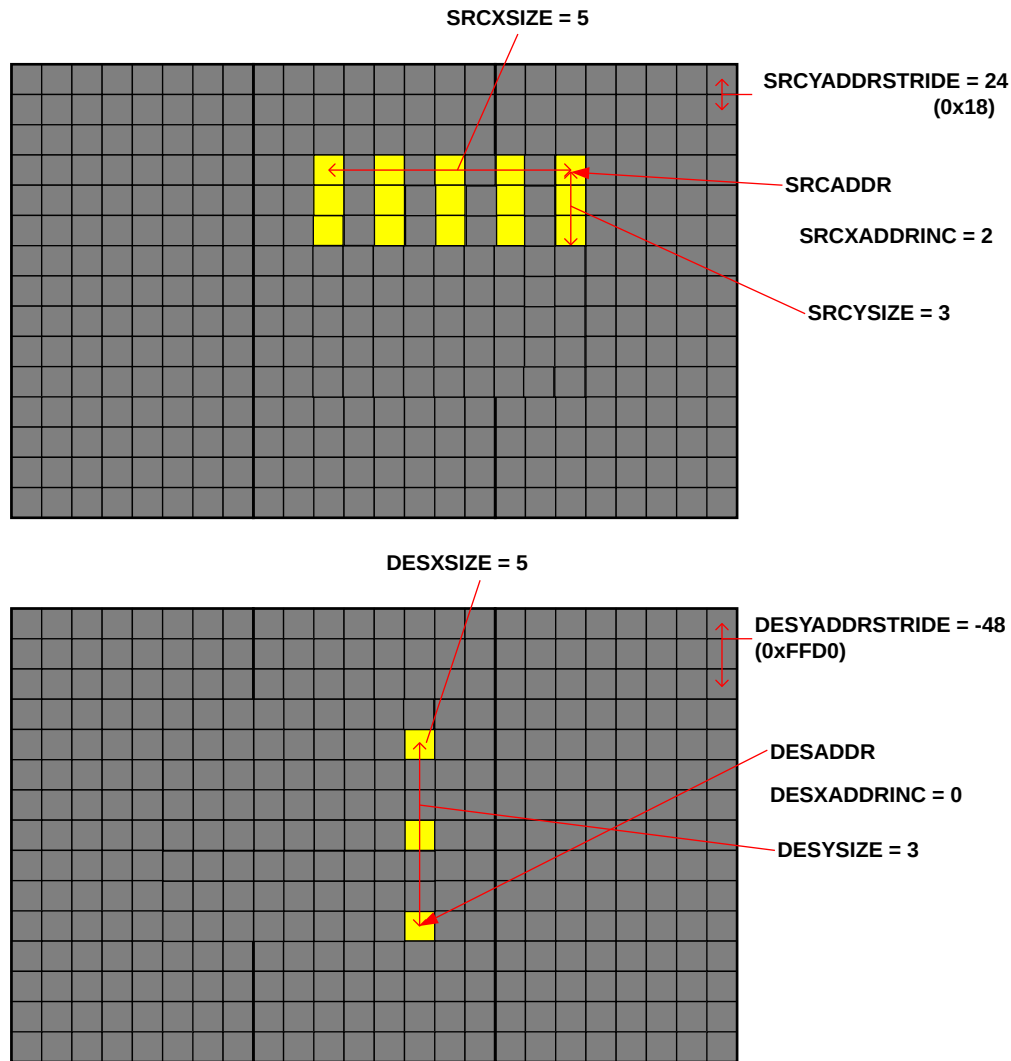
**Figure 5-5: 2D transfer example**



The 1D with increment can also be realized with 2D if XSIZE is set to 1 and the YADDRSTRIDE is equal to the XADDRINC.

The following example shows a scenario where the (SRC/DES)XADDRINC is enabled and gapped data is copied to FIFOs at the destination which use the same address. The DESYADDRSTRIDE is also set to negative to show that the DESADDR value is decremented through the operation.

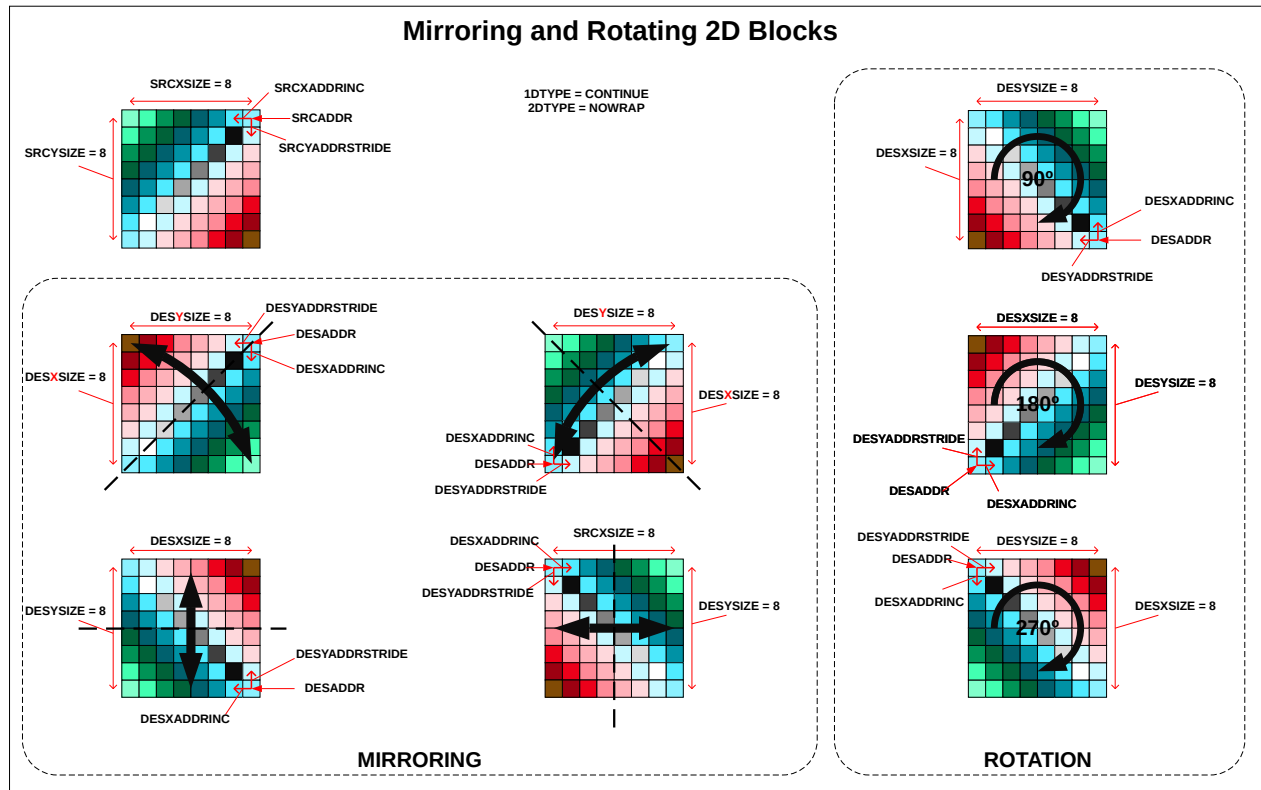
**Figure 5-6: 2D transfer with increments**



Special corner cases might occur if the  $\text{abs}(\text{YADDRSTRIDE})$  value is smaller than the  $\text{abs}(\text{XSIZE} * \text{XADDRINC})$  (considering negative values). In these scenarios, the same or slightly shifted data is read multiple times or the same destination area is overwritten. This might be useful for signal processing purposes. For example, FFT or DCT, where multiple samples in the time domain are combined and used in the frequency domain.

Using negative values for the increments also result in special 2D image copy features that provide mirroring, rotating or transposing images. Using  $\text{XADDRINC}$  to be bigger than  $\text{YADDRSTRIDE}$  and choosing the destination address starting point at an arbitrary location changes X and Y directions. These cases are shown in the following examples.

**Figure 5-7: 2D transformations using increments**



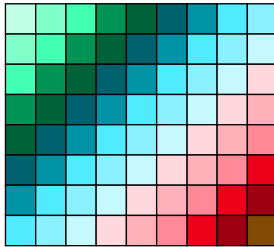
### 5.3.2 WRAP for 2D

The WRAP operations extend the normal 2D copy by allowing different size and arrangement for the destination. The SRCXSIZE and SRCYSIZE parameters of the source can be mapped to a DESXSIZE / DESYSIZE destination location where the sizes can be the same or different between source and destination. Mapping the parameters results in having smaller or larger destination location, therefore requiring different wrap types. This allows reshaping the source data to the destination and also allows filling borders with a predefined fill value.

When you use 2D wrapping, the XTYPE settings are also considered to handle wrap types on a line basis. It also supports a continuous flow of data from one line into different lines on the destination side.

The following figure shows a 9x8 source block where each data item has a different color. This is used in the following examples to show where these items land at the destination location. A black rectangle shows the original size of the frame at the source and the yellow dotted rectangular shape shows the full content of the source data block.

**Figure 5-8: 2D 9x8 size source block**



## Source Block

The YTYPE settings are very similar to the XTYPE values but they work in the Y direction of the transfer.

### disable

No 2D transfer occurs, the YSIZE and YADDRSTRIDE values are ignored.

### continue

Wrapping is disabled, SRCYSIZE number of lines are read and min(SRCYSIZE, DESYSIZE) limits the number of lines to write during the transfer.

### wrap

Wrapping enabled, the source lines are copied multiple times to the destination area.

### fill

Filling enabled, the remaining lines in the destination area are filled with the predefined pattern when the source runs out of data.

### 5.3.2.1 List of cases for 2D WRAP

The following cases are highlighted when YTYPE is not disabled, but because of XSIZE and YSIZE settings they do not always result in real 2D transfers:

**Table 5-3: 2D corner cases**

ID	SRCXSIZE	SRCYSIZE	DESXSIZE	DESYSIZE	Case	Comment
1	0	0	0	0	1	No transfer
2	0	0	0	>0	1	No transfer
3	0	0	>0	0	1	No transfer
4	0	0	>0	>0	2	Write only
5	0	>0	0	0	1	No transfer
6	0	>0	0	>0	1	No transfer
7	0	>0	>0	0	1	No transfer
8	0	>0	>0	>0	2	Write only
9	>0	0	0	0	1	No transfer
10	>0	0	0	>0	1	No transfer
11	>0	0	>0	0	1	No transfer

ID	SRCXSIZE	SRCYSIZE	DESXSIZE	DESYSIZE	Case	Comment
12	>0	0	>0	>0	2	Write only
13	>0	1	>0	1	4	1D to 1D
14	>0	1	>0	>1	5	1D to 2D
15	>0	>0	0	0	3	Read only
16	>0	>0	0	>0	3	Read only
17	>0	>0	>0	0	3	Read only
18	>0	>1	>0	1	5	2D to 1D
19	>0	>1	>0	>1	5	2D to 2D

The cases listed in the table are as follows:

**1**

Source or destination lines have no width or height so transfers are not sent.

**2**

Source area has nothing to send as source has no width or height, but the destination area is defined to be a single line or more lines.

- If XTYPE or YTYPE is set to fill, then the destination area is filled with the fill pattern. YTYPE must be set to fill to use DESYSIZE.
- Otherwise, no transfers are sent.

**3**

Source is defined to be a single line or multiple lines, but the destination area has no width or height.

The source is read but no writes happen. Possibly used for sending data to the stream interface. XTYPE and YTYPE wrap settings are ignored. YTYPE must be other than disable to use SRCYSIZE. No reads happen if stream is not configured properly for this mode, that is, stream output only.

**4**

Source is defined to be a single line, the destination area is also a single line, so it results in a simple 1D to 1D copy.

XTYPE values matter, YTYPE ignored. See [List of cases for 1D WRAP](#) for XTYPE settings.

**5**

At least one of the source or destination areas have a 2D dimensions so it results in 1D to 2D, 2D to 1D or 2D to 2D copy.

Both XTYPE and YTYPE settings matter. See the following sections for more details about the behavior.

### 5.3.2.2 2D wrap cases

The 2D wrap cases are as follows.

#### **SRCXSIZE == DESXSIZE**

When the lengths of the two lines are equal results in simple scenarios where the settings of XTYPE do not matter.

#### **SRCYSIZE == DESYSIZE**

No wrapping occurs, source area is same as destination area.

#### **SRCYSIZE > DESYSIZE**

Destination is smaller than the source, copy stops when destination is filled, regardless of the wrap type. This ensures that no overwrite happens on the destination side. The source data is still read as it might be required for the stream output interface or simply dropped.

#### **SRCYSIZE < DESYSIZE**

Destination is larger than the source so different wrapping options can be used:

YTYPE:

##### **continue**

Stops copying data when SRCYSIZE is reached.

##### **wrap**

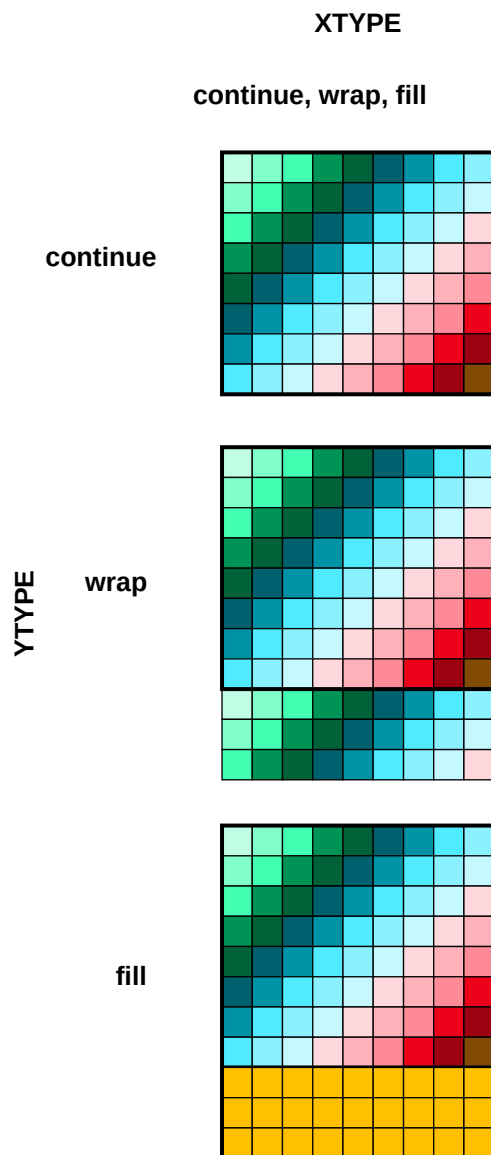
The address counter wraps around and starts copying the beginning of the source data to the remaining part of the destination memory location.

##### **fill**

Fills the remaining Y lines with the fill data



**Figure 5-9: 2D wrap when SRCXSIZE == DESXSIZE**



### **SRCXSIZE > DESXSIZE**

These scenarios cover the cases when the source line is wider than the destination line.

### **SRCYSIZE >= DESYSIZE**

Copying to a smaller area so wrapping of lines on the destination side can occur. YTYPE does not matter since the YSIZE is smaller or equal so no overwrite occurs. The XTYPEs can make a difference in how the data is copied to the destination location.

XTYPE:

### wrap, fill

Since the destination is smaller than the source, the copy stops when the destination line is filled

### continue

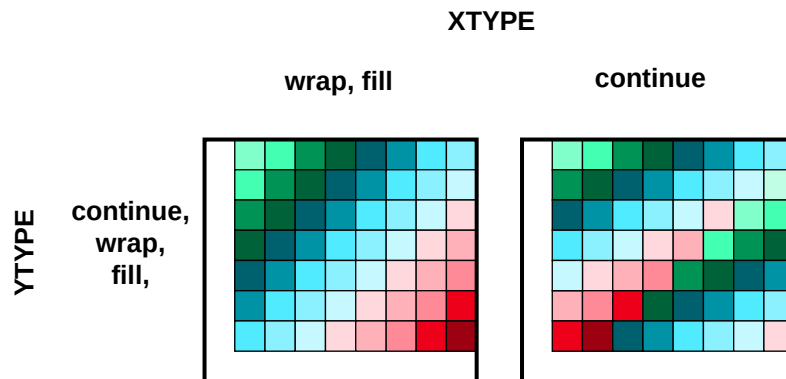
The remaining data from the source line is continuously written to the next line in at the destination. The last few beats of the data that do not fit into the destination are not copied.

YTYPE:

### continue, wrap, fill

The destination height is smaller than the source, so the copy stops when the destination area is filled.

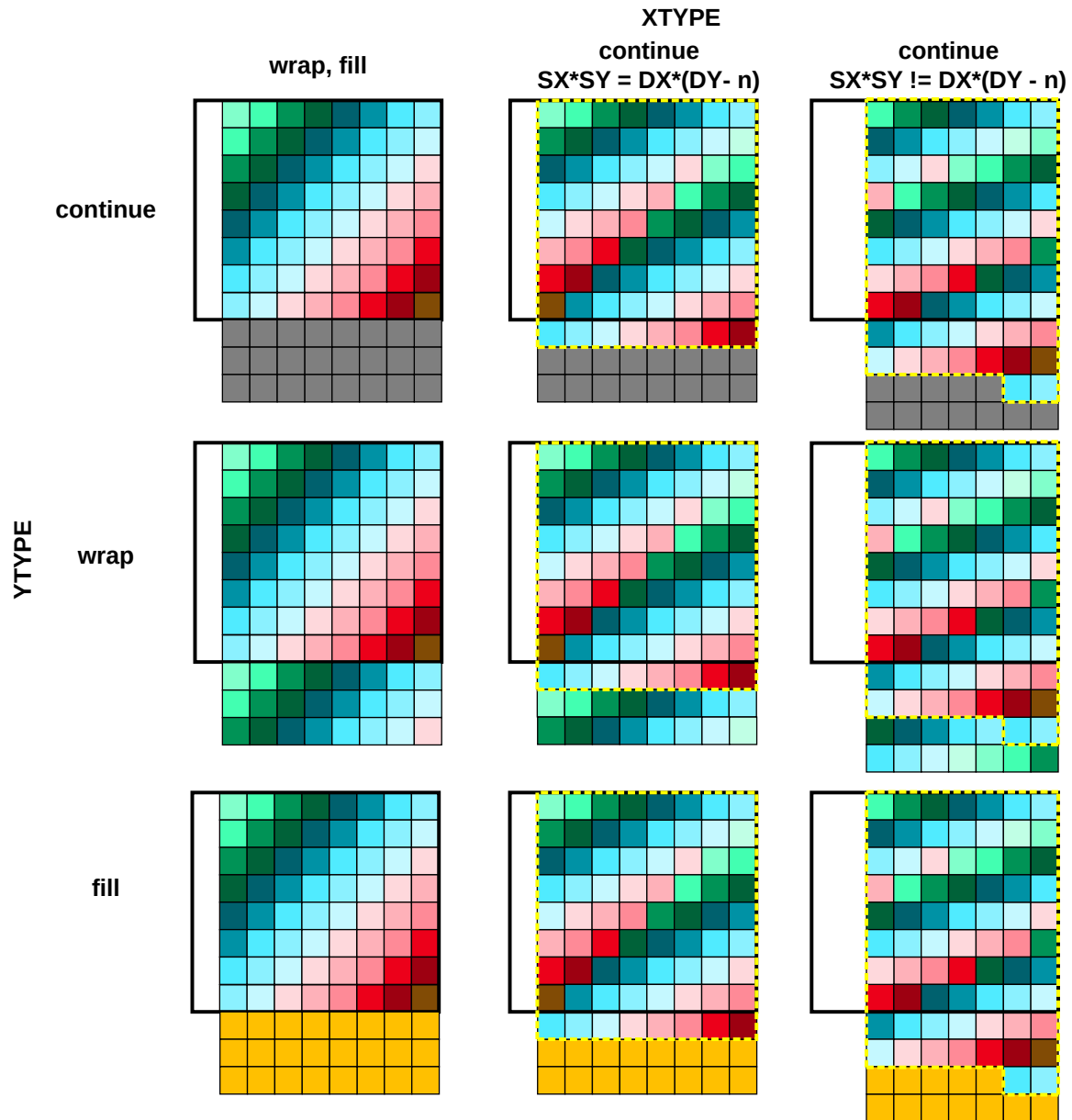
**Figure 5-10: 2D wrap when SRCXSIZE > DESXSIZE and SRCYSIZE >= DESYSIZE**



SRCYSIZE < DESYSIZE

Copying to an area of different shape where X is narrower, but Y is higher at the destination. This reshaping of data could be handled differently with the selection of the 1D and 2D wrap types. The following figure summarizes the options we have when doing the memory copy in these scenarios.

**Figure 5-11: 2D wrap when SRCXSIZE > DESXSIZE and SRCYSIZE < DESYSIZE**



The XTYPE continue, wrap, and fill values do not matter in these scenarios as the X line is narrower. The continue option results in filling the next line with the remaining data from the source line.

There is a special case when the source and destination area is the same size, but different shape. This example shows  $SX=9$ ,  $SY=8$ ,  $DX=8$ ,  $DY=9$  case in the middle column in the figure where the  $n$  number in the equation is 0. In this case the YTYPE has no meaning because the complete data from the source fills the destination, so no wrapping or filling can occur. If  $n$  is greater than 0, it

means that there are full DESXSIZE-wide lines remaining empty after all the data is used from the source and in this case wrapping and filling can occur.

The other continue column shows the case when the last line in the destination is only partially filled with the source data. These scenarios can also result in wrapping and filling but these start in the middle of the destination line.

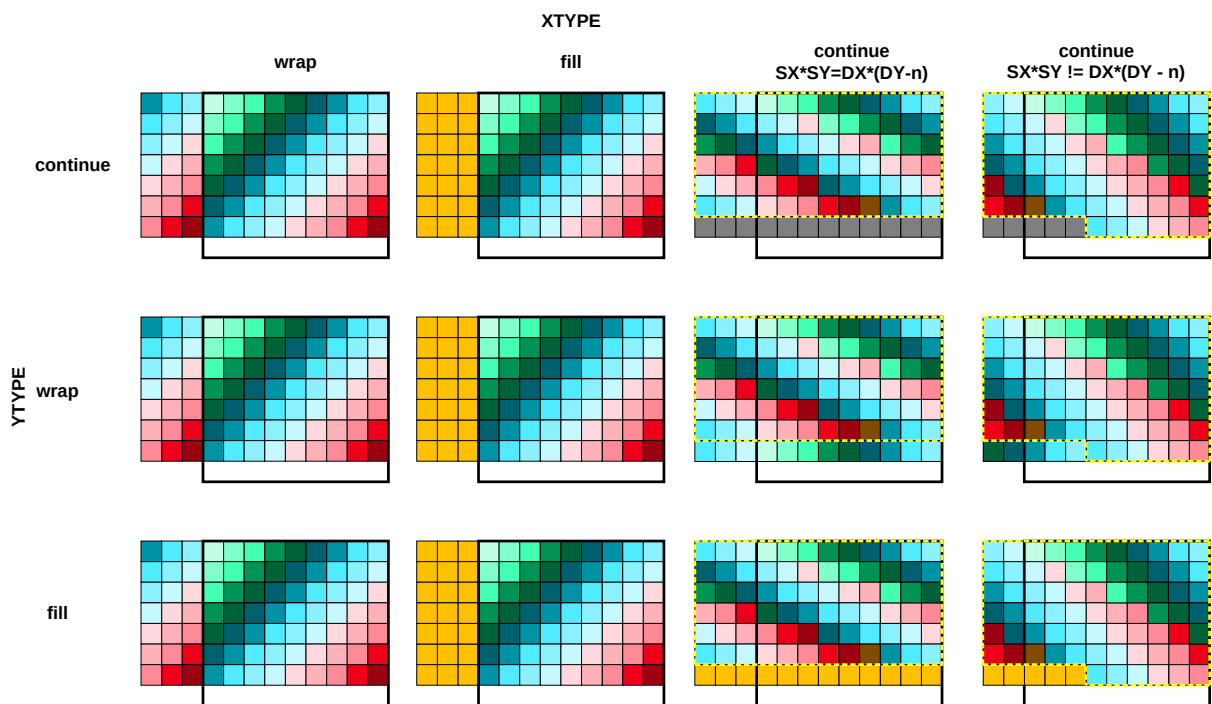
## SRCXSIZE < DESXSIZE

The following scenarios cover the cases when the line at the destination is wider than the source.

### SRCYSIZE >= DESYSIZE

The source rectangle can still be higher than the destination one which results in different situations. The source data can be truncated if it cannot fit into the destination, or it is reshaped when the destination rectangle size matches the source area, or even some wrapping can occur if the destination is larger than the source. The XTYPE also counts since the destination line is wider than the source, so different wrapping types can result in different patterns for a single line.

**Figure 5-12: 2D wrap when SRCXSIZE < DESXSIZE and SRCYSIZE >= DESYSIZE**



The YTYPE continue, wrap, fill values do the same if the XTYPE is continued, wrap or fill. This is because having less or equal lines in the destination than the source means the destination lines are not overwritten.

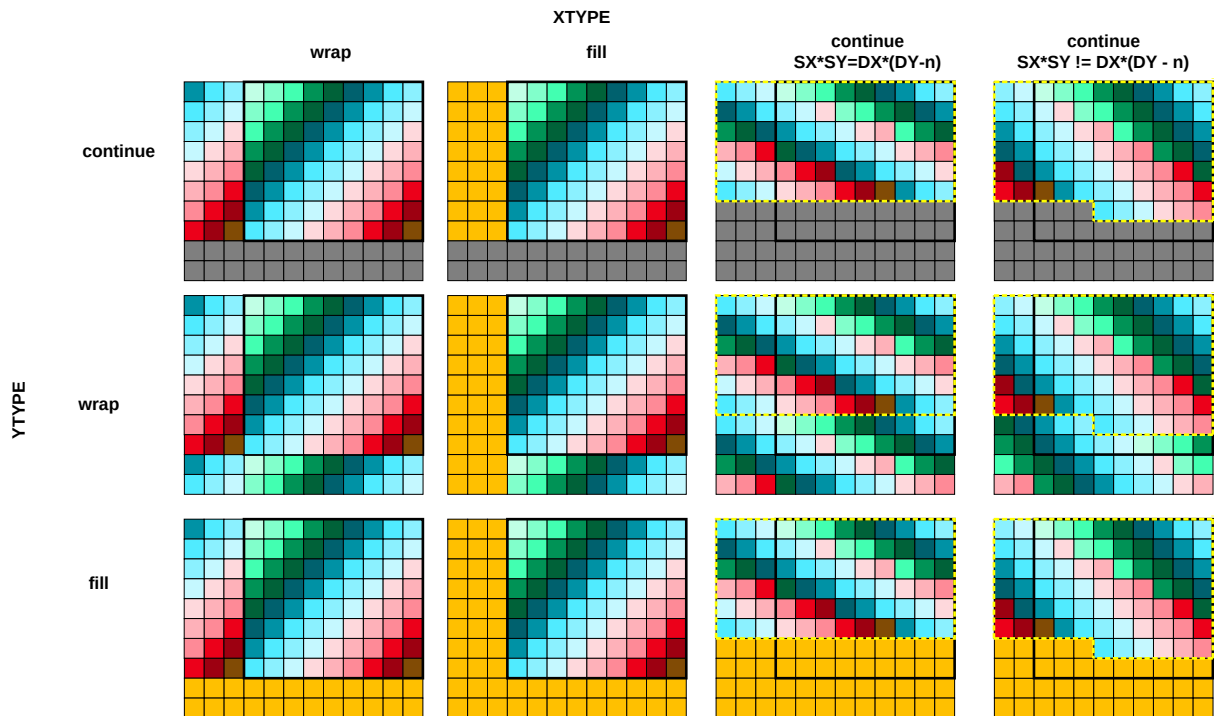
The XTYPE continue case is special in this sense because multiple source lines are merged to fill a single destination line continuously. This enables the reshaping of data at the destination and the

YTYPE takes effect in how the remaining space is filled at the destination. This case also has special subcases when the line is completely filled at the destination side and wrapping or filling can occur on a full line level. The other case when the destination line is only partially filled with the last data beats from the source. The remaining part of the destination can also contain wrapped source data from the beginning or it can be filled with the predefined pattern.

SRCYSIZE < DESYSIZE

This scenario shows when the destination rectangle is bigger in all directions so wrapping can occur in both 1D and 2D directions. The difference to the previous case is that YTYPE also matters when XTYPE is set to continue, wrap or fill. The XTYPE continue setting works the same way as in the previous case.

**Figure 5-13: 2D wrap when SRCXSIZE < DESXSIZE and SRCYSIZE < DESYSIZE**



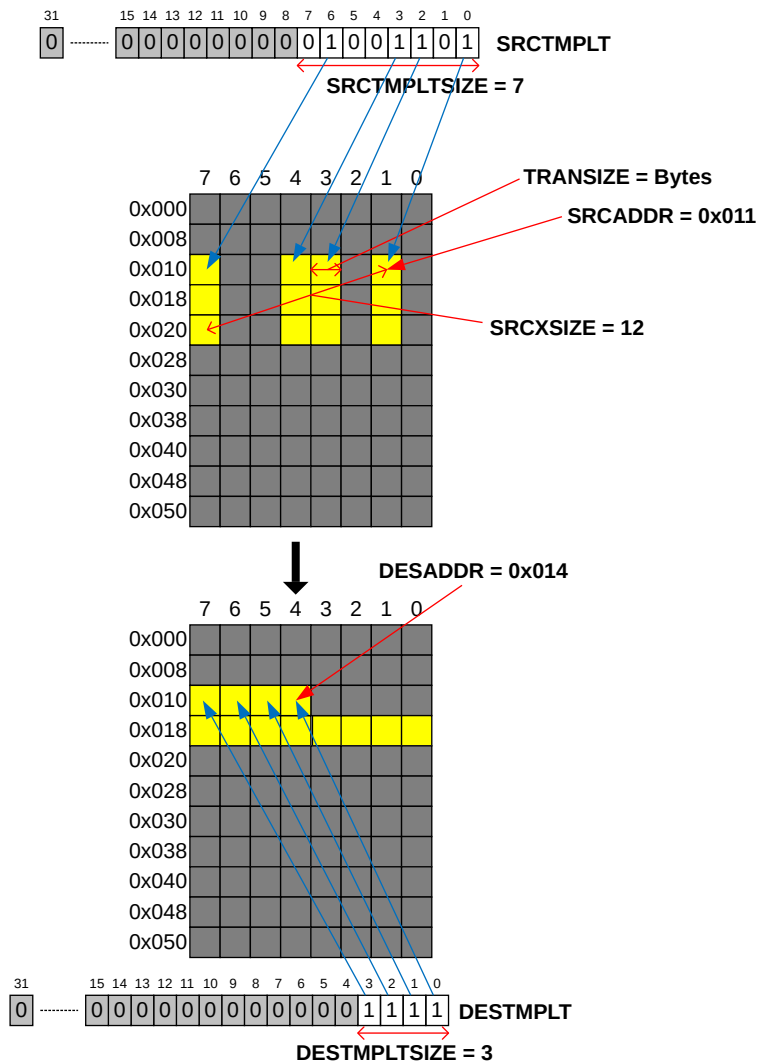
The XTYPE continue cases may extend or fill the destination area in Y direction depending on the YTYPE setting. The YTYPE continue cases can extend or fill the destination area in the X direction. It can also be used for 1D to 2D copies when selecting the continue option so the source data is copied to the destination in a continuous manner. When wrap is set to both types, it can serve as tiling the same data to a larger area. Fill in both directions can create a border around two sides of the image. A full border on the two other sides can only be created by linking multiple commands together. Border on the up and right side of the image can be created by using negative increments.

### 5.3.3 Templated transfers

The templated transfer feature allows 1D transfers to selectively copy data using predefined patterns. The patterns are defined by template registers that describe repetitive bit-mask of addresses to be transferred.

The following diagram gives an example for templated transfers:

**Figure 5-14: TMPLT**



The bit masks are defined in the template registers SRCTMPLT / DESTMPLT. There are separate source and destination template registers as well as source and destination template size fields that define the length of the templates. The source and destination sides can be configured independently.

The lowest bit of the template registers are fixed to 1. There cannot be initial gaps in the transfer and there is always data transferred at the start addresses defined by SRCADDR / DESADDR. If an

initial gap is still required, the addresses must be displaced so that the first transfers are located at SRCADDR / DESADDR.

The actual template sizes are defined as SRCTMPLTSIZE + 1 / DESTMPLTSIZE + 1. The default settings of zeroes mean that templating is disabled. The template feature is enabled by setting one of the template sizes to be greater than one, that is SRCTMPLTSIZE > 0 and DESTMPLTSIZE > 0 or both. The higher bits of the mask in the template registers that fall out of the defined template size are ignored.

### Address calculation

The templated transfer feature always uses single transfers to transfer data, it does not try to combine the consecutive transfers into bursts. A new address is calculated for the transfer in each cycle.

### Templated transfers with stream interface

The stream interface (AXI4-Stream) is not aware of the template type as it always receives packed data. Only the AXI addressing deals with the patterns on both source and destination sides.

## 5.4 DMAC operation triggers

Trigger inputs and outputs can synchronize activities within a system without SW intervention. The DMAC can have trigger inputs and outputs on a channel basis when enabled. The input triggers support multiple modes for different purposes, while the output triggers be used to mark the final step of a complete DMAC operation. The trigger matrix part of the DMA-350 handles the connections between the external trigger ports and the DMA channels, making it possible for the DMA channels to select which trigger ports to use or which other DMA channel to connect to.

### 5.4.1 Trigger inputs

Trigger input ports use the trigger signals to enable peripherals to communicate with the DMAC and sequence DMAC operations without SW intervention. Triggers provide flow control for transfers within a DMAC operation and can also be used to start entire DMAC operations. The mode in which the trigger input is used can be selected through configurable registers.

A 4-phase handshake bus is used between the peripheral and the DMA-350. Extra qualifier signals extend the req and ack signal pair in both directions. The requesting peripheral can signal to the DMAC that it has data to be transferred by the req signal. The additional qualifier signals, reqtype, are used to show the DMAC that it has single or block data to be transferred, and that it has the last data amount to be transferred. The different request types can be seen in the following table.

**Table 5-4: Trigger request types**

Value	Name	Description
2'b00	SINGLE	The peripheral can ask for a single beat of the defined transfer size.
2'b10	BLOCK	The peripheral can ask for a larger block of transfer size elements. The block size is defined in the DMAC and in the peripheral as well and can comprise multiple bursts.

Value	Name	Description
2'b01	LAST SINGLE	When the peripheral is the flow controller of the transfer and the DMAC does not know the size of the transferable data, the peripheral can close the transfer by requesting a final single beat.
2'b11	LAST BLOCK	Similar to LAST SINGLE but the final transfer request is the block size defined in the DMAC and the peripheral. This is used when the complete transfer size is divisible by block size operations.

The DMA-350 shows the peripheral that the ack signal received the request. The additional qualifier bus signals, acktype, provide the peripheral with extra information that the trigger request was accepted or not, or if the DMAC transfer is finished after the current transaction.

**Table 5-5: Trigger acknowledgment types**

Value	Name	Description
2'b00	OKAY	This acknowledge type indicates the DMAC accepted the request type. The DMAC indicates that it executes the transfer, but not necessarily at the time of the acknowledge. The DMAC may accept multiple requests before executing the requested operation.
2'b10	LAST OKAY	This acknowledge type indicates that the DMAC accepted the request type and tells the peripheral that this is the final transfer of the operation. This acknowledge type has two purposes: <ul style="list-style-type: none"> <li>The DMAC acknowledges a LAST request from the peripheral to show that both ends are in sync.</li> <li>The DMAC is the flow controller and tells the peripheral that no more transfers are expected, so the peripheral needs to reset its transfer counters. For example, a shorter burst is sent as a final transfer.</li> </ul>
2'b01	DENY	This acknowledge type indicates that the DMAC cannot accept this request now. This can be useful when the DMAC is the flow controller and the peripheral requests SINGLE transfers, but the DMAC expects more transfers to accumulate and optimizes for a block transfer. In this case, the DMAC denies the SINGLE requests so that the handshake can go back to IDLE state.
2'b11	RESERVED	Not used.

A channel has two trigger input ports when enabled:

- One for source triggering to control read operations.
- One for destination triggering to control write operations.

The number of trigger inputs are configurable and can be different than the number of channels. The SW is able to select trigger inputs for channels using the trigger matrix, see [Trigger matrix for selectable sources](#).

When a trigger port is present but SW wants to disable triggering, the triggers can be set to disabled mode for the command in the control register. In this mode, the DMA channel does not wait for any external input to happen and starts execution after the DMAC Enable bit is set.

Trigger input modes are set through SW programmable registers. Trigger related registers can only be adjusted when the DMA channel is idle.

The trigger inputs are connected to the DMA channel after the DMA channel is enabled. The connections are kept until the current command lasts. The triggers are given back to the trigger matrix when all the operations are finished.

When a channel finishes a command, there might be a pending req on the trigger input port used by the channel. This can happen when a trigger request arrives after a command has received all the expected trigger requests and before the trigger input is returned to the trigger matrix.



The req signal on the top-level trigger port remains pending without getting an ack signal. Similarly, when a req signal arrives on a trigger input that is not connected to any channel, it does not get an ack signal, and the req signal remains pending. When a channel is configured to use a trigger input port with pending request, the channel receives the req immediately after channel enable.

In a normal operation, the DMA channel releases the output trigger when the trigger operation is finished. If the channel receives a stop signal and there is already a pending request on the trigger port, the channel releases the port immediately without waiting for the req-ack handshake to complete. The channel is ready to be configured for a new command. However, the DMA-350 waits for the handshake to be finished on the trigger port and no other channel can select this port until then.

When a channel is paused, the pending triggers are left as is, and the service is pending. The interface can stall in any state. During a pause, external signals, like a req signal in the trigger input or an ack signal in the trigger output, can change state. After the channel operation is resumed, the trigger operation continues.

When a DMA channel is stopped, the trigger matrix takes back the control over the trigger input. The channel services an already acknowledged trigger before giving back the control to the matrix. The interface can be left in idle or request state. For the trigger output, the trigger matrix takes back the control and responds to the already pending triggers for the channel by an internal ack signal. The pending external triggers must be served by the trigger matrix by waiting for an ack signal to arrive. After the ack signal has arrived, the req signal is pulled LOW. The interface returns to idle state.

The SW can initiate a trigger request automatic clear. The DMAC can send a “deny” acknowledge to a pending request on a trigger input. This way, the unneeded pending request can be cleared without trigger protocol violation.



For debug purposes, the SW can mimic the external HW trigger. See [Software triggers](#) for more information.

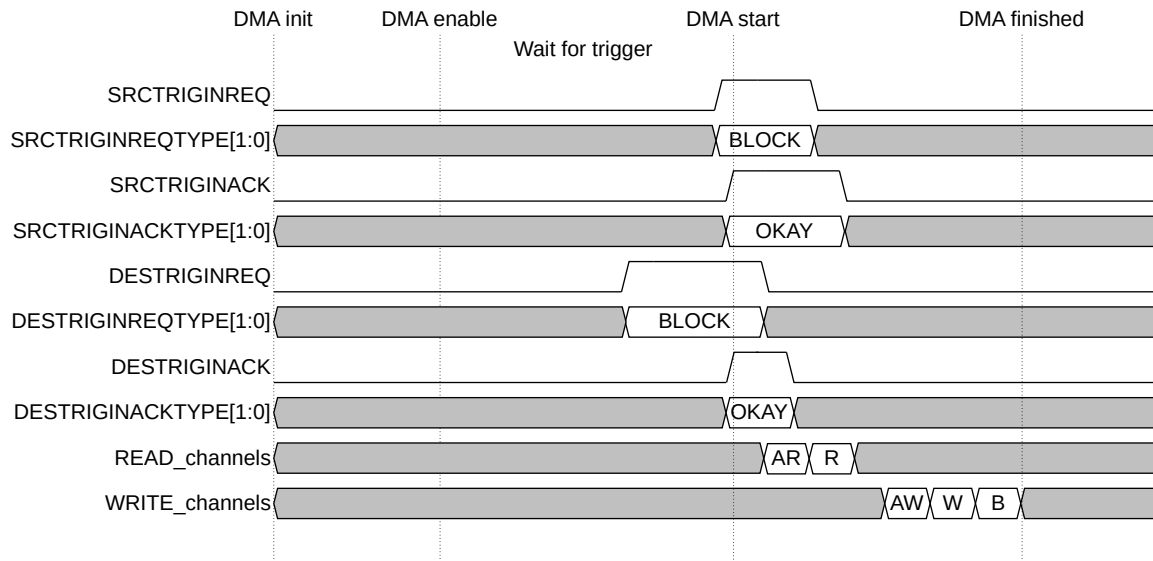
---

#### 5.4.1.1 Trigger input command mode

Trigger inputs can be used to initiate full DMA commands. When either source or destination side is set to this mode, the DMA command waits the trigger handshake before it actually starts any memory operation. This is true even if the DMA channel's enable bit is set. The DMAC waits for

both requests to be asserted before acknowledging them to show when the command is really started.

**Figure 5-15: Command trigger for both source and destination**



The figure shows that the destination trigger request is received first. It is not acknowledged until the DMAC also detects the source trigger request. At this point, both source and destination triggers are acknowledged and the DMAC transfers can start. The trigger handshake can return to IDLE at different points in time depending on the driver of the request signal but it does not interfere with the execution of the data transfers. However, the command can only finish when the trigger acknowledge signals are both deasserted.

When either source or destination side is set to command mode triggering, the DMA channel operation waits for the trigger handshake to happen, ack HIGH, to start operation. Without the handshake, the DMAC is in Active waiting state and not sending transfers to the bus interface.

When all command-based trigger input handshakes are done, meaning req and ack are both HIGH, the command execution can start and the DMA channel exits from the waiting state. The DMAC does not accept new trigger requests until the complete DMAC operation lasts.

For command mode triggers, the reqtype[1:0] can take any of the following values: SINGLE / BLOCK / LAST SINGLE / LAST BLOCK.

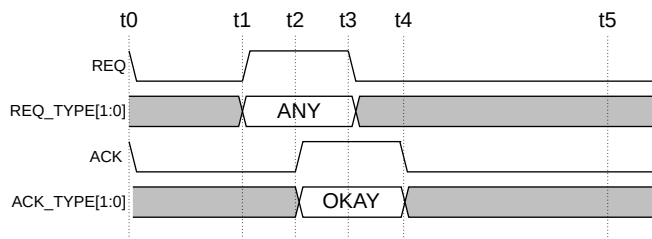
The acktype[1:0] value of OKAY / LAST OKAY are both valid for command mode triggers. The DMA-350 returns OKAY value. The DMA channel acktype[1:0] value of DENY is not used in case of command triggering.

Both the source and destination side can be set to Hardware/Software controlled command mode. In this mode, the DMA channel waits for trigger to start operation. The trigger input is primarily external hardware trigger.

However, the input trigger requests can be overridden by generating internal trigger request events by writing an SW programmable register. When either type of trigger arrives, the DMA channel starts its operation. If an SW trigger arrives when the command is already running nothing happens. If an HW trigger arrives when the SW already requested a trigger, the HW trigger is left pending. The HW trigger gets serviced when both arrive at the same time.

When the trigger request is deasserted for a command trigger, the DMAC deasserts the acknowledge as soon as possible. This enables the sender of the trigger to continue its operation.

**Figure 5-16: Trigger input ACK assertion and deassertion for command triggers**



The figure shows the ack deassertion approach of the DMAC when using command triggers:

**t0**

The Trigger Interface is in IDLE.

**t1**

The peripheral requests any type of trigger to start the command.

**t2**

The DMAC accepts the request if the DMA channel is waiting for the trigger and ready to execute the operation.

**t3**

The peripheral deasserts the req signal when it receives the OKAY response from the DMAC. The DMAC still executes the requested operation.

**t4**

The DMAC detects that the req is deasserted and it deasserts the ack so the Trigger Interface returns to IDLE. The command is still running.

**t5**

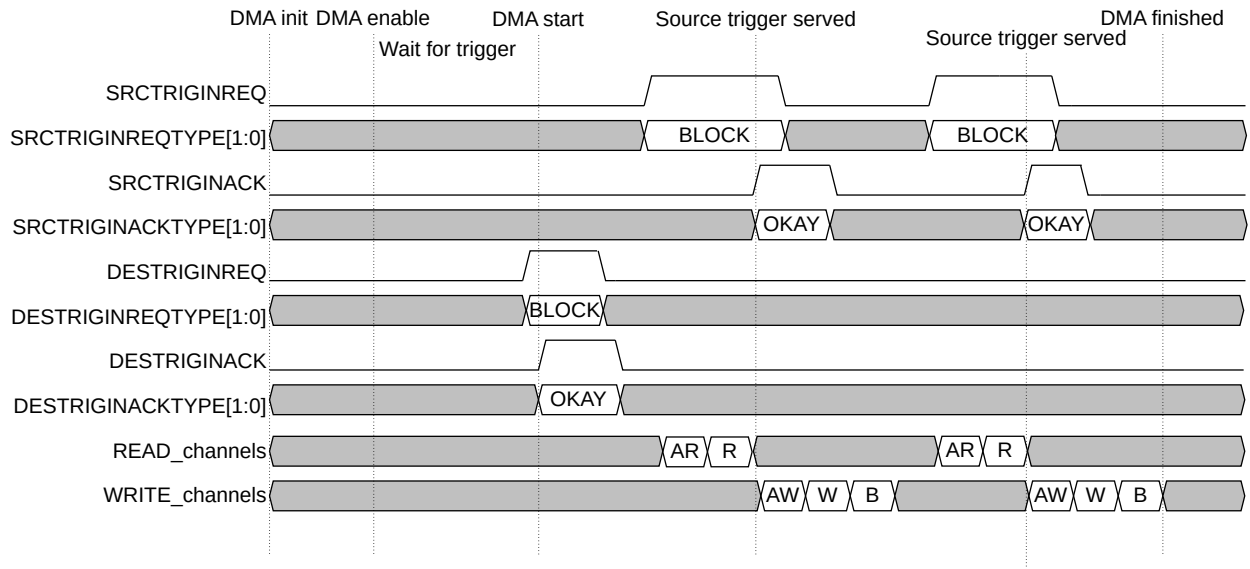
The command is finished but it is not reflected on the trigger input interface. Trigger out signals can be used to show the end of a command.

### 5.4.1.2 Trigger input flow control mode

Trigger inputs can also be used for flow control purposes and only enable small parts of a larger command for every trigger. For this mode, the SW must set the trigger size which tells the number of transfer size blocks to be accessed for one trigger event. The source trigger input is used

to control the read flow, while the destination trigger input controls the write flow of a DMA operation.

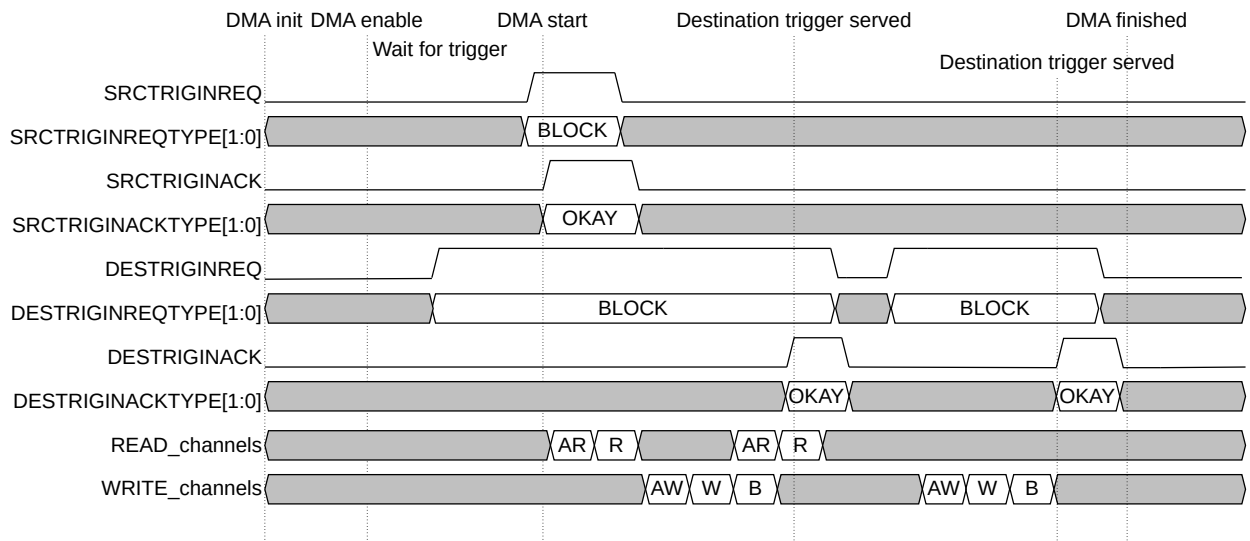
**Figure 5-17: Flow control mode trigger for source and command for destination**



The figure shows that the read transfers can only start after the destination command trigger and the source trigger is also received. Writes on the destination side can only start when there is enough data received from the source side.

When source trigger is set to flow control mode, read operations of a DMA channel stall until the first trigger request is received.

**Figure 5-18: Command trigger for source and block for destination**



The figure shows that the destination trigger request arrives first but the command still must wait for the source trigger to arrive. The read transfers start right after the source command trigger is received. The writes can only start when the destination trigger is received and there is enough data to send. The DMAC preloads the data in its internal FIFO so that the writes can happen as soon as possible when the next destination trigger is received.

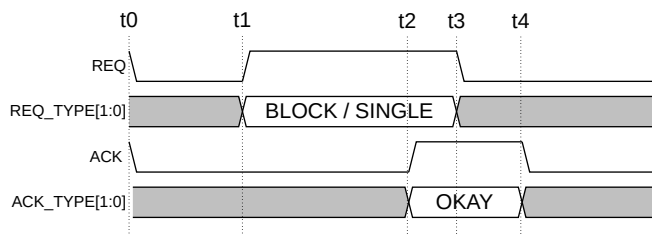
When destination trigger is set, write operations of a DMA channel stall until it receives the first trigger request. Read operations can already start and fill the FIFO before the destination trigger handshake is done.

When a trigger input request is received, the command execution can start and the DMA channel exits from the waiting state. The DMAC only accesses the number of transfers specified by the block size register as a maximum. The DMAC does not raise the trigger acknowledge until the DMA operation for the block size amount of transfers last (allowing smaller blocks for the final block) and the response of the last transfer is received.

The DMA channel waits for all response transactions to arrive before the acknowledge is asserted. The handshake must return to idle for the state machine to return to the “wait for trigger” or “finished” state. This adds higher latency between triggers as the address generation stalls until the last response is received. However, it also avoids timing conflicts between parallel trigger acknowledgments and AXI transfers arriving at the peripheral side.

The DMA channel can be set to allow DMAC flow-control operation for both source and destination sides. In this mode, the DMA channel counts the number of transfers and, when the required number is reached, it ends the DMA channel operation. The DMA channel sends OKAY ACK to the peripheral when there are still transfer size blocks to be transferred. When the last transfer size block is reached the DMA channel sends LAST OKAY ack.

**Figure 5-19: Trigger input ACK assertion and deassertion for block triggers**



The figure shows the ack deassertion approach of the DMAC when using block triggers.

**t0**

The Trigger Interface is in IDLE.

**t1**

The peripheral requests a BLOCK or SINGLE trigger type.

**t2**

The DMAC started the execution of the request as soon as possible and at t2 all transfers requested for this operation are finished.

**t3**

The peripheral deasserts the req signal when it receives the OKAY response from the DMAC.

**t4**

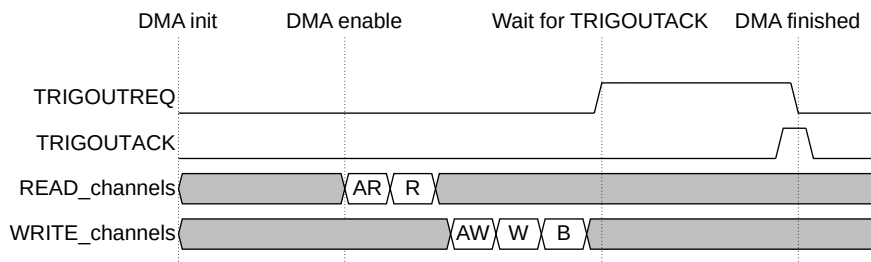
The DMAC is finished with servicing the request and deasserts the ack signal when it is ready to accept a new request.

## 5.4.2 Trigger outputs

Trigger outputs define the end of the command execution before the DONE interrupt is asserted. They can be used to synchronize activities between HW elements in the system.

When waiting for trigger output acknowledge, the channel operation is stalled until the acknowledge is received and trigger output interface of the channel returns back to IDLE.

**Figure 5-20: Trigger output request assertion**



A channel has one trigger output port when enabled.



For debug purposes, the SW can mimic the external HW trigger. See [Software triggers](#) for more information.

## 5.4.3 Trigger matrix for selectable sources

The DMA-350 implements a trigger matrix for both trigger inputs and outputs as each trigger can be assigned to a channel based on the register settings. The trigger matrix can also create a connection between one channel's trigger output and another channel's trigger input port to have internal events starting triggers.

The number of trigger inputs can be configurable and can be different than the number of channels. The SW is able to select trigger inputs for channels using the trigger matrix.

The DMA-350 manages the selection procedure and only allows valid connections between triggers and channels. The security attribute of the channels and the trigger ports are checked and a trigger port cannot be selected by more than one channel at the same time. There are scenarios when the external trigger ports are not selected, but are still busy with handling requests because of

software intervention. These scenarios are also covered by the matrix and the trigger ports are not selectable until the software intervention side effects are settled.

The trigger input selection can result in an error if the trigger input interface is not selectable for the following reasons:

- It is used by a different channel.
- Trigger port is not selected, but more than one channel tries to select it in the same cycle.
- Not allowed for selection because of security attribute differences.
- Trigger in interface is not in IDLE because of an ongoing software initiated clear handshake.
- When using internal triggers, the trigger input port cannot be connected to the trigger output port of the same channel.

The trigger output selection can result in an error in case the trigger output interface is not selectable for the following reasons:

- It is used by a different channel.
- Trigger port is not selected, but more than one channel tries to select it in the same cycle.
- Not allowed for selection because of security attribute differences of the channel and the trigger port.
- Trigger out interface is not in IDLE because of a previously unfinished trigger handshake. This might be a pending req not receiving and ack or an ack stuck HIGH at the end of the handshake.
- When using internal triggers, the trigger output port cannot be connected to the trigger input port of the same channel.

#### 5.4.4 Internal trigger connection

The DMA-350 enables synchronization of two channels by internal trigger connection. The trigger output port of a channel (sending channel) can be internally connected to a trigger input port of another channel (receiving channel). The req on the trigger output port signals the finish of a command and starts the operation of another command on the receiving channel.

A channel that is using internal triggering is always in command trigger mode regardless of the configured trigger type.

The sending channel is waiting for the trigger ack from the receiving channel. When the receiving channel is configured to command trigger, the ack is deasserted right after the sending channel deasserts the req. This means the sending channel can continue with a new command when the receiving channel started its command.

A chain of more than two channels with internally connected triggers can be formed. A closed loop can be formed from such connected channels. In such a case, software trigger can be used to start the very first operation of the chain.

When the number of channels is only one, internal triggering cannot be used. If internal hardware trigger mode is configured, it is treated as external hardware trigger mode.

## 5.4.5 Software triggers

The software Trigger Interfaces are provided to enable the software to interact with triggering. They can be used for synchronizing the DMAC operation with the software, or during development, it can be a substitute for hardware trigger events.

### 5.4.5.1 Software Trigger Interface

The software Trigger Interface consists of the following DMA channel register fields.

The software trigger control fields in the CH<x>\_CMD register:

- SRCSWTRIGINREQ
- SRCSWTRIGINTYPE
- DESSWTRIGINREQ
- DESSWTRIGINTYPE
- SWTRIGOUTACK

The software trigger status fields in the CH<x>\_STATUS register:

- STAT\_SRCTRIGINWAIT
- STAT\_DESTRIGINWAIT
- STAT\_TRIGOUTACKWAIT

The software trigger interrupt status fields in the CH<x>\_STATUS register:

- INTR\_SRCTRIGINWAIT
- INTR\_DESTRIGINWAIT
- INTR\_TRIGOUTACKWAIT

The software trigger interrupt enables fields in the CH<x>\_INTREN register:

- INTREN\_SRCTRIGINWAIT
- INTREN\_DESTRIGINWAIT
- INTREN\_TRIGOUTACKWAIT



### 5.4.5.2 Software trigger protocol

The wait status fields (STAT\_\*WAIT) indicate when the DMA channel can accept a software trigger. Software triggers cannot be initiated when the wait status field is zero.

#### Software trigger input protocol

The intended trigger input type must be set in the \*SWTRIGINTYPE field. The trigger input request is initiated by writing 1 into the \*SWTRIGINREQ field. This can be done with the same register Write-Access as the one that sets the type or with a consequent Write-Access.

When the request is initiated, the associated STAT\_\*TRIGINWAIT flag is cleared to indicate that the processing of the request has been begun. The \*SWTRIGINTYPE field becomes read-only to keep the trigger input type stable throughout the trigger event.

When the trigger request is completed, the \*SWTRIGINREQ field is cleared and \*SWTRIGINTYPE field changes to be read/write again.

#### Software trigger output protocol

The STAT\_TRIGOUTACKWAIT field indicates that a trigger output request is active and the software might send an acknowledge to it by writing 1 into SWTRIGOUTACK field.

The STAT\_TRIGOUTACKWAIT status field is cleared when the acknowledge is accepted.

### 5.4.5.3 Software trigger interrupts

The INTR\_\* interrupt status fields are set whenever the corresponding STAT\_\* fields are set and the corresponding interrupt is enabled by INTREN\_\* being set to 1.

The interrupt flags are cleared automatically when the status flags are cleared.

## 5.5 AXI4 stream operation

To offer flexible data manipulation, the DMA-350 can include an external engine in the dataflow. The data read by the AXI5 manager interface can be sent out through the AXI4 stream interface to, for example, a filter unit. The unit sends back the corrected data to the DMAC and the DMAC writes the modified data to memory through the AXI5.

### 5.5.1 Packet boundaries on stream in interface

The stream in interface expects a single packet for the whole DMA command. When the tlast is received from the external engine, it means that it has no more data to receive and the DMA command is finished.

When tlast is received while DESXSIZE or DESYSIZE is greater than 0, the command is finished and no more transfers are to be generated on the destination side when using the continue option after the data with tlast is sent out. This can be considered as a use case for compression, when

an upper limit is set for the destination area but the compression resulted in less data for the destination side than originally expected. The fill settings add the FILLVALUE to the remaining area until the DESXSIZE is reduced to 0 for 1D transfers. For 2D transfers, the fill can only be set for YTYPE as the single-line width is not known and so XTYPE fill results in an error.



The size of last beat must be TRANSIZE aligned, otherwise the AXI5 write side cannot process the data. If not, stream in error is generated and the DMAC job is stopped. TRANSIZE must be carefully chosen, which ultimately depends on the algorithm type running on the external stream engine. If it produces less outgoing data, then the question is how much less it can be. The smallest possible granularity should be the TRANSIZE.

When DESXSIZE and DESYSIZE are reduced to 0 without receiving tlast, the command stops sending out more data on AXI. This is considered a programming fault because the DMAC expected to finish the transfer earlier than it should. This is indicated to SW by asserting an error. The channel still receives but drops incoming stream in transfers until the tlast is received or the interface is reset.

### Stream in flush behavior

The FLUSH output signal is asserted when the DMA command must finish its operation before receiving a tlast on the stream in interface. This shows that the DMAC discards all data received with a TVALID/TREADY handshake while the FLUSH is asserted. The FLUSH is asserted until a TVALID / TREADY handshake is received on the stream in interface where tlast is also asserted. The DMA command cannot conclude until FLUSH is asserted.

## 5.5.2 Packet boundaries on stream out interface

The stream out interface sends a single packet in AXI4 stream terms to the external engine for the whole DMA command. This packet can contain multiple transfers (beats). Since the interface does not have TID, all transfers are within one packet.

## 5.5.3 Stream interworking with other modes

The operation of the stream interface can be enabled in parallel with other features of the DMAC. However, some restrictions apply when using the stream interface for the command.

### 5.5.3.1 Interworking with 1D and 2D types

The 1D and 2D modes the XTYPE and YTYPE settings are constrained when using stream interface. This constriction is because the stream interface cannot use wrap modes and fill in some cases. The following table summarizes the allowed configuration settings:

**Table 5-6: Stream interface for 1D and 2D types**

Mode	XTYPE	YTYPE	Stream allowed	Comment
OFF	Disabled	Disabled Continue Wrap Fill	N/A	No data transfers occur regardless of stream interface enable.
1D	Continue	Disabled	Yes	All stream types can be used for simple 1D transfers. When only stream output is used DESXSIZE is ignored, when only stream input is used SRCXSIZE is ignored.
1D	Fill	Disabled	Yes	When the destination line has more space after tlast is received, the remaining area is filled with FILLVAL.
1D	Wrap	Disabled	No	The stream input data cannot be fetched again so it results in a configuration error.
2D	Continue	Continue	Yes	All stream types can be used for simple 2D transfers. When only stream output is used, DESXSIZE and DESYSIZE are ignored. When only stream input is used, SRCXSIZE and SRCYSIZE are ignored. If the stream input sends an early tlast the destination transfer can be stopped in the middle of the line.
2D	Continue	Fill	Yes	When the destination area has more space after tlast is received, the remaining area is filled with FILLVAL.
2D	Continue	Wrap	No	The stream input data cannot be fetched again so it results in a configuration error.
2D	Wrap	Continue Wrap Fill	No	The stream input data cannot be fetched again so it results in a configuration error.
2D	Fill	Continue Wrap Fill	No	The stream input data does not indicate the end of the line so the DMAC cannot fill the remaining part of the line with data and it results in a configuration error.

### 5.5.3.2 Interworking with Trigger Interface

The stream interface usage has some restrictions on the trigger input interface as the block-based transfers might behave improperly when converting them to stream output transfers.

The whole stream out transfer is one packet, which cannot have strobes in the middle of the packet. Therefore any block, or single beat transfer will stall the interface if it is not aligned to the data width of the stream interface. If the DMAC fetches the remaining data, but it is stored in an internal FIFO, the external stream engine does not have all the data to process until the next block is triggered.

**Table 5-7: Stream interface for triggers**

Mode	SRC/ DESTRIGINTYPE	Stream allowed	Comment
Software Controlled only Command Trigger	000	Yes	SW can initiate the whole command that uses the stream interface in any direction.
HW Controlled Command trigger	100	Yes	HW can initiate the whole command that uses the stream interface in any direction.
Any Flow control trigger	001, 101, 110	No	Stream interface cannot stall or strobe non-data width aligned trigger block size settings.



SRC/DESTRIGINTYPE values are defined in the [CH\\_DESTRIGINCFG](#) section of the [Programmers model](#).

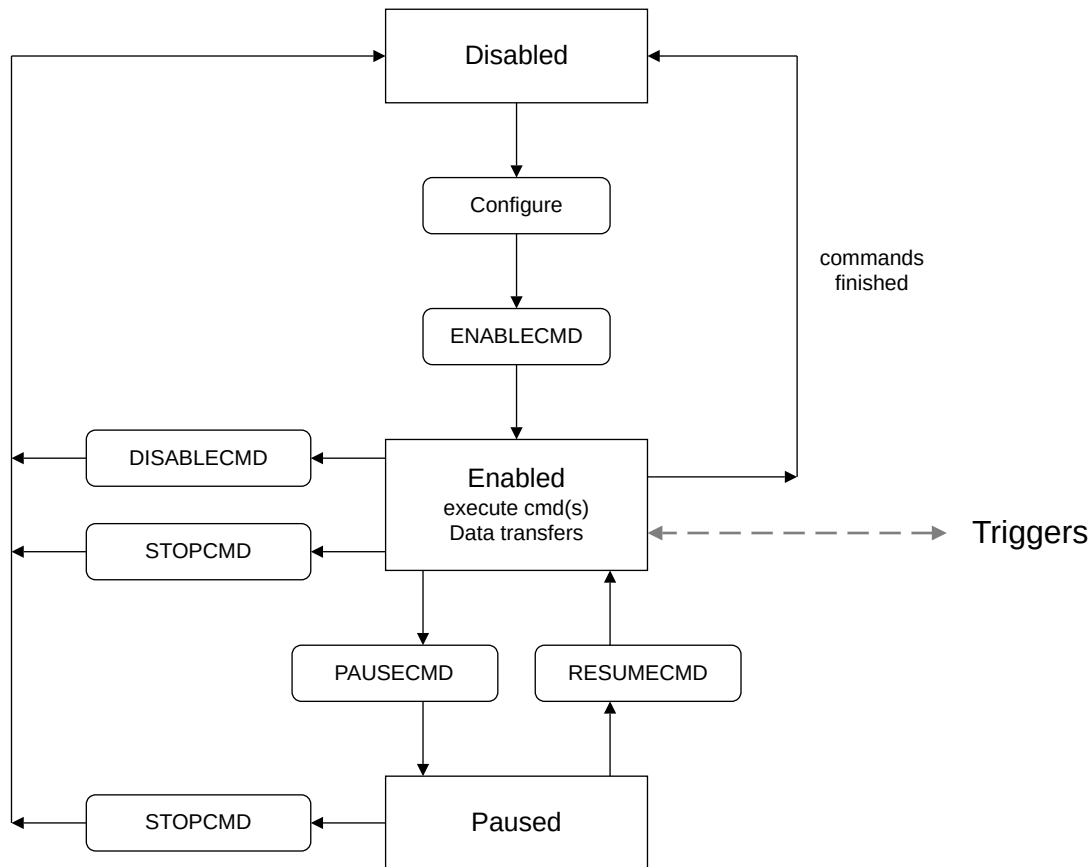
If packet based triggering is required from a peripheral, the SRCXSIZE or DESXSIZE must be set to match the size of a single packet and the command is reloaded for the next packet. This is to replace a large XSIZE setting for multiple packets with using BLOCK triggers where TRIGBLOCKSIZE represents one single packet.

Trigger out signaling can be used without any restrictions when using the stream interface. Trigger loops can only be used to start whole commands on the receiving side as flow control is not supported.

## 5.6 DMA Channel lifecycle

The operation of a DMA channel is shown in the following figure:

**Figure 5-21: DMA channel lifecycle**



### Disabled state

After reset, the channel is in the disabled state. The command to be executed must be set up by configuring the channel registers. The software can modify every writable configuration register in this state.

After the channel is configured, the execution is started by setting the ENABLECMD bit in the CH<x>\_CMD register.

### Enabled state

Execution of the command happens in the Enabled state. When the ENABLECMD bit is set, the ch\_enabled signal is asserted to indicate ongoing command execution. This status signal remains HIGH until the command is completed and no other command linked command is configured, or the channel operation is stopped manually or because of an error event.

In the enabled state, all configuration registers of the channel become HW-controlled ones. The SW is still allowed to read the registers, but the HW is updating their contents throughout the operation of the command. The only registers the SW can adjust during the command operation are the CH<x>\_CMD, CH<x>\_STATUS, and CH<x>\_WRKREGPTR registers that allow the SW to stop or pause the command execution, clear interrupts and inspect internal work register states.

The following size and address registers are continuously updated during the data transfer to show the current state of the transfer:

- CH<x>\_SRCADDR / CH<x>\_SRCADDRHI
- CH<x>\_DESADDR / CH<x>\_DESADDRHI
- CH<x>\_XSIZE / CH<x>\_XSIZEHI
- CH<x>\_YSIZE

The \*ADDR registers point to the next location in the memory where accesses are made. The \*SIZE registers count down from the starting value to 0 and show the remaining number of transfers when read. A counter value cannot explicitly show that the command is finished as 2D or wrap operations might restart the counters or finish earlier.

The command execution ends in the following cases:

- The command is finished successfully.
- An error event (for example, configuration error) causes the command to stop.
- The channel is disabled because of setting the DISABLECMD bit in the CH<x>\_CMD register.
- The channel is stopped because of:
  - Setting the STOPCMD bit in the CH<x>\_CMD register
  - Setting the ALLCHSTOP bit in the NSEC\_CTRL / SEC\_CTRL register (depending on the configured security of the channel)
  - Asserting the allch\_stop\_req\_nonsec / allch\_stop\_req\_sec signal (depending on the configured security of the channel)

The channel returns to the disabled state when any of the previous conditions cause the command to end its execution. The status flags (STAT\_\* fields in the CH<x>\_STATUS register) are set when returning to disabled state. The SW can inspect the status flags to determine whether the execution was successful or an error occurred.

## Paused state

The channel enters the paused state when a pause request is issued. A pause request can be issued by:

- Setting the PAUSECMD bit in the CH<x>\_CMD register (SW pause request).
- Setting the ALLCHPAUSE bit in the NSEC\_CTRL / SEC\_CTRL register (depending on the configured security of the channel).
- Asserting the allch\_pause\_req\_nonsec / allch\_pause\_req\_sec signal (depending on the configured security of the channel).

- Automatic SW pause request when the STAT\_DONE flag is asserted and the channel has been preconfigured to enable done-pause by setting the DONEPAUSEEN bit in the CH<x>\_CTRL register.
- Cross Trigger Interface
- Warm reset

In the paused state, the channel execution is stalled. The bus transactions are stopped and processing of triggers is also halted. The SW can inspect the channel state while being paused, as the address and size registers reflect the current state of the channel. Internal channel registers can also be inspected by setting the CH<x>\_WRKREGPTR and reading the CH<x>\_WRKREGVAL register.

Operation of the channel is continued when the pause request is revoked:

- SW pause request, including done-pause, can be revoked by setting the RESUMECMD bit in the CH<x>\_CMD register.
- All-channel-pause request is revoked by clearing the STAT\_ALLCHPAUSED flag in the NSEC\_STATUS / SEC\_STATUS register.
- HW pause request is revoked by deasserting the allch\_pause\_req\_nonsec / allch\_pause\_req\_sec signal.

The channel can be stopped in the paused state by issuing a stop command.



The channel remains enabled (ch\_enabled asserted) in the paused state, only the execution is halted temporarily. The configuration registers remain read-only when paused.

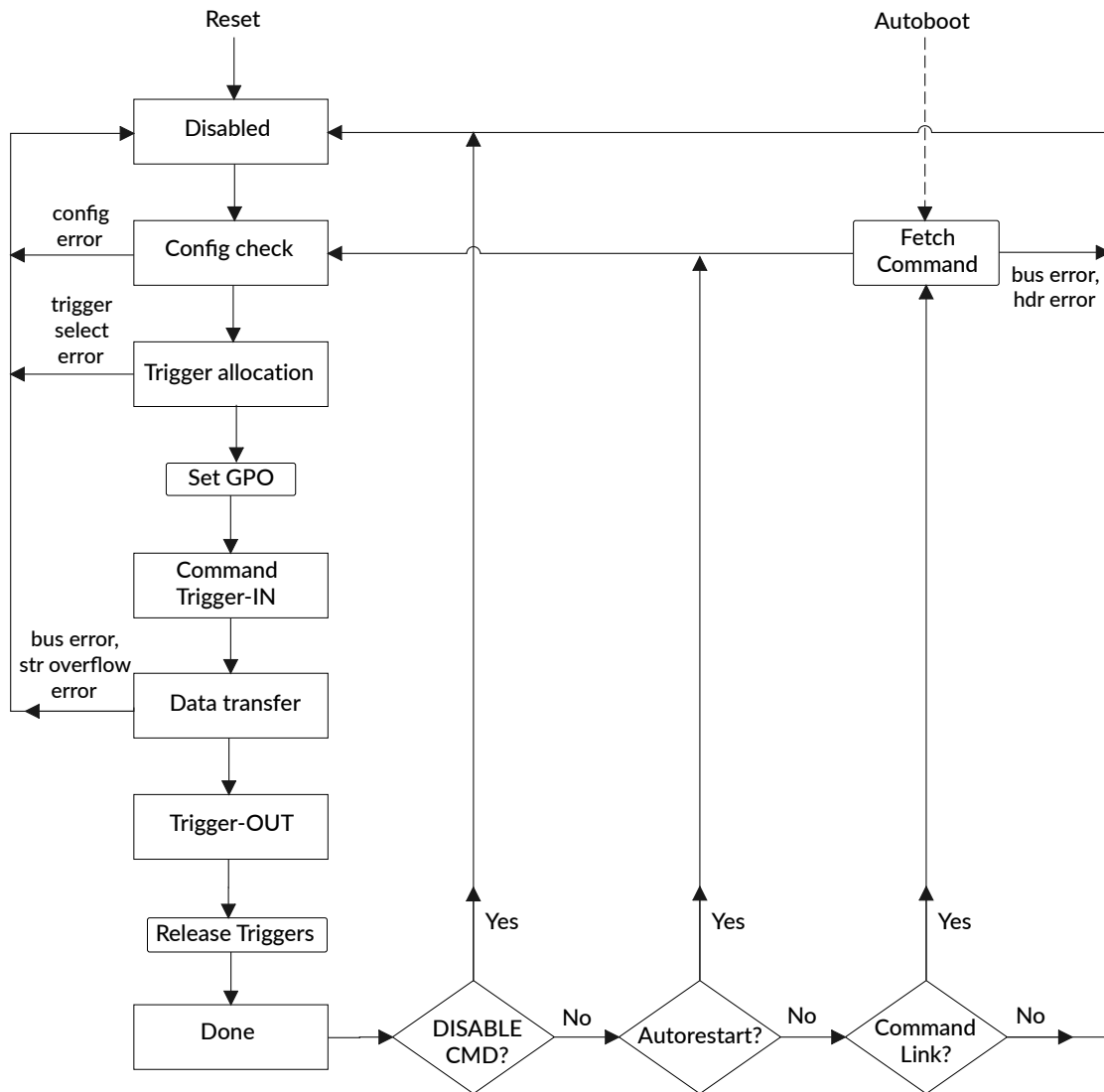
---

### 5.6.1 Command execution states

Each command has various steps that are executed in a fixed order. Some steps can be skipped depending on the configuration of the command.

The following flowchart illustrates the DMA channel command execution states.

**Figure 5-22: DMA channel command states**



### Configuration checking state

This state verifies the configuration of the channel. If an invalid register value or conflicting settings are detected, the corresponding error status flags are set, and the channel operation is stopped.

### Trigger allocation state

If an external HW trigger or internal triggering is configured, the selected trigger port or channel connection is made in this state. If the connection could not be established, a trigger selection error is indicated, and the channel operation is stopped. If only SW triggering, or no triggering is configured, this step is ignored. The GPO output is set to the new value (if configured) before proceeding to the next state.



## Command trigger state

The command trigger state handles the incoming command trigger requests when a command trigger input is configured. The channel execution is halted until a command trigger request is received on all trigger inputs that are in command trigger mode. The trigger acknowledge is asserted only when all request is received. If no command trigger is configured, this step is ignored.

## Data transfer state

The data transfers occur in this state. Flow-control trigger are processed only in this state.

## Trigger output state

As the last step of a command, the trigger output request is asserted, if output triggering is configured. Operation halts until an acknowledgment is received. If no output triggering is configured, this step is skipped.

## Done state

Reaching this state indicates that the command has completed successfully. The trigger connections are released, and the STAT\_DONE flag is set depending on the configured DONE\_TYPE field.

If a DISABLECMD has been issued up to this point, the channel execution is finished and the STAT\_DISABLED flag is set.

If CMDRESTARTCNT is nonzero or CMDRESTARTINFEN is set, then the current command is restarted automatically. The CMDRESTARTCNT counter is decremented by 1 if it is nonzero.

If no automatic restart of the current command is configured, then the LINKADDREN flag is checked. If set, the command link feature is enabled, and the next command is fetched using the LINKADDR pointer.



Each command in a command link can choose input and output triggers. The consecutive commands do not necessarily select the same triggers. Regardless, the triggers are reconnected at each command during the execution of the command-chain.

When a trigger input request is sent to a DMA channel after the channel has received the last trigger it was expecting, the trigger request remains pending. The next command that is configured to use the same external trigger port or internal channel receives it. If the new linked command uses this same trigger, the next command can respond to it and no trigger event is lost.

---

If the auto restart and command link are both disabled, the channel operation is finished, and the channel returns to disabled state.

Otherwise, the DONEPAUSEEN flag is checked. If set, the channel execution is paused until a RESUMECMD is received.

## 5.6.2 Automatic restart of commands

The auto restart feature improves the efficiency when a single command must be repeated many times in a loop, by eliminating unnecessary reloading of the same command.

Auto restart can be enabled by setting the CMDRESTARTINFEN or CMDRESTARTCNT fields to a nonzero value. The number of iterations depends on the settings of these fields:

- CMDRESTARTINFEN is set: The automatic restarting keeps occurring in an infinite loop until the channel is disabled or stopped.
- CMDRESTARTINFEN is not set: The automatic restarting occurs for the number of times that is configured in CMDRESTARTCNT field. If it is configured to zero, the auto restart feature is not enabled.

If the auto restart feature is enabled, the destination and source addresses and the transfer size settings are reloaded with their original value depending on the setting of the REGRELOADTYPE field. If an address is not reloaded, the starting address of the next iteration of command execution is defined based on the operation type.

## 5.6.3 Error handling

There are four types of errors that can occur during DMAC operation:

- AXI5 bus error during operation
- Configuration error
- Trigger selection error during trigger allocation
- AXI-Stream interface error during operation

Any of these errors causes the DMA channel to stop its operation. The error status flag, STAT\_ERR, is set in the CH<x>\_STATUS register when the channel state transitions to DISABLED (ch\_enabled = 1'b0).

If the related interrupt enable flag is set, INTREN\_ERR, the interrupt flag is also asserted, INT\_ERR.

The cause of the error is indicated in the CH<x>\_ERRINFO register which contains valid information when STAT\_ERR is asserted.

The following table lists the possible errors indicated through the CH<x>\_ERRINFO register.

**Table 5-8: DMA-350 channel errors**

Error type	CH<x>_ERRINFO field	Description
Bus errors	BUSERR	Error occurred on any of the AXI managers or AXI-Stream interfaces during data transfer or command link fetch. The exact cause is indicated in the AXI*ERR fields, see the table below.
Configuration errors	CFGERR	The command that is about to be executed has an invalid configuration in the registers, or invalid command link header was received. See LINKHDRERR, REGVALERR, and CFGCONFLERR fields for the exact cause and the table detailing the error information below.

Error type	CH<x>_ERRINFO field	Description
Trigger in selection errors	SRCTRIGINSELERR, DESTRIGINSELERR	A trigger input port selected by the channel is already in use by another enabled channel or by the trigger matrix.
Trigger out selection errors	TRIGOUTSELERR	A trigger output port selected by the channel is already in use by another enabled channel or by the trigger matrix.
Stream overflow error	STREAMERR	The stream interface encountered an error. See STR*ERR fields for the exact cause of the error.

The following ERRINFO fields give additional information on the reasons for the errors indicated in the CH<x>\_ERRINFO register. Bit offsets are relative to CH<x>\_ERRINFO register.

**Table 5-9: DMA-350 channel error reasons - ERRINFO fields**

Bits	Name	Description
[31:27]	-	Reserved, <b>RAZ/WI</b>
[26]	CFGCONFLERR	Configuration error because there are conflicting settings in the configuration registers.
[25]	REGVALERR	Configuration error because one of the configuration registers is set to an illegal value.
[24]	LINKHDRERR	Configuration error because an invalid command-link header was read.
[23:22]	-	Reserved, <b>RAZ/WI</b>
[21]	STRINEARLYTERM	Stream interface error because the AXI-Stream In interface finished earlier than AXI-Stream Out.
[20]	STRINOVERRUN	Stream interface error because AXI5-Write interface finished while AXI-Stream In is still busy or the Write-FIFO is not empty (and Stream In was used).
[19]	STRINTSTRBERR	Stream interface error because of invalid tstrb detected on the AXI-Stream In interface.
[18]	AXIRDPOISERR	Bus error because of AXI read poison detected.
[17]	AXIWRRESPERR	Bus error because of AXI write error response.
[16]	AXIRDRESPERR	Bus error because of AXI read error response.

## AXI5 bus errors

When a bus error is detected, the current DMA command stops immediately and the data associated with the bus error is discarded. All outstanding transactions are completed, but their data is also discarded. If BUSERR flag is set, at least one of AXIRDRESPERR, AXIWRRESPERR or AXIRDPOISERR is also set in the CH<x>\_ERRINFO register to indicate the source of the bus error condition.

## Configuration errors

Configuration errors are checked for before data transfer execution. No bus transfers are initiated when a configuration error is detected.

The three possible sources of configuration errors are:

- Invalid command link header (LINKHDRERR) - all bits are Read-As-Zero.
- Illegal field value set in a configuration register (REGVALERR).
- Incompatible settings are present in the configuration registers (CFGCONFLERR).

### Illegal field value errors

The causes of illegal field value error (REGVALERR) are:

- TRANSIZE field is greater than bus data width.
- Trigger selector (SRCTRIGINSEL, DESTRIGINSEL, TRIGOUTSEL) value is out of valid range.

### Incompatible setting errors

The reasons for conflicting configuration settings (CFGCONFLERR) are:

- The trigger-in configuration conflicts with the configured transfer mode (XTYPE / YTYPE or stream settings).
- Templated transfer is configured in an unsupported transfer mode.
- AXI-Stream interface is enabled with an unsupported transfer mode.

See [Configuration](#) for detailed conditions on incompatible settings.

### Trigger in/out selection errors

These errors could occur during the trigger allocation state when the selected trigger port or channel is not available at the clock cycle of selection. The command execution stops after the failed trigger allocation, therefore no bus transfers are initiated. The possible reasons for unavailable trigger port or channel are:

- The selected trigger port is already in use by another channel when external trigger is configured.
- The selected channel is already connected to another channel when internal trigger is configured.
- The same trigger is selected for both source and destination sides.

### AXI-Stream interface error

When an error condition is detected related to the Stream interfaces, the ongoing DMA command is stopped and the STREAMERR field is set. If STREAMERR flag is set, at least one of STRINTSTRBERR, STRINOVERRUN or STRINEARLYTERM is also set in the CH<x>\_ERRINFO register to indicate the source of the Stream interface error condition.

The possible causes for Stream interface error are:

- Stream In tstrb error (STRINTSTRBERR). This indicates that an invalid tstrb attribute is detected.
- Stream overrun error (STRINOVERRUN). This occurs when no more AXI writes are left according to DESXSIZE, but the incoming AXI-Stream has not been terminated with a tlast signal. The channel operation is stopped if this condition is detected, and the signal str\_in\_<x>\_flush is asserted on the AXI Stream-IN interface to indicate discarding of the remaining inbound data.
- Early Stream In termination (STRINEARLYTERM). If the Stream In interface finishes earlier than the Stream Out interface, this flag is set.

## 5.6.4 Command execution status reporting

The software can read the contents of X, Y size, and address registers after the DMA command execution.

The contents of these registers depend on the manner of DMA command stop:

DONE status reached - command execution was completed:

- Size registers are 0, no more transactions to be completed on either read or write side.
- Address registers contain the next address as if there were subsequent DMAC operations of the same type as the completed command.

STOPPED/PAUSED status - command execution was stopped or paused before execution completed:

- Size registers contain the actual coordinate of the current transaction that was not completed.
- Address registers contain the address of the next transaction that was not completed.

In STOPPED/PAUSED state, address and size register values are only approximations of the actual command execution status, and might not contain the exact position. They serve as a hint about where approximately the command execution was halted.

If pausing occurs during command link execution, the registers read back might contain an incoherent command, because not all registers might be updated when the DMA enters PAUSED state.

### Address register at the end of command

At the end of command, address registers contain the address of the next DMAC transaction of the same type that was not completed. If the command is completed, the next address points to the address that would have been used for the transaction if the command contained more transactions.

Major cases are covered in the following tables for clarification.

**Table 5-10: Read operations**

Read	DMA command last line end	DMA command not last line end	DMA command mid-line
1D without wrap	Incr_addr	N/A	Incr_addr
1D with wrap	Cur_line_start_addr	N/A	Incr_addr
2D with Y not wrap	Incr_line_start_addr	Incr_line_start_addr	Incr_addr
2D with Y wrap	First_line_start_addr	Incr_line_start_addr	Incr_addr

When X type is wrap, 1D with wrap or 2D with Y wrap, there are two possible line ends for read:

- When read line is ended only, next address is always the start address of the same line.
- When write line is ended, with or without read line ending at the same time, the next address is determined by the read operations in the preceding table.

Because 1D commands consist of only one line, it is also deemed as the last line.

**Table 5-11: Write operations**

Write	DMA command last line end	DMA command not last line end	DMA command mid-line
1D without wrap	Incr_addr	N/A	Incr_addr
1D with wrap	Incr_addr	N/A	Incr_addr
2D with Y not wrap	Incr_line_start_addr	Incr_line_start_addr	Incr_addr
2D with Y wrap	Incr_line_start_addr	Incr_line_start_addr	Incr_addr

Command line definitions:

**Incr\_addr**

Increment the address of the last DMAC transaction based on increment or template configurations and configured TRANSIZE.

**Incr\_line\_start\_addr**

Increment the starting address of current line based on Y stride and TRANSIZE configurations. Address points to the first byte of this line.

**Cur\_line\_start\_addr**

Address points to starting byte of current line.

**First\_line\_start\_addr**

Address points to the first byte of the first line in the current 2D DMA command.

**DMA command last line end**

Last transaction of the command executed by DMA in this direction is the last transaction of the last line.

**DMA command not last line end**

Last transaction of the command executed by DMA in this direction is the last transaction of a line.

**DMA command mid-line**

Last transaction of the command executed by DMA in this direction is the not last transaction of a line.

**Examples**

1D command without wrap for read/write:

- X type: continue/fill
- Y type: disabled
- X size: 5
- Y size: -
- Transize: 1 byte
- Start address: 0x0

If last read/write transaction address of command: 0x4, (5. byte), both addresses point to 0x5 when software reads their values back.

#### 1D command with wrap for read/write:

- X type: wrap
- Y type: disabled
- Source X size: 5
- Dest. X size: 7
- Y size: -
- Transize: 1 byte
- Start address: 0x0
  - If last read transaction address of command: 0x4, (5. read byte), read address points to 0x0 when software reads its value back.
  - If last read transaction address of command: 0x1, (2. read byte), read address points to 0x2 when software reads its value back.
  - If last read transaction address of command: 0x1, (7. read byte), read address points to 0x0 when software reads its value back (destination size reached for wrap).
  - If last write transaction address of command: 0x6, (7. write byte), write address points to 0x7 when software reads its value back.

#### 2D command without wrap for read/write:

- X type: continue/fill
- Y type: continue/fill
- X size: 5
- Y size: 3
- Y stride: 0x10
- Transize: 1 byte
- Start address: 0x0
  - If last read/write transaction address of command: 0x24, (3. line, 5. byte), read/write addresses point to 0x30 when software reads their values back.
  - If last read/write transaction address of command: 0x14, (2. line, 5. byte), read/write addresses point to 0x20 when software reads their values back.
  - If last read/write transaction address of command: 0x13, (2. line, 4. byte), read/write addresses point to 0x14 when software reads their values back.

#### 2D command with wrap for read/write:

- X type: continue/fill
- Y type: wrap
- Source X size: 5
- Y size: 3
- Y stride: 0x10

- Transize: 1 byte
- Start address: 0x0
  - If last read/write transaction address of command: 0x24, (3. line, 5. byte), read address points to 0x0 and write address points to 0x30 when software reads their values back.
  - If last read/write transaction address of command: 0x14, (2. line, 5. byte), read/write addresses point to 0x20 when software reads their values back.
  - If last read/write transaction address of command: 0x13, (2. line, 4. byte), read/write addresses point to 0x14 when software reads their values back.

2D command with wrap for read/write:

- X type: wrap
- Y type: wrap
- Source X size: 5
- Dest. X size: 7
- Y size: 3
- Y stride: 0x10
- Transize: 1 byte
- Start address: 0x0
  - If last read transaction address of command: 0x24, (3. line, 5. read byte), read address points to 0x20 when software reads its value back.
  - If last read transaction address of command: 0x21, (3. line, 2. read byte), read address points to 0x22 when software reads its value back.
  - If last read transaction address of command: 0x21, (3. line, 7. read byte), read address points to 0x0 when software reads its value back.
  - If last read transaction address of command: 0x11, (2. line, 2. read byte), read address points to 0x12 when software reads its value back.
  - If last read transaction address of command: 0x11, (2. line, 7. read byte), read address points to 0x20 when software reads its value back.
- Write addresses work the same way as in 2D wrap with not 1D wrap case.

### Addresses after empty commands

Read/write addresses remain unchanged, if their respective direction was not involved in command execution.

For example, when write is active but read is inactive in a DMA command, the read address does not change but the write address does change according to the behavior described above.

### Size registers at the end of command

Size registers contain the coordinates of last unprocessed transaction within the current line. Both size registers are 0 when the DMA command completes.



When a stop or bus error interrupts the DMA command, 1D commands only have X registers and they contain the remaining unsent DMAC transactions in TRANSIZE.

2D commands have X and Y size registers, and they contain the X and Y coordinates of the next unsent transaction:

- When the DMA command is mid-line, the X and Y sizes contain the unsent coordinates of the next unsent data.
- When the DMA command has completed the current line, depending on the timing of the interrupting event, both following coordinates are possible. These coordinates are equivalent, as they indicate the same amount of data that was not transmitted from the DMA command.
  - Y: current line number, X:0
  - Y: next line number, X: total X-size of the line

Additionally, X and Y sizes can be optimized for certain Y and X configurations at the beginning of the DMA command:

- If X type is WRAP or FILL and destination X-size < source X-size, source X-size is set to destination X-size.
- If X type is WRAP or FILL and Y type is FILL or CONTINUE, and source Y-size > destination Y-size, source Y-size is set to destination X-size.

These optimizations help reduce unnecessary read operations that can already be predicted from the DMA command configuration.

### Size after empty commands

If there is an empty command execution, sizes do not change at the end of DMA command. If either side is involved in command execution while the other remains idle, size registers on the idle side become zero when the command is completed.

## 5.7 Command linking

The command linking feature enables the DMA channels to execute more operations by automatically loading the next commands from the system memory to its configuration registers. This feature makes the DMAC versatile in combining multiple commands in a DMAC transaction.

Each command can define new transfer parameters, triggering behavior, and interrupt settings. This provides great flexibility in the possible usage of the DMA unit and makes it possible to implement complex data transfer tasks by properly designing command chains. The commands are defined by descriptors stored in memory. The channel fetches the descriptors by using the pointer address stored in the link register (LINKADDR). The first command must be set directly in the channel registers to start the linked command chain. If required, the first command can be an empty command which only points to a memory location with the first non-empty command.

Certain commands might only change some of the configuration registers, for example, the destination address or the number of transfers, but keep all other registers the same for the next command. To do this, the command descriptors in the memory have a header part that specifies

what registers to update in the channel register bank. This reduces the number of bus transfers executed during command fetching.

The command link chain finishes when a command in the chain does not have the LINKADDREN bit set. This can be achieved by not setting this LINKADDR register bit in the descriptor of the last command in the chain.

The new command is fetched after the DONE state when a command is finished, see [Channel lifecycle](#). The register values are read and updated according to the header word.

Execution of a command chain can be stopped by setting the DISABLECMD bit in the channel command register (CH<x>\_CMD). It can be used to stop an endless looped command link. The disable stops the command chain cleanly; the command being executed currently is finished but the next one is not loaded.

When a command in a command link is paused it behaves just like pausing a single command, see [Stop and pause control](#). After resuming, the execution of the command is resumed and the command link is continued.

When a command in the command link stopped, the remaining part of the command and the remaining part of the command link are canceled. The stop waits for all the outstanding responses from read and write transactions but it tries to finish the channel operation as soon as possible. For more information, see [Stop and pause control](#).

### 5.7.1 Command structure

The commands are stored in the system memory within a linked list data structure. Each command has information on what DMA channel configuration registers must be updated and a pointer to the next command.

The pointer to the next command is stored in the CH<x>\_LINKADDR / CH<x>\_LINKADDRHI configuration registers. The registers can be updated by the next command as any other channel configuration register.

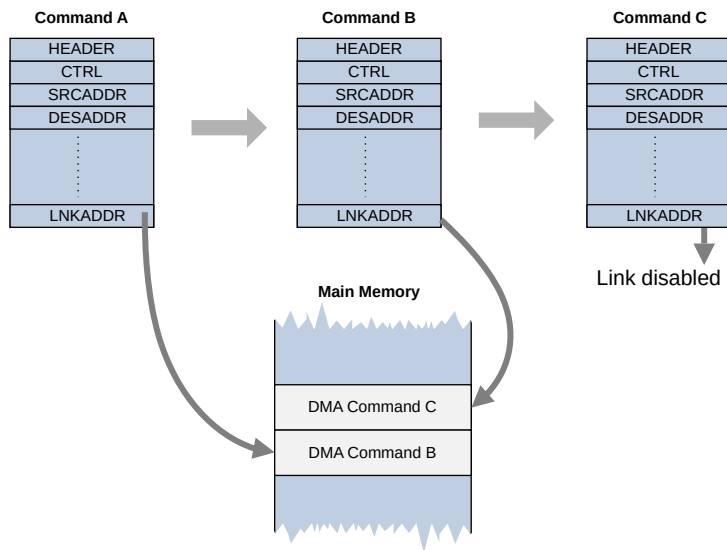
The command chain is terminated by disabling the LINKADDREN flag (the 0<sup>th</sup> bit) in the CH<x>\_LINKADDR register. All other commands must have this flag enabled to indicate a valid link address.

The first command of the command chain must be set manually in the channel configuration registers. The first command can be empty and only link to the first real command in the chain.

#### Linked command chain structure

The following figure illustrates a command chain data structure:

**Figure 5-23: Command linking operation**



## Command descriptors

The commands descriptors are encoded within a continuous array of 32-bit words in the memory.

The first 32-bit word of each command serves as a header that contains a bitmap to indicate what registers require updating. The header also has a special flag, REGCLEAR, that instructs the DMA channel to clear all the configuration registers that are modifiable by command link before updating them with the new values of the command. If this flag is not set, the configuration registers not updated by the command are kept at their previous settings.

The following table specifies the header word of the commands:

**Table 5-12: Command link header**

Bit	Name	Description
0 (LSB)	REGCLEAR	Clear registers before updating with new values
1	Reserved	Reserved for future use, must be zero
2	INTREN	Update the CH<x>_INTREN register
3	CTRL	Update the CH<x>_CTRL register
4	SRCADDR	Update the CH<x>_SRCADDR register
5	SRCADDRHI	Update the CH<x>_SRCADDRHI register (when 40-bit addressing is used)
6	DESADDR	Update the CH<x>_DESADDR register
7	DESADDRHI	Update the CH<x>_DESADDRHI register (when 40-bit addressing is used)
8	XSIZE	Update the CH<x>_XSIZE register
9	XSIZEHI	Update the CH<x>_XSIZEHI register
10	SRCTRANSCFG	Update the CH<x>_SRCTRANSCFG register
11	DESTRANSCFG	Update the CH<x>_DESTRANSCFG register
12	XADDRINC	Update the CH<x>_XADDRINC register

Bit	Name	Description
13	YADDRSTRIDE	Update the CH<x>_YADDRSTRIDE register
14	FILLVAL	Update the CH<x>_FILLVAL register
15	YSIZE	Update the CH<x>_YSIZE register
16	TMPLTCFG	Update the CH<x>_TMPLTCFG register
17	SRCTMPLT	Update the CH<x>_SRCTMPLT register
18	DESTMPLT	Update the CH<x>_DESTMPLT register
19	SRCTRIGINCFG	Update the CH<x>_SRCTRIGINCFG register
20	DESTRIGINCFG	Update the CH<x>_DESTRIGINCFG register
21	TRIGOUTCFG	Update the CH<x>_TRIGOUTCFG register
22	GPOENO	Update the CH<x>_GPOENO register
23	Reserved	Reserved for future use, must be zero
24	GPOVAL0	Update the CH<x>_GPOVAL0 register
25	Reserved	Reserved for future use, must be zero
26	STREAMINTCFG	Update the CH<x>_STREAMINTCFG register
27	Reserved	Reserved for future use, must be zero
28	LINKATTR	Update the CH<x>_LINKATTR register
29	AUTOCFG	Update the CH<x>_AUTOCFG register
30	LINKADDR	Update the CH<x>_LINKADDR register
31 (MSB)	LINKADDRHI	Update the CH<x>_LINKADDRHI register (when 40-bit addressing is used)



At least one bit of the header word must be set to 1, otherwise a configuration error is reported and execution of the command stops. For more information, see [Error handling](#).

The consecutive words in a command must contain the new values of the configuration registers that were indicated for updating in the header. They must be in the same order as the flags in the header starting from LSB to MSB.

### Example descriptors for linked commands

An example of three descriptors in the memory when 32-bit addressing is used:

**Table 5-13: Command link example memory map, HEADER\_0**

Descriptor offset	Field	Value
0x000	HEADER_0	0x0000_0D5D
0x004	INTREN	Any
0x008	CTRL	Simple 32-bit wide 1D
0x00C	SRCADDR	Any
0x010	DESADDR	Any
0x014	XSIZE	Any
0x018	SRCTRANSCFG	Any

Descriptor offset	Field	Value
0x01C	DESTRANSCFG	Any

**Table 5-14: Command link example memory map, HEADER\_1**

Descriptor Offset	Field	Value
0x020	HEADER_1	0x4000_0158
0x024	CTRL	Simple 8-bit wide 1D
0x028	SRCADDR	Any
0x02C	DESADDR	Any
0x030	XSIZE	Any
0x034	LINKADDR	Descriptor offset + 0x038 + 1 (ENABLE BIT)

**Table 5-15: Command link example memory map, HEADER\_2**

Descriptor Offset	Field	Value
0x038	HEADER_2	0x4000_0140
0x03C	DESADDR	Any
0x040	XSIZE	Any
0x044	LINKADDR	0x000

The Any value means that the SW can set it to whatever the command requires.

The HEADER\_0 defines a command that sets up a complete 1D transfer with all registers necessary. This is because the header contained the REGCLEAR bit set which clears all previous settings from the channel registers. It also terminates that command chain because the REGCLEAR command cleared the LINKADDR as well.

Another chained command is stored in the memory defined by HEADER\_1, independent from the first one. It sets a new CTRL register content, updates SRCADDR and DESADDR values and sets the XSIZE to define the length of the command. The other register settings remain the same as the first command in this command chain. It also updates the LINKADDR register and links the command to the next command defined by HEADER\_2. When the command defined by HEADER\_1 is finished, the next descriptor at HEADER\_2 is read. This only changes the DESADDR and the XSIZE compared to the previous command. This might be used to copy the trailing source data to a different location. This is the final command in the chain, so it terminates the link by setting the LINKADDR register to 0 that clears the LINKADDREN enable bit.

## 5.7.2 Loading commands

The command linking feature loads commands from the system memory using the same AXI bus manager interfaces that are used for data transferring.

### Command descriptor fetch attributes

The same AXI ID is used by the channels for command fetching as for the data payloads.

The command is fetched from the memory with AXI burst reads with TRANSIZE parameter set as the value of the DATA\_WIDTH parameter and the \*MAXBURSTLEN field is set to the maximum value.

The command link read uses the security and privilege state of the channel regardless of the transfer properties set in the CH<x>\_SRCTRANSCFG / CH<x>\_DESTRANSCFG registers. As a result, a Non-secure channel can only load Non-secure commands and a Secure channel can only load Secure commands.

Command link reads are distinguished on the AXI bus from data transfers with arprot\_m<x>[2] set to 1 (instruction access). An extra User signal, arcmdlink\_m<x>, is also added on the AR-channels to indicate that the read belongs to a command link fetch.

### Updating configuration registers

The header word is the first word that is read as part of the command link fetch. The consecutive reads contain the new values of the configuration registers that are updated. The register values are updated in the order as they are received.

## 5.7.3 Automatic boot feature

The DMA unit implements an automatic boot feature called autoboot to speed up the bootup of the system. The autoboot can load the first DMA command into channel 0 and start executing it. Autoboot is implemented with the command-link feature and it can be configured with dedicated input signals.

The boot command or commands must be prepared in memory, which may contain any DMAC configuration register value supported by command linking. This provides the possibility to set up an initial single memory-to-memory copy command or a chain of multiple commands that are executed automatically at start-up.

### Enabling the autoboot feature

The address of the boot command and some additional attributes must be configured as input signals, and the boot enable signal must be set HIGH to enable the autoboot feature. To execute an autoboot, the following signals must be set before deactivating the device reset. They must also be stable until the fetching of the boot command is started.

- boot\_en = 1'b1
- boot\_addr = <address of boot command descriptor>
- boot\_memattr = <AXI memory attributes to be used during fetching the boot command descriptor>
- boot\_shareattr = <AXI memory Shareability attributes to be used during fetching the boot command descriptor>

When Security Extension is used, DMA channel 0 starts as Secure, therefore the boot address must point to a Secure address location.

## Autoboot process

The autoboot process is initiated after the DMA-350 is released from reset and both LPI channels have entered their active state. The input signals are sampled at this point, and the process begins with writing the preconfigured boot address and attributes to Channel 0 configuration registers as an initial, empty linked-command.

After this process, the boot command referenced by `boot_addr` is fetched and gets written to Channel 0 configuration registers. Finally, the DMAC executes the boot command.

Channel 0 status signals can be used to track the boot process. For example, `ch_enabled[0]` is asserted throughout the boot command like any other DMA command executed on Channel 0.

## 5.8 Arbitration

Channel arbitration is required when multiple channels want to use a single bus interface to send transfers.

Each channel has a register to set the channel's priority (`CH_CTRL.CHPRIO`). The priority register defined in the architecture is connected to the QoS bits on AXI but it also affects the arbitration scheme used for that channel. The SW is responsible for adjusting the priorities of the channels properly as the DMA unit does not know which channel operation is more time critical than the other. The SW can only change the arbitration scheme when a channel is stopped or is in IDLE. The arbitration scheme is simple to allow the SW to adjust it flexibly.

### 5.8.1 Arbitration requests

The channels can generate requests for arbitration when the following conditions occur.

If there are arbitration requests for read:

- Channel FSM prepared the next address to read with all control information.
- The data FIFO has enough room to store the complete read response.

If there are arbitration requests for write:

- Channel FSM prepared the next address to write with all control information.
- The data FIFO has a complete write burst to be sent out.

This means that channel writes can only start when a full burst is read into the FIFO first. This avoids having circular dependency between reads and writes because writes do not have to wait for any incoming read data.

## 5.8.2 Arbitration scheme

This section describes the *Least-Recently Granted* (LRG) arbitration algorithm.

### LRG Arbitration algorithm

The BIU implements an LRG arbitration algorithm for selecting which DMA channel request to serve. The LRG algorithm's input parameters are priority settings indicated by AXI `arqos` and `awqos` attributes in the requests. The priorities are set in configuration registers by SW and can only be changed when the channel is in IDLE.

The arbitration policy is separated to two layers depending on the priority setting of the channel:

- Priority-based arbitration layer that arbitrates between channels of different priority levels.
- Round-robin scheme arbitration layer that arbitrates between channels with the same priority level, based on which of them was least recently granted.

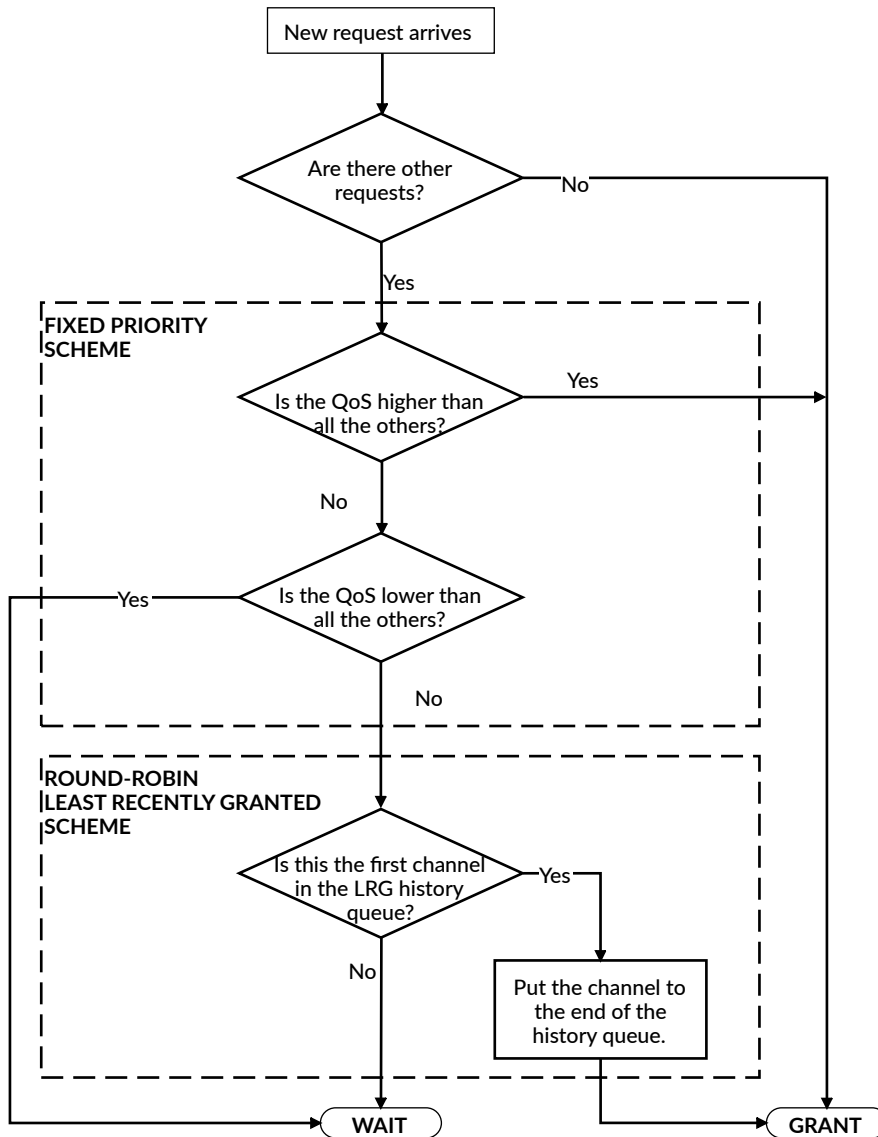
The base layer is a fixed arbitration scheme if the priority settings are different per channel. When multiple requests have different priorities, the single request with the highest priority wins the arbitration. When multiple channels have the highest priority a round robin scheme, least-recently granted (LRG) is applied resulting in a fair share of the bus interface. When all channels are using the round robin scheme, all channels get arbitrated in every round if they had a request since arbitration happens after every burst. When one channel must have higher priority than the others, it can be escalated to the fixed priority layer by setting the priority value above the others. This can lead to starvation of low-priority requests but it ensures a deterministic execution of high priority tasks. A channel using the fixed priority scheme has a higher priority than the round robin scheme so it gets arbitrated first until it runs out of requests. This scheme keeps the flexibility for the SW to set any kind of priority policy in the system. This also allows for time-critical tasks to happen and channels with the same priority use the bus in an even share.

The round robin scheme saves the history of the granted requests. The scheme sorts the channels by descending order of the time of grant, that is from the least recently granted to the most recently granted. When multiple channels have higher priority, the round robin scheme only considers the channels with the highest priority level and selects the least recently granted channel based on the serving history. The same applies when all requests have been served on the higher priority level and the same channels have new requests on a lower priority level which has more outstanding requests. That is, the round robin scheme considers all the channels with the same (highest) priority level and selects the least recently granted channel.

The following figure describes the LRG algorithm in detail:



**Figure 5-24: LRG arbitration algorithm**



In some corner cases, starvation can occur for lower priority channels when high priority requests and low-priority requests are interleaved. For example, this starvation can be caused by an infinitely looped command linked list where the SW frequently enables a higher priority channel and takes all the timeslots from lower priority channels. Reads for command linking also happen with the same priority and same arbitration scheme as the other data-related reads. These cases can be avoided by managing the risks of starvation in SW when using high priority channels together with low-priority ones. The DMAC does not implement fair share algorithms to avoid these situations and it is up to the SW designer to take this into consideration. In a generic approach, free timeslots occur when the high priority channels wait for trigger, wait for responses to arrive, get reconfigured over APB4 or read a new command from the linked list. These free timeslots could be used to enable the low-priority channels to progress in their operation.



Infinitely looped command link lists can result in starvation of lower priority levels.

---

## 5.9 DMAC power management and DMAC control

This section describes the DMA-350 power management and configuration.

### 5.9.1 Power management

The DMA-350 has separate power and clock control management systems.

#### 5.9.1.1 Power P-Channel

The DMA-350 is using one full LPI P-Channel for power management. The purpose of this feature is to enable lower power states (removing power) to reduce power consumption while not in active use. The power P-Channel is used to control the power state of the DMAC logic.

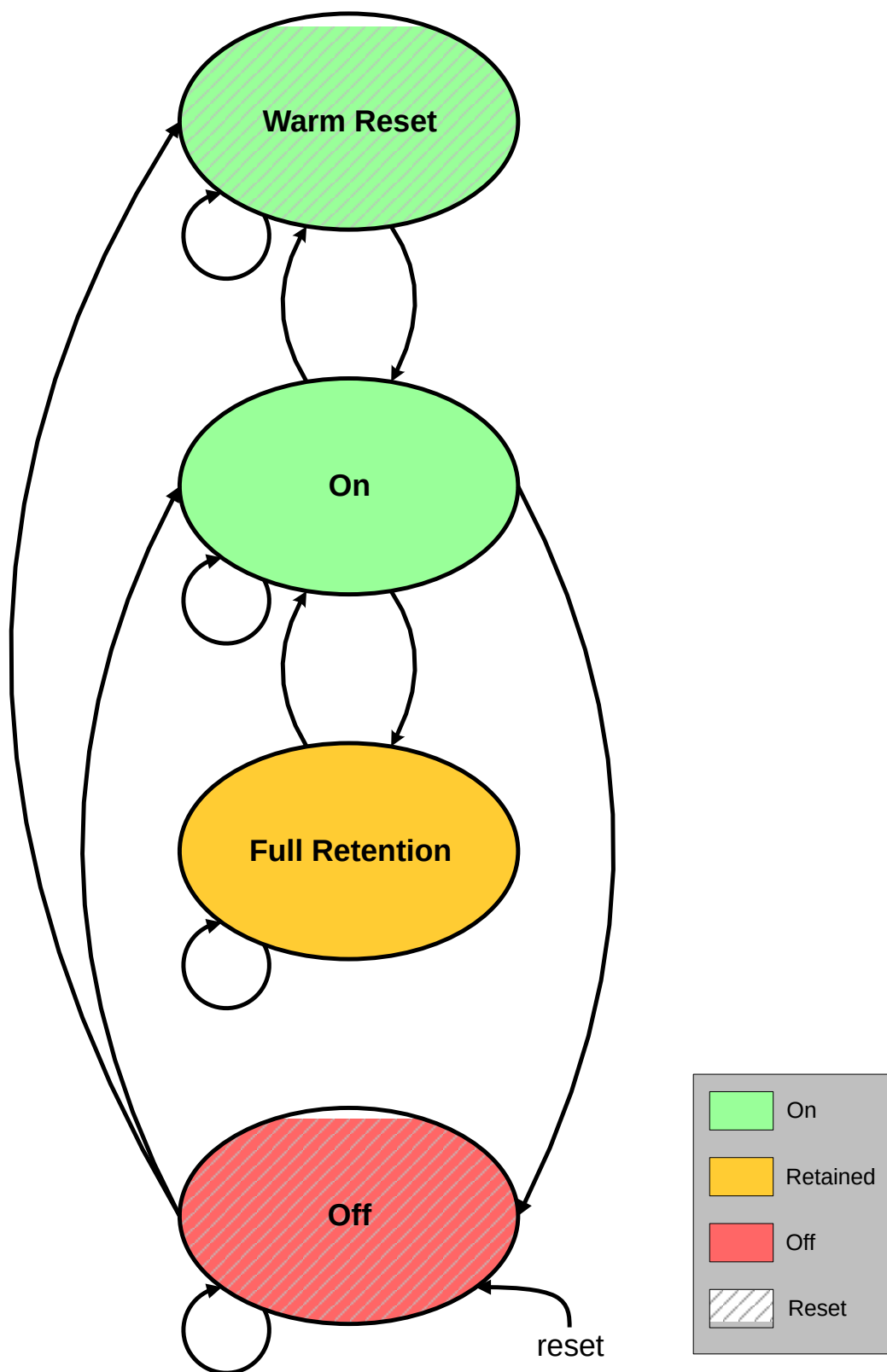
The following four power states can be requested over the P-Channel interface, any other state requests are denied:

- On
- Warm reset mode
- Full retention mode
- Off

The DMAC has a simple power management scenario with the P-Channel interface. When the DMAC is active, power quiescence requests are denied and the operation simply continues. Warm reset mode request pauses all ongoing channel operation and exiting from Warm reset mode to On resumes the channel operation. When the DMAC is actively waiting for an event, the quiescence request to Full retention mode state is accepted but Off state is denied. When the DMAC is inactive, request to Off state is accepted. Before entering Off state, FIFOs are emptied and register values are not kept.

The following figure shows the supported power states and transitions between power states.

**Figure 5-25: Supported power states and transitions**



Entering lower power modes (Full retention mode, Off) can be disabled with the DISMINPWR field of configuration registers [NSEC\\_CTRL](#) and [SEC\\_CTRL](#). In effect of the settings of the registers the power state request is denied if the minimum power state set in DISMINPWR is higher than the requested power state and there is at least one channel configured as Non-secure or Secure respectively.

Entering Full retention mode state while the DMA is in an actively waiting state can be Disabled and Enabled by the IDLERETEN field of configuration registers [NSEC\\_CTRL](#) and [SEC\\_CTRL](#). In effect of the settings of these registers the power state request is always denied if IDLERETEN is set to Disabled, the requested state is Full retention mode and at least one Non-secure or Secure channel (according to the IDLERETEN field's security) is waiting for an event.

For more information on the P-Channel handshake mechanism, see the [AMBA® Low Power Interface Specification](#) document.

For P-Channel interface signals, see [Signal descriptions](#).

### 5.9.1.2 Clock Q-Channel

The DMA-350 is using one full LPI Q-Channel for clock management. The purpose of this feature is to reduce power consumption by shutting down the clock while not in active use. The clock Q-Channel is used to turn on and off the clock for the DMAC logic.

The Q-Channel interface controls the DMAC clock management. When the DMAC is active, clock quiescence requests are denied and the operation simply continues. When the DMAC is actively waiting for an event or is inactive, the quiescence request is accepted and the clock can be shut down.

For more information on the Q-Channel handshake mechanism, see the [AMBA® Low Power Interface Specification](#) document.

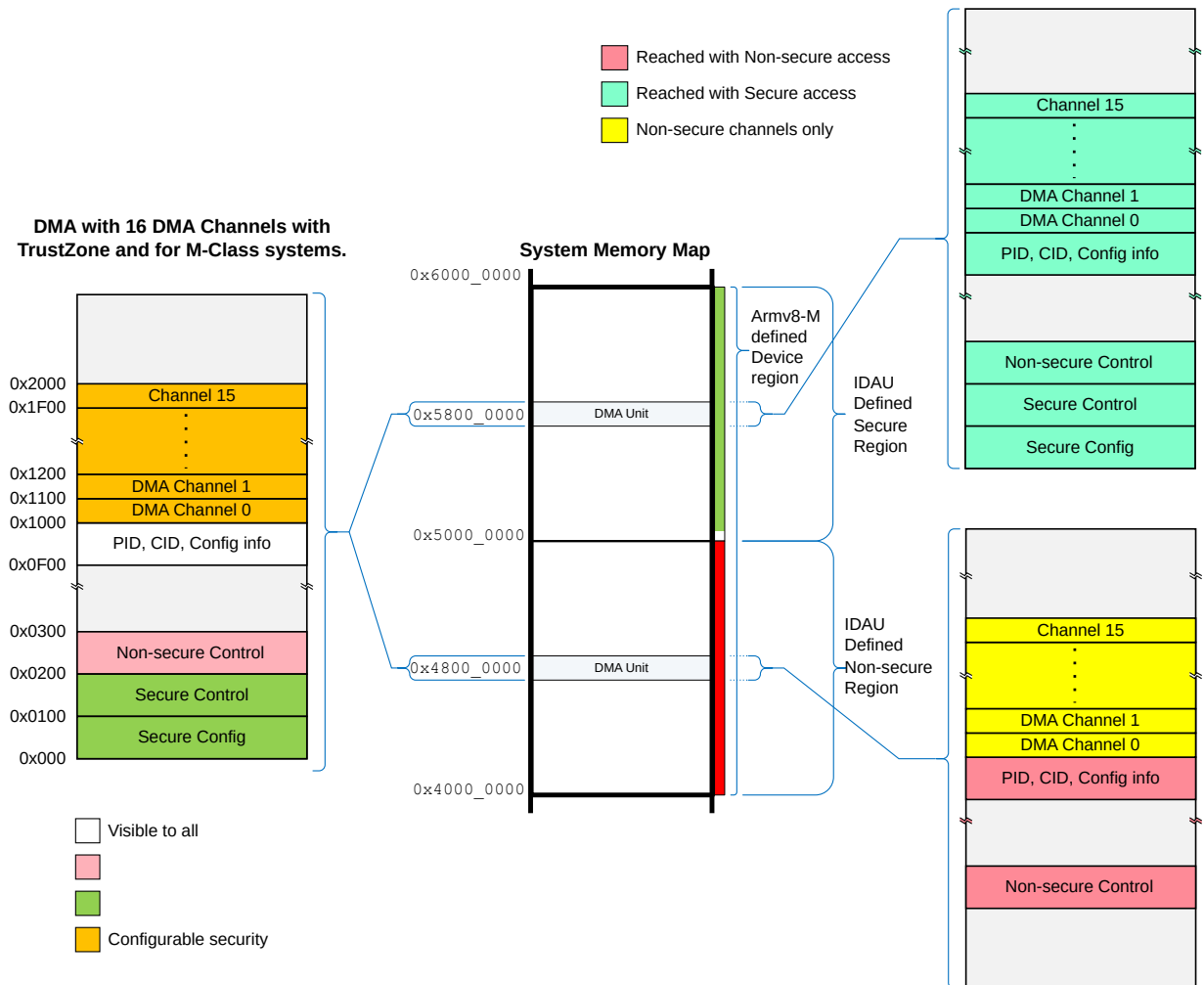
For Q-Channel interface signals, see [Signal descriptions](#).

## 5.9.2 Configuration

The DMA-350 can be set up through its configuration registers. These registers are divided into several frames. Each register frame occupies a 256 byte address space.

The layout of the register frames is shown in the following figure.

**Figure 5-26: DMA register frame layout**



The whole DMA unit occupies an 8kB configuration address space. For detailed description about each register, see [Register descriptions](#).

### 5.9.2.1 DMA unit configuration

The following four register frames are associated with the DMA unit:

#### Security Configuration Register Frame (DMASECCFG)

This frame contains registers to configure the security and privilege of each DMA channel, and the security of the external trigger ports.

#### Secure Control Register Frame (DMASECCTRL)

This frame contains status and control registers that affect all channels configured as Secure.

#### Non-secure Control Register Frame (DMANSECCTRL)

This frame contains status and control registers that affect all channels configured as Non-secure.

## Information Register Frame (DMAINFO)

This frame provides information about the capabilities and parameters of the DMA unit.

### 5.9.2.2 Channel configuration

Each DMA channel has its own register frame which contains the channel-specific configuration registers. The configuration registers are writable only when the channel is not enabled. When the ENABLECMD bit is set in the CH<x>\_CMD register, the settings are frozen throughout the execution of the command.

To configure a command correctly, the settings written to the configuration registers must meet the following criteria:

- The register fields must contain allowed values only. To see which values are valid, see [Programmers model](#). All bits of reserved fields must be zeroes. Failing to meet this criteria results in REGVALERR.
- The settings must describe a command that is valid, that is, there are no conflicting parameters. Failing to meet this criteria results in CFGCONFLERR.

### Configuring a channel command

The following registers and fields must be written to set up a DMA command:

#### 1. CH<x>\_CTRL register:

- CHPRIO - DMA channel priority
- TRANSIZE - Unit of data transfers
- XTYPE / YTYPE - Type of intended data transfer
- USE\* fields to select whether triggers / GPO / Stream interface to be used
- DONETYPE / DONEPAUSEEN fields to control at which point of the command execution the STAT\_DONE flag is to be signaled, and if the channel is to be paused as a result.

#### 2. Source / destination start addresses

CH<x>\_SRCADDR and CH<x>\_SRCADDRHI define source start address, while CH<x>\_DESADDR and CH<x>\_DESADDRHI define destination start address of the transfer.

#### 3. Source / destination sizes, Y-stride

SRCXSIZE and SRCXSIZEHI define source XSIZE, while DESXSIZE and DESXSIZEHI define destination XSIZE parameters of the transfer. These fields are located in the CH<x>\_XSIZE and CH<x>\_XSIZEHI registers.

If a 2D transfer was selected through YTYPE, the YSIZE parameters must also be set by the SRCYSIZE and DESYSIZE fields in the CH<x>\_YSIZE register. In addition, the SRCYADDRSTRIDE / DESYADDRSTRIDE parameters must be configured in the CH<x>\_YADDRSTRIDE register for 2D transfers.

#### 4. Source / destination address increments:

SRCXADDRINC and DESXADDRINC define the address increments in the CH<x>\_XADDRINC register.

These fields are encoded as two's complement numbers, so negative and zero values are also possible to be used.

5. Source / destination bus attributes in the CH<x>\_SRCTRANSCFG / CH<x>\_DESTRANSCFG registers:
  - Security and privilege attributes (\*NONSECATTR / \*PRIVATTR)
  - Memory and Shareability attributes (\*MEMATTRLO / \*MEMATTRHI / \*SHAREATTR)
  - Maximum burst length (\*MAXBURSTLEN)
6. If the selected transfer type in XTYPE or YTYPE is 'fill', the fill value must be set in the CH<x>\_FILLVAL register.
7. For templated transfers, the template patterns and template sizes must be set in the CH<x>\_TMPLTCFG, CH<x>\_SRCTMPLT, and CH<x>\_DESTMPLT registers. To disable templated transfers, set the template sizes to zero.
8. If triggering is used (selected by the USESRCTRIGIN, USEDESTRIGIN or USETRIGOUT fields), the triggers must be configured for the enabled triggers.
  - For trigger inputs, the following fields must be set in the CH<x>\_SRCTRIGINCFG / CH<x>\_DESTRIGINCFG registers:
    - \*TRIGINTYPE - the type of the input trigger (SW only, external HW, or internal trigger)
    - \*TRIGINMODE - the trigger input mode (command trigger, DMAC driven flow control trigger, or peripheral driven flow control trigger)
    - \*TRIGINSEL - to select which external trigger port to be used if external HW type is selected, or which channel to be connected if internal type is selected.
    - \*TRIGINBLKSIZE - to define the trigger block size for flow control trigger modes
  - For trigger outputs, the following fields must be set in the CH<x>\_TRIGOUTCFG register:
    - TRIGOUTTYPE - the type of the output trigger (SW only, external HW, or internal trigger)
    - TRIGOUTSEL - to select which external trigger port to be used if external HW type is selected (don't care for other types)
9. GPO value (CH<x>\_GPOVAL0) and enable mask (CH<x>\_GPOEN0) if GPO is to be used.
10. If Stream interface is to be used, the STREAMTYPE field must be configured in the CH<x>\_STREAMINTCFG register.
11. To enable auto restart, the restart counter must be set, or the infinite restart flag must be enabled:
  - Set CMDRESTARTCNT to the number of intended cycled minus one, or
  - Set CMDRESTARTINFEN to 1 to enable infinite automatic restarting of commands.
  - Select the registers to be reloaded with their initial value after each cycle by setting the REGRELOADTYPE field in the CH<x>\_CTRL register.
12. For command linking, the following fields must be set:

- LINKADDREN must be set to 1 to enable command linking
- LINKADDR / LINKADDRHI - to set the address of the linked-command descriptor
- LINKMEMATTRLO / LINKMEMATTRHI / LINKSHAREATTR - to set the bus transfer attributes to be used for fetching the descriptor

If a feature is not used by the command, we recommend that the fields associated with the feature are left in the reset values.

## Empty commands

There is a subset of settings that describe a valid command, but the execution does not result in actual data transfers. Such commands, called empty commands, can be useful. For example, to initiate a linked command chain, to set a GPO value, or to synchronize with other channels using the triggering features.

An empty command is executed if the XTYPE field is explicitly set to 'disable' (3'b000), or when neither data reads nor data writes are configured for the AXI5 interface.

The following cases result in no data reads through the AXI5 interface:

- Explicit empty command is requested:

```
XTYPE = 'disable'
```

- Stream to AXI mode is selected (if Stream is supported by the channel):

```
USESTREAM = 1'b1 and STREAMTYPE = 'Stream in only'
```

- Data source is configured to AXI, but source size is set to zero:

```
( USESTREAM = 1'b0 or (USESTREAM = 1'b1 and STREAMTYPE = {'Stream in and out' or  
'Stream out only'}) )  
and  
( SRCXSIZE = 0 or (YTYPE != 'disable' and SRCYSIZE = 0) )
```

- AXI to AXI mode is selected, but destination size is set to zero:

```
( USESTREAM = 1'b0 or (USESTREAM = 1'b1 and STREAMTYPE = 'Stream in and out') )  
and  
( DESXSIZE = 0 or (YTYPE != 'disable' and DESYSIZE = 0) )
```

The following cases result in no data writes through the AXI5 interface:

- Explicit empty command is requested:

```
XTYPE = 'disable'
```

- AXI to stream mode is selected (if Stream is supported by the channel):

```
USESTREAM = 1'b1 and STREAMTYPE = 'Stream out only'
```



- Data destination is configured to AXI, but destination size is set to zero:

```
( USESTREAM = 1'b0 or (USESTREAM = 1'b1 and STREAMTYPE = {'Stream in and out' or
'Stream in only'}) )
and
( DESXSIZE = 0 or (XTYPE != 'disable' and DESYSIZE = 0) )
```

- AXI to AXI mode is selected, but there is no data producer (source size is set to zero and no Y fill mode selected for 2D commands or no X fill selected for 1D commands):

```
( USESTREAM = 1'b0 or (USESTREAM = 1'b1 and STREAMTYPE = 'Stream in and out') )
and
( SRCXSIZE = 0 or (YTYPE != 'disable' and SRCYSIZE = 0) )
and
( ( YTYPE != 'disable' and YTYPE != 'fill' ) or ( YTYPE = 'disable' and XTYPE !=
'fill' ) )
```



The recommended method for configuring an empty command intentionally is to set XTYPE to 'disable'.

### Configuration errors because of invalid settings

The following invalid settings result in configuration errors. No data transfers are made, and the CFGERR and REGVALERR fields are set in the CH<x>\_ERRINFO register.

- TRANSIZE is set to be greater than the bus width.
- Non-existent trigger resource is selected:
  - If you configure HW trigger type (SRCTRIGINTYPE, DESTRIGINTYPE or TRIGOUTTYPE = 2'b10), SRCTRIGINSEL, DESTRIGINSEL or TRIGOUTSEL points to a HW trigger port number that is too high.
  - If you configure internal trigger type (SRCTRIGINTYPE or DESTRIGINTYPE = 2'b11), SRCTRIGINSEL, DESTRIGINSEL points to a channel number that is too high, or points to the channel itself that is being configured.
  - Configuring HW trigger type (SRCTRIGINTYPE, DESTRIGINTYPE or TRIGOUTTYPE = 2'b10), when the \$product does not have HW trigger in or trigger out port.

### Configuration errors because of conflicting settings

The following settings result in configuration conflict errors. No data transfers are, and the CFGERR and CFGCONFLERR fields are set in the CH<x>\_ERRINFO register.

- Flow control trigger input modes are only supported with 1D source or destination. It means that even if a 2D transfer type is selected, the YSIZE must be 1 on the given side.

In addition, flow control trigger modes are not supported when using the stream interface. The reason for this restriction is that block-based transfers might behave improperly when converted to stream output transfers.

Finally, wrap transfer typed are not allowed with source flow control trigger mode. The following configuration settings are illegal:

- Source flow control trigger input mode with 2D source:

```
USESRCTRIGIN = 1'b1 and SRCTRIGINMODE = 2'b1X and XTYPE != 'disable' and YTYPE != 'disable' and SRCYSIZE > 1
```

- Destination flow control trigger input mode with 2D destination:

```
USEDESTRIGIN = 1'b1 and DESTRIGINMODE = 2'b1X and XTYPE != 'disable' and YTYPE != 'disable' and DESYSIZE > 1
```

- Using Stream interface with source flow control trigger input mode

```
USESRCTRIGIN = 1'b1 and SRCTRIGINMODE = 2'b1X and USESTREAM = 1'b1
```

- Using Stream interface with destination flow control trigger input mode

```
USEDESTRIGIN = 1'b1 and DESTRIGINMODE = 2'b1X and USESTREAM = 1'b1
```

- Source flow control trigger input mode with wrap transfers:

```
USESRCTRIGIN = 1'b1 and SRCTRIGINMODE = 2'b1X and (XTYPE == 'wrap' or YTYPE == 'wrap')
```

- Flow control trigger modes imply that there must be data transfer configured for the given direction. For the definitions of no AXI5 read or write configured, see the Empty commands section above.

Therefore, the following settings are illegal:

- Source flow control trigger input mode when no AXI5 source data:

```
USESRCTRIGIN = 1'b1 and SRCTRIGINMODE = 2'b1X and No AXI5 read configured
```

- Destination flow control trigger input mode when no AXI5 destination data:

```
USEDESTRIGIN = 1'b1 and DESTRIGINMODE = 2'b1X and No AXI5 write configured
```

- The stream interface cannot work with certain transfer types. The following settings are illegal:

```
USESTREAM = 1'b1
and
( (XTYPE == 'wrap' and YTYPE == 'disable') or
  (XTYPE == 'continue' and YTYPE == 'wrap') or
  (XTYPE == {'wrap' or 'fill'} and YTYPE == {'wrap' or 'continue' or 'fill'})
)
```

- When using the stream interface with AXI to AXI transfers and fill transfer types, there must be AXI data source. Therefore, the following settings are illegal:

```
USESTREAM = 1'b1 and STREAMTYPE = 'Stream in and out'
and
( (XTYPE == 'fill' and YTYPE == 'disable') or (XTYPE == 'continue' and YTYPE ==
'fill') )
and
No AXI5 read configured
```

- Templated transfers are not allowed with 2D transfer types.

```
YTYPE != 'disable' and (SRCTMPLTSIZE != 0 or DESTMPLTSIZE != 0)
```

### 5.9.3 Halting and restarting the DMA with CTI

The DMA-350 can be halted for debug purposes using the CTI interface.

When a HIGH is received on the CTI halt\_req signal, pausing of both the Secure and Non-secure channels is initiated.

When all the channels reach either paused or not enabled state after a halt request, a pulse on the halted CTI signal is sent. After the DMAC reaches the paused state, it waits for a HIGH on the restart\_req CTI signal. When it is received, the DMA channels return to their state before the halt request.

The CTI halt request and restart request are ignored in certain cases:

- The halt\_req is ignored after a previous halt request and before the DMAC returns to normal operation.
- The restart request is ignored before a halt request is received. After a valid HIGH is received on the restart\_req, further level changes are ignored.

The halt request from the CTI is combined with the hardware pause, allch\_pause\_req, and the software initiated allchpause. The DMA channels can be paused using any of the three methods.

## 5.10 Initialization

The DMAC must be configured to set the security and privilege attributes for each channel and their attached resources before setting the channels for memory transfers. When the configuration is complete, the channel configuration registers are protected against malicious accesses. The security configuration can be locked until the next reset of the DMAC to further protect the security settings. Privilege settings of a channel can be adjusted when the channel is in IDLE. The register settings are automatically cleared when the security or privilege settings of a channel change.



We recommend that the security or privilege change is done with a sequence of writing and reading back the desired new state to assure the success of the change.

The DMAC checks the boot\_en port after reset. If automatic booting is turned on, the DMAC sets the boot address as the first link address for channel 0 and enables the channel before any APB4 configuration can occur. When security is enabled for the DMAC, the channel 0 is set to be Secure as all other channels so only Secure commands can be fetched from the memory.

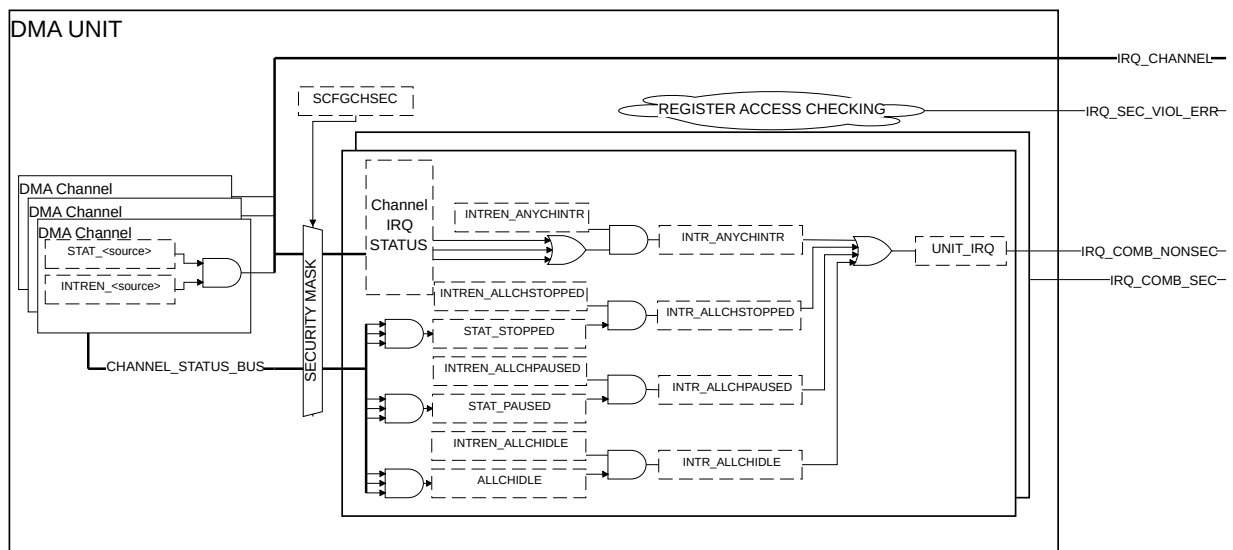
## 5.11 Interrupt operation

Each channel has its own interrupt to indicate state changes within the channel. There are DMA unit level interrupts that show unit level state changes. The SW can enable, disable, or clear the interrupts.

When both Secure and Non-secure interrupts are used, each interrupt has its handler running in the proper security world. This means that alignment is required when changing the Security state of the channels. The channel interrupt is considered Secure when the channel is configured as Secure.

The channel and unit level interrupts can be seen in the following figure.

**Figure 5-27: DMA Unit level interrupts**



**Table 5-16: Non-secure DMA level interrupt signal sources**

Interrupt source	Description
chintrstatus0	Collated channel Non-secure interrupts.

Interrupt source	Description
intr_anychintr	This interrupt is a combined Non-secure interrupt, combining the other Non-secure interrupt sources and all the Non-secure channel interrupts when NSEC_CTRL.INTREN_ANYCHINTR is set to 1.
intr_allchidle	This interrupt is raised when every Non-secure channel returns to IDLE state from a non-IDLE state. The interrupt is not asserted after reset.
intr_allchstopped	This interrupt is raised when the last Non-secure channel enters stopped state.
intr_allchpaused	This interrupt is raised when the last Non-secure channel enters paused state.

**Table 5-17: Secure DMA level interrupt signal sources**

Interrupt source	Description
chintrstatus0	Collated channel Secure interrupt flags.
intr_anychintr	This interrupt is a combined Secure interrupt, combining the other Secure interrupt sources and all the Secure channel interrupts when SEC_CTRL.INTREN_ANYCHINTR is set to 1.
intr_allchidle	This interrupt is raised when every Secure channel returns to idle state from a non-idle state. The interrupt is not asserted after reset.
intr_allchstopped	This interrupt is raised when the last Secure channel enters stopped state.
intr_allchpaused	This interrupt is raised when the last Secure channel enters paused state.

The irq\_sec\_viol\_err interrupt signal is provided to notify the Secure entity that a security violation has occurred with register access to the DMA. The irq\_sec\_viol\_err interrupt signal does not exist when SECEXT\_PRESENT is set to 0.

## 6. Programmers model

This section describes the functionality of the DMA-350 from a programming perspective.

### 6.1 About the programmers model

This section describes the functions and programmers model of the CoreLink DMA-350.

When using the programmers model, adhere to the following guidelines:

- Do not attempt to access reserved or unused address locations. Attempting to access these locations can result in unpredictable behavior.
- Unless otherwise stated in the accompanying text:
  - Do not modify undefined register bits.
  - Ignore undefined register bits on reads.
  - Unless otherwise specified, all register bits are reset to a logic 0 by a system or power up reset.

The following describes the access type:

#### **RW**

Read and write

#### **RO**

Read-only

#### **WO**

Write-only

#### **RAZ**

Read-As-Zero

#### **WI**

Writes ignored

#### **W1C**

Write 1 to Clear

### 6.2 Programming considerations

The DMA channels can be configured to transfer several data elements from one location to another. There are multiple settings for every command that define the way the DMAC sends the transfers to the bus interface.

TRANSIZE setting adjusts the size of the data elements the command moves. The setting is the same for both read and write directions and ranging in power of 2 steps from a single byte to the

number of bytes that fit into the DATA\_WIDTH of the bus interface. This sets the base for all internal counters, the address increment logic, and the burst calculation logic.

The XSIZE setting defines the number of data elements moved in the X direction. For 1D transfers, this is the total number of transfers. For 2D transfers, this is the length of a line so the total number of transfers can be calculated by XSIZE \* YSIZE.



The DMAC generates bursts so that they do not cross between lines. So that after completing a line, the DMAC starts transferring the next line with a separate burst.

The TRIGINBLKSIZE register field adjusts the maximum number of elements sent for a block trigger request. This can be used to split up the command into smaller parts. The total number of elements in the command is not necessarily a multiple of TRIGINBLKSIZE so partial blocks can also be sent at the end of a command. The TRIGINBLKSIZE setting can be different on source and destination sides which may result in having partial data stored temporarily in the FIFO until the triggers are acknowledged.

For example, the SRCTRIGINBLKSIZE is 5 and DESTTRIGINBLKSIZE is 8, that results in having 2 triggers on the source side that generate enough data to the destination side but leaves 2 data elements in the FIFO until the next destination trigger is received.

1. Source trigger received – 5 elements read, FIFO level increased to 5.
2. Source trigger received – 5 elements read, FIFO level increased to 10.
3. Destination trigger received – 8 elements written, FIFO level decreased to 2.

Trigger events can happen in parallel and it depends on the FIFO size and its actual level whether the read or write side of the channel must wait for the other side to finish. Care must be taken when programming the trigger block size settings for timing critical applications.

The MAXBURSTLEN field limits the maximum number of beats sent out by the DMAC on the bus interface for a burst request. This setting can be used to adjust the bus utilization during the command and also creates arbitration points when multiple channels try to access the bus concurrently.

Based on these settings one example DMA command consists of XSIZE number of TRANSIZE-based data elements. This command can be split into several blocks defined by TRIGINBLKSIZE which can also be split into multiple bursts limited by the MAXBURSTLEN. The following example shows how the XSIZE=20, TRIGINBLKSIZE=7 and MAXBURSTLEN=3 settings affect the command execution. Trigger blocks and bursts are only defined to have a maximum size so the DMA command may send smaller trigger blocks and bursts as it is seen in the figure.

**Table 6-1: Using different sizes in a command**

XSIZE = 20							
TRIGINBLKSIZE = 7			TRIGINBLKSIZE = 7			TRIGINBLKSIZE = 6	
BURST	BURST	BURST	BURST	BURST	BURST	BURST	BURST

XSIZE = 20							
TRIGINBLKSIZE = 7			TRIGINBLKSIZE = 7			TRIGINBLKSIZE = 6	
3	3	1	3	3	1	3	3

The DMAC selects the largest burst possible to reduce the number of address requests on the bus, but this can be limited if the bandwidth or latency requirements require adjustments from software.

Besides DATA\_WIDTH, TRANSIZE, XSIZE, TRIGINBLKSIZE and MAXBURSTLEN, burst generation is also affected by the following factors:

- The size of the FIFO within a DMA channel limits the maximum amount of data in a burst. The DMA channels only generate bursts that do not exceed half of the capacity of the corresponding Data FIFO. This limitation mitigates latency on write and read AXI buses and so enables gapless command execution within a line.
- To enable easier AXI to AHB conversion, the DMAC only generates bursts that do not cross the 1kB address boundary.
- Certain transfer properties such as transfer type (device or memory) and aligned or unaligned starting and ending addresses might also affect how the DMAC generates bursts, as the DMAC will try to optimize bandwidth by utilizing as much of the bus width as possible and will group transfers if it can.

For more details, see [AXI bandwidth utilization](#).

## 6.3 Memory map

The memory map includes the maximum number of implemented blocks.

**Table 6-2: Memory map**

Address range	Description
0x0000 - 0x00ff	DMA Unit Security Configuration Register Frame
0x0100 - 0x01ff	DMA Unit Secure Control Register Frame
0x0200 - 0x02ff	DMA Unit Non-Secure Control Register Frame
0x0300 - 0x0eff	Reserved
0x0f00 - 0x0fff	DMA Unit Information Register Frame
0x1000 - 0x10ff	DMA Channel 0 Register Frame
0x1100 - 0x11ff	DMA Channel 1 Register Frame
0x1200 - 0x12ff	DMA Channel 2 Register Frame
0x1300 - 0x13ff	DMA Channel 3 Register Frame
0x1400 - 0x14ff	DMA Channel 4 Register Frame
0x1500 - 0x15ff	DMA Channel 5 Register Frame
0x1600 - 0x16ff	DMA Channel 6 Register Frame
0x1700 - 0x17ff	DMA Channel 7 Register Frame
0x1800 - 0x1fff	Reserved



## 6.4 Register summary

The DMA-350 register map is divided into the following sections:

- [DMACH<n> summary](#)
- [DMANSECCTRL summary](#)
- [DMASECCTRL summary](#)
- [DMASECCFG summary](#)
- [DMAINFO summary](#)

### 6.4.1 DMACH<n> summary

DMA Channel Register Frame.

This block contains the channel registers related to the execution of the DMA command. When the registers for the command are configured and the ENABLECMD is set, then most of the registers become read-only except for a few that allow control from the SW during the operation of the command.

The following global constraints apply to all registers in this block:

- When the channel is set to Secure, Non-secure accesses to this register block are not allowed and response is based on the security configuration settings.
- When the channel is set to privileged, then unprivileged accesses to this register block are treated as **RAZ/WI**.
- Base address:  $0x1000 + 0x0100 * <n>$
- Size:  $0x0100$
- Instances: NUM\_CHANNELS

**Table 6-3: DMACH<n> register summary**

Offset	Name	Type	Default	Width	Description
0x00	CH_CMD	RW	0x00000000	32	See <a href="#">CH_CMD</a>
0x04	CH_STATUS	RW	0x00000000	32	See <a href="#">CH_STATUS</a>
0x08	CH_INTREN	RW	0x00000000	32	See <a href="#">CH_INTREN</a>
0x0C	CH_CTRL	RW	0x00200200	32	See <a href="#">CH_CTRL</a>
0x10	CH_SRCADDR	RW	0x00000000	32	See <a href="#">CH_SRCADDR</a>
0x14	CH_SRCADDRHI	RW	0x00000000	32	See <a href="#">CH_SRCADDRHI</a>
0x18	CH_DESADDR	RW	0x00000000	32	See <a href="#">CH_DESADDR</a>
0x1C	CH_DESADDRHI	RW	0x00000000	32	See <a href="#">CH_DESADDRHI</a>
0x20	CH_XSIZE	RW	0x00000000	32	See <a href="#">CH_XSIZE</a>
0x24	CH_XSIZEHI	RW	0x00000000	32	See <a href="#">CH_XSIZEHI</a>
0x28	CH_SRCTRANSCFG	RW	0x000F0400	32	See <a href="#">CH_SRCTRANSCFG</a>
0x2C	CH_DESTRANSCFG	RW	0x000F0400	32	See <a href="#">CH_DESTRANSCFG</a>

Offset	Name	Type	Default	Width	Description
0x30	CH_XADDRINC	RW	0x00000000	32	See <a href="#">CH_XADDRINC</a>
0x34	CH_YADDRSTRIDE	RW	0x00000000	32	See <a href="#">CH_YADDRSTRIDE</a>
0x38	CH_FILLVAL	RW	0x00000000	32	See <a href="#">CH_FILLVAL</a>
0x3C	CH_YSIZE	RW	0x00000000	32	See <a href="#">CH_YSIZE</a>
0x40	CH_TMPLTCFG	RW	0x00000000	32	See <a href="#">CH_TMPLTCFG</a>
0x44	CH_SRCTMPLT	RW	0x00000001	32	See <a href="#">CH_SRCTMPLT</a>
0x48	CH_DESTMPLT	RW	0x00000001	32	See <a href="#">CH_DESTMPLT</a>
0x4C	CH_SRCTRIGINCFG	RW	0x00000000	32	See <a href="#">CH_SRCTRIGINCFG</a>
0x50	CH_DESTRIGINCFG	RW	0x00000000	32	See <a href="#">CH_DESTRIGINCFG</a>
0x54	CH_TRIGOUTCFG	RW	0x00000000	32	See <a href="#">CH_TRIGOUTCFG</a>
0x58	CH_GPOENO	RW	0x00000000	32	See <a href="#">CH_GPOENO</a>
0x60	CH_GPOVAL0	RW	0x00000000	32	See <a href="#">CH_GPOVAL0</a>
0x68	CH_STREAMINTCFG	RW	0x00000000	32	See <a href="#">CH_STREAMINTCFG</a>
0x70	CH_LINKATTR	RW	0x00000000	32	See <a href="#">CH_LINKATTR</a>
0x74	CH_AUTOCFG	RW	0x00000000	32	See <a href="#">CH_AUTOCFG</a>
0x78	CH_LINKADDR	RW	0x00000000	32	See <a href="#">CH_LINKADDR</a>
0x7C	CH_LINKADDRHI	RW	0x00000000	32	See <a href="#">CH_LINKADDRHI</a>
0x80	CH_GPOREAD0	RO	0x00000000	32	See <a href="#">CH_GPOREAD0</a>
0x88	CH_WRKREGPTR	RW	0x00000000	32	See <a href="#">CH_WRKREGPTR</a>
0x8C	CH_WRKREGVAL	RO	0x00000000	32	See <a href="#">CH_WRKREGVAL</a>
0x90	CH_ERRINFO	RO	0x00000000	32	See <a href="#">CH_ERRINFO</a>
0xC8	CH_IIDR	RO	IMPL_DEF	32	See <a href="#">CH_IIDR</a>
0xCC	CH_AIDR	RO	IMPL_DEF	32	See <a href="#">CH_AIDR</a>
0xE8	CH_ISSUECAP	RW	IMPL_DEF	32	See <a href="#">CH_ISSUECAP</a>
0xF8	CH_BUILDCFG0	RO	IMPL_DEF	32	See <a href="#">CH_BUILDCFG0</a>
0xFC	CH_BUILDCFG1	RO	IMPL_DEF	32	See <a href="#">CH_BUILDCFG1</a>

## 6.4.2 DMANSECCTRL summary

DMA Unit Non-secure Control Register Frame.

This block contains the configuration registers for all non-secure channels and other Non-secure resources of the DMAC. The Non-secure SW can control and monitor the status of the channels before and during the execution of the DMA commands. Some of the registers become read-only when the ENABLECMD bit is set for the selected channel. This register frame can be accessed by both secure privileged and Non-secure privileged accesses.

The following global constraints apply to all registers in this block:

- Unprivileged accesses result in **RAZ/WI** response.
- Base address: 0x0200
- Size: 0x0100

- Instances: 1

**Table 6-4: DMANSECCTRL register summary**

Offset	Name	Type	Default	Width	Description
0x00	NSEC_CHINTRSTATUS0	RO	0x00000000	32	See <a href="#">NSEC_CHINTRSTATUS0</a>
0x08	NSEC_STATUS	RW	0x00000000	32	See <a href="#">NSEC_STATUS</a>
0x0C	NSEC_CTRL	RW	0x00000000	32	See <a href="#">NSEC_CTRL</a>
0x14	NSEC_CHPTR	RW	0x00000000	32	See <a href="#">NSEC_CHPTR</a>
0x18	NSEC_CHCFG	RW	0x00000000	32	See <a href="#">NSEC_CHCFG</a>
0xF0	NSEC_STATUSPTR	RW	0x00000000	32	See <a href="#">NSEC_STATUSPTR</a>
0xF4	NSEC_STATUSVAL	RO	0x00000000	32	See <a href="#">NSEC_STATUSVAL</a>
0xF8	NSEC_SIGNALPTR	RW	0x00000000	32	See <a href="#">NSEC_SIGNALPTR</a>
0xFC	NSEC_SIGNALVAL	RW	0x00000000	32	See <a href="#">NSEC_SIGNALVAL</a>

### 6.4.3 DMASECCTRL summary

DMA Unit Secure Control Register Frame.

This block contains the configuration registers for all Secure channels and other Secure resources of the DMAC. The Secure software can control and monitor the status of the channels before and during the execution of the DMA commands. Some of the registers become read-only when the ENABLECMD bit is set for the selected channel. This register frame can be accessed by Secure privileged accesses only.

The following global constraints apply to all registers in this block:

- Unprivileged accesses result in **RAZ/WI** response.
- Non-secure accesses result in **RAZ/WI** or error response depending on the security configuration of the DMAC.
- Base address: 0x0100
- Size: 0x0100
- Instances: 1

**Table 6-5: DMASECCTRL register summary**

Offset	Name	Type	Default	Width	Description
0x00	SEC_CHINTRSTATUS0	RO	0x00000000	32	See <a href="#">SEC_CHINTRSTATUS0</a>
0x08	SEC_STATUS	RW	0x00000000	32	See <a href="#">SEC_STATUS</a>
0x0C	SEC_CTRL	RW	0x00000000	32	See <a href="#">SEC_CTRL</a>
0x14	SEC_CHPTR	RW	0x00000000	32	See <a href="#">SEC_CHPTR</a>
0x18	SEC_CHCFG	RW	0x00000000	32	See <a href="#">SEC_CHCFG</a>
0xF0	SEC_STATUSPTR	RW	0x00000000	32	See <a href="#">SEC_STATUSPTR</a>
0xF4	SEC_STATUSVAL	RO	0x00000000	32	See <a href="#">SEC_STATUSVAL</a>
0xF8	SEC_SIGNALPTR	RW	0x00000000	32	See <a href="#">SEC_SIGNALPTR</a>

Offset	Name	Type	Default	Width	Description
0xFC	SEC_SIGNALVAL	RW	0x00000000	32	See <a href="#">SEC_SIGNALVAL</a>

## 6.4.4 DMASECCFG summary

DMA Unit Security Configuration Register Frame.

This block contains the configuration registers for the security related behavior of the DMAC. The security configuration is recommended to be defined before the operation of the DMAC and caution must be taken if the configuration is changed during runtime. This register frame can be accessed by Secure privileged accesses only.

The following global constraints apply to all registers in this block:

- Unprivileged accesses result in **RAZ/WI** response.
- Non-secure accesses result in **RAZ/WI** or error response depending on the security configuration of the DMAC.
- Base address: 0x0000
- Size: 0x0100
- Instances: 1

**Table 6-6: DMASECCFG register summary**

Offset	Name	Type	Default	Width	Description
0x00	SCFG_CHSECO	RW	0x00000000	32	See <a href="#">SCFG_CHSECO</a>
0x08	SCFG_TRIGINSECO	RW	0x00000000	32	See <a href="#">SCFG_TRIGINSECO</a>
0x28	SCFG_TRIGOUTSECO	RW	0x00000000	32	See <a href="#">SCFG_TRIGOUTSECO</a>
0x40	SCFG_CTRL	RW	0x00000000	32	See <a href="#">SCFG_CTRL</a>
0x44	SCFG_INTRSTATUS	RW	0x00000000	32	See <a href="#">SCFG_INTRSTATUS</a>

## 6.4.5 DMAINFO summary

DMA Unit Information Register Frame.

This frame provides information about the capabilities and parameters of the DMA unit. The registers in this frame can be access by all types of accesses.

- Base address: 0x0F00
- Size: 0x0100
- Instances: 1

**Table 6-7: DMAINFO register summary**

Offset	Name	Type	Default	Width	Description
0xB0	DMA_BUILDCFG0	RO	IMPL_DEF	32	See <a href="#">DMA_BUILDCFG0</a>

Offset	Name	Type	Default	Width	Description
0xB4	DMA_BUILDCFG1	RO	IMPL_DEF	32	See <a href="#">DMA_BUILDCFG1</a>
0xB8	DMA_BUILDCFG2	RO	IMPL_DEF	32	See <a href="#">DMA_BUILDCFG2</a>
0xC8	IIDR	RO	IMPL_DEF	32	See <a href="#">IIDR</a>
0xCC	AIDR	RO	IMPL_DEF	32	See <a href="#">AIDR</a>
0xD0	PIDR4	RO	IMPL_DEF	32	See <a href="#">PIDR4</a>
0xE0	PIDR0	RO	IMPL_DEF	32	See <a href="#">PIDR0</a>
0xE4	PIDR1	RO	IMPL_DEF	32	See <a href="#">PIDR1</a>
0xE8	PIDR2	RO	IMPL_DEF	32	See <a href="#">PIDR2</a>
0xEC	PIDR3	RO	0x00000000	32	See <a href="#">PIDR3</a>
0xF0	CIDR0	RO	0x0000000D	32	See <a href="#">CIDR0</a>
0xF4	CIDR1	RO	0x000000F0	32	See <a href="#">CIDR1</a>
0xF8	CIDR2	RO	0x00000005	32	See <a href="#">CIDR2</a>
0xFC	CIDR3	RO	0x000000B1	32	See <a href="#">CIDR3</a>

## 6.5 Register descriptions

[Register summary](#) provides cross references to individual registers.

### 6.5.1 DMACH<n> description

DMA Channel Register Frame.

For an overview of the frame, and a list of constraints that apply to this block, see [DMACH<n> summary](#).

#### 6.5.1.1 CH\_CMD

The Channel DMA Command register allows the SW to control the operation of a DMA command.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x000

##### Type

RW

## Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-8: CH\_CMD register bit descriptions**

Bits	Name	Description	Type	Default
[31:25]	-	Reserved, <b>RAZ/WI</b>	-	-
[24]	SWTRIGOUTACK	Software Generated Trigger Output Acknowledge. Write '1' to acknowledge a Trigger Output request from the DMA. Once set to '1', this remains high until the DMA Trigger Output is raised (either on the trigger output signal or as an interrupt) and the acknowledge is accepted. When the channel is not enabled, write to this register is ignored.	W1S	0x0
[23]	-	Reserved, <b>RAZ/WI</b>	-	-
[22:21]	DESSWTRIGINTYPE	Software Generated Destination Trigger Input Request Type.   RW   0x0  Selects the trigger request type for the destination trigger input when the SW triggers the DESSWTRIGINREQ bit. <ul style="list-style-type: none"> <li>00: Single request</li> <li>01: Last single request</li> <li>10: Block request</li> <li>11: Last block request</li> </ul> This field cannot be changed while the DESSWTRIGINREQ bit is set. The field is <b>RAZ/WI</b> when the following condition is False: (1) & (NUM_TRIGGER_IN > 0)		
[20]	DESSWTRIGINREQ	Software Generated Destination Trigger Input Request. Write to '1' to create a SW trigger request to the DMA with the specified type in the DESSWTRIGINTYPE register. Once set to '1', this remains high until the DMA is accepted the trigger and returns this to '0'. It is also cleared automatically if the current command is completed without expecting another trigger event. When the channel is not enabled, write to this register is ignored.	W1S	0x0
[19]	-	Reserved, <b>RAZ/WI</b>	-	-
[18:17]	SRCSWTRIGINTYPE	Software Generated Source Trigger Input Request Type.  Selects the trigger request type for the source trigger input when the SW triggers the SRCSWTRIGINREQ bit. <ul style="list-style-type: none"> <li>00: Single request</li> <li>01: Last single request</li> <li>10: Block request</li> <li>11: Last block request</li> </ul> This field cannot be changed while the SRCSWTRIGINREQ bit is set. The field is <b>RAZ/WI</b> when the following condition is False: (1) & (NUM_TRIGGER_IN > 0)	RW	0x0

Bits	Name	Description	Type	Default
[16]	SRCSWTRIGINREQ	Software Generated Source Trigger Input Request. Write to '1' to create a SW trigger request to the DMA with the specified type in the SRCSWTRIGINTYPE register. Once set to '1', this remains high until the DMA accepted the trigger and returns this to '0'. It is also cleared automatically if the current command is completed without expecting another trigger event. When the channel is not enabled, write to this register is ignored.	W1S	0x0
[15:6]	-	Reserved, <b>RAZ/WI</b>	-	-
[5]	RESUMECMD	Resume Current DMA Operation. Writing this bit to '1' means that the DMAC can continue the operation of a paused channel. Can be set to '1' when the PAUSECMD or a STAT_DONE assertion with DONEPAUSEEN set HIGH results in pausing the current DMA channel operation indicated by the STAT_PAUSED and STAT_RESUMEWAIT bits. Otherwise, writes to this bit are ignored.	W1S	0x0
[4]	PAUSECMD	Pause Current DMA Operation. Once set to '1' the status cannot change until the DMA operation reached the paused state indicated by the STAT_PAUSED and STAT_RESUMEWAIT bits. The bit can be set by SW by writing it to '1', the current active DMA operation is paused as soon as possible, but the ENABLECMD bit remains HIGH to show that the operation is still active. Cleared automatically when STAT_RESUMEWAIT is set and the RESUMECMD bit is written to '1', meaning that the SW continues the operation of the channel. Note that each DMA channel can have other sources of a pause request and this field does not reflect the state of the other sources. When set at the same time as ENABLECMD or when the channel is not enabled then write to this register is ignored.	W1S	0x0
[3]	STOPCMD	Stop Current DMA Operation. Once set to '1', this remains high until the DMA channel is stopped cleanly. Then this returns to '0' and ENABLECMD is also cleared. When set at the same time as ENABLECMD or when the channel is not enabled then write to this register is ignored. Note that each DMA channel can have other sources of a stop request and this field does not reflect the state of the other sources.	W1S	0x0
[2]	DISABLECMD	Disable DMA Operation at the end of the current DMA command operation. Once set to '1', this field stays high and the current DMA command is allowed to complete, but the DMA does not fetch the next linked command or auto-restarts the DMA command even if they are set. Once the DMA has stopped, it returns to '0' and ENABLECMD is also cleared. When set at the same time as ENABLECMD or when the channel is not enabled then write to this register is ignored.	W1S	0x0
[1]	CLEARCMD	DMA Clear command. When set to '1', it remains high until all DMA channel registers and any internal queues and buffers are cleared, before returning to '0'. When set at the same time as ENABLECMD or while the DMA channel is already enabled, the clear only occurs after any ongoing DMA operation is either completed, stopped or disabled and the ENABLECMD bit is deasserted by the DMA.	W1S	0x0
[0]	ENABLECMD	Channel Enable. When set to '1', enables the channel to run its programmed task. When set to '1', it cannot be set back to zero, and this field automatically clears to zero when a DMA process is completed. To force the DMA to stop prematurely, you must use CH_CMD.STOPCMD instead.	W1S	0x0

### 6.5.1.2 CH\_STATUS

The Channel Status register shows the internal status of the DMA command and also reports interrupts about internal events.

#### Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x004

### Type

RW

### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-9: CH\_STATUS register bit descriptions**

Bits	Name	Description	Type	Default
[31:27]	-	Reserved, <b>RAZ/WI</b>	-	-
[26]	STAT_TRIGOUTACKWAIT	Channel is waiting for output Trigger Acknowledgment Status. This bit is set to HIGH when DMA channel starts waiting for output trigger Acknowledgment. Automatically cleared when the output trigger Acknowledgment is received either from HW or SW source.	RO	0x0
[25]	STAT_DESTRIGINWAIT	Channel is waiting for Destination Trigger Status. This bit is set to HIGH when DMA channel starts waiting for destination input trigger request. Automatically cleared when the destination trigger request is received either from HW or SW source.	RO	0x0
[24]	STAT_SRCTRIGINWAIT	Channel is waiting for Source Trigger Status. This bit is set to HIGH when DMA channel starts waiting for source input trigger request. Automatically cleared when the source trigger request is received either from HW or SW source.	RO	0x0
[23:22]	-	Reserved, <b>RAZ/WI</b>	-	-
[21]	STAT_RESUMEWAIT	Waiting for resume from software Flag. This flag indicates that the DMA channel successfully paused the operation of the command and needs SW acknowledgment to resume the operation. Is set to HIGH if STAT_PAUSED is asserted and the PAUSECMD bit set in the command register or when the STAT_DONE is asserted and the DONEPAUSEEN bit is set. Cleared when the RESUMECMD bit is set in the command register.	RO	0x0
[20]	STAT_PAUSED	Paused Status Flag. This flag is set to HIGH if the DMA channel successfully paused the operation of the command. The pause request can come from many internal or external sources. When the request to pause is not asserted anymore the bit is cleared automatically and the command operation can continue.	RO	0x0
[19]	STAT_STOPPED	Stopped Status Flag. This flag is set to HIGH if the DMA channel successfully reached the stopped state. The stop request can come from many internal or external sources. Write '1' to this bit to clear it. Automatically cleared when the ENABLECMD is set.	W1C	0x0
[18]	STAT_DISABLED	Disabled Status Flag. This flag is set to HIGH if the DMA channel is successfully disabled using the DISABLECMD command. Write '1' to this bit to clear it. Automatically cleared when the ENABLECMD is set.	W1C	0x0



Bits	Name	Description	Type	Default
[17]	STAT_ERR	Error Status Flag. This flag is set to HIGH if the DMA encounters an error during its operation. The details about the error event can be found in the ERRINFO register. Write '1' to this bit to clear it. When cleared, it also clears the ERRINFO register. Automatically cleared when the ENABLECMD is set.	W1C	0x0
[16]	STAT_DONE	Done Status Flag. This flag is set to HIGH when the DMA command reaches the state defined by the DONETYPE settings. When DONEPAUSEEN is set the DMA command operation is paused when this flag is asserted. Write '1' to this bit to clear it. Automatically cleared when the ENABLECMD is set.	W1C	0x0
[15:11]	-	Reserved, <b>RAZ/WI</b>	-	-
[10]	INTR_TRIGOUTACKWAIT	Channel is waiting for output Trigger Acknowledgment Interrupt Flag. This interrupt is set to HIGH if the INTREN_TRIGOUTACKWAIT is set and the STAT_TRIGOUTACKWAIT status flag is asserted. Automatically cleared when STAT_TRIGOUTACKWAIT is cleared.	RO	0x0
[9]	INTR_DESTRIGINWAIT	Channel is waiting for Destination Trigger Interrupt Flag. This interrupt is set to HIGH if the INTREN_DESTRIGINWAIT is set and the STAT_DESTRIGINWAIT status flag is asserted. Automatically cleared when STAT_DESTRIGINWAIT is cleared.	RO	0x0
[8]	INTR_SRCTRIGINWAIT	Channel is waiting for Source Trigger Interrupt Flag. This interrupt is set to HIGH if the INTREN_SRCTRIGINWAIT is set and the STAT_SRCTRIGINWAIT status flag is asserted. Automatically cleared when STAT_SRCTRIGINWAIT is cleared.	RO	0x0
[7:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3]	INTR_STOPPED	Stopped Interrupt Flag. This interrupt is set to HIGH if the INTREN_STOPPED is set and the STAT_STOPPED flag gets raised. Automatically cleared when STAT_STOPPED is cleared.	RO	0x0
[2]	INTR_DISABLED	Disabled Interrupt Flag. This interrupt is set to HIGH if the INTREN_DISABLED is set and the STAT_DISABLED flag gets raised. Automatically cleared when STAT_DISABLED is cleared.	RO	0x0
[1]	INTR_ERR	Error Interrupt Flag. This interrupt is set to HIGH if the INTREN_ERR is set and the STAT_ERR status flag gets raised. Automatically cleared when STAT_ERR is cleared.	RO	0x0
[0]	INTR_DONE	Done Interrupt Flag. This interrupt is set to HIGH if the INTREN_DONE is set and the STAT_DONE status flag gets raised. Automatically cleared when STAT_DONE is cleared.	RO	0x0

### 6.5.1.3 CH\_INTREN

The Channel Interrupt Enable register can enable the interrupt generation for internal events.

The content of this register can be updated during command linking by setting bit[2] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x008

## Type

RW

## Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-10: CH\_INTREN register bit descriptions**

Bits	Name	Description	Type	Default
[31:11]	-	Reserved, <b>RAZ/WI</b>	-	-
[10]	INTREN_TRIGOUTACKWAIT	Channel is waiting for output Trigger Acknowledgement Interrupt Enable. When set to HIGH, enables INTR_TRIGOUTACKWAIT to be set and raise an interrupt when STAT_TRIGOUTACKWAIT status flag is asserted. When set to LOW, it prevents INTR_TRIGOUTACKWAIT to be asserted. Currently pending interrupts are not affected by clearing this bit.	RW	0x0
[9]	INTREN_DESTRIGINWAIT	Channel is waiting for destination Trigger Interrupt Enable. When set to HIGH, enables INTR_DESTRIGINWAIT to be set and raise an interrupt when STAT_DESTRIGINWAIT status flag is asserted. When set to LOW, it prevents INTR_DESTRIGINWAIT to be asserted. Currently pending interrupts are not affected by clearing this bit.	RW	0x0
[8]	INTREN_SRCTRIGINWAIT	Channel is waiting for Source Trigger Interrupt Enable. When set to HIGH, enables INTR_SRCTRIGINWAIT to be set and raise an interrupt when STAT_SRCTRIGINWAIT status flag is asserted. When set to LOW, it prevents INTR_SRCTRIGINWAIT to be asserted. Currently pending interrupts are not affected by clearing this bit.	RW	0x0
[7:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3]	INTREN_STOPPED	Stopped Interrupt Enable. When set to HIGH, enables INTR_STOPPED to be set and raise an interrupt when STAT_STOPPED status flag is asserted. When set to LOW, it prevents INTR_STOPPED to be asserted. Currently pending interrupts are not affected by clearing this bit.	RW	0x0
[2]	INTREN_DISABLED	Disabled Interrupt Enable. When set to HIGH, enables INTR_DISABLED to be set and raise an interrupt when STAT_DISABLED status flag is asserted. When set to LOW, it prevents INTR_DISABLED to be asserted. Currently pending interrupts are not affected by clearing this bit.	RW	0x0
[1]	INTREN_ERR	Error Interrupt Enable. When set to HIGH, enables INTR_ERROR to be set and raise an interrupt when the STAT_ERR status flag is asserted. When set to LOW, it prevents INTR_ERR to be asserted. Currently pending interrupts are not affected by clearing this bit.	RW	0x0
[0]	INTREN_DONE	Done Interrupt Enable. When set to HIGH, enables the INTR_DONE to be set and raise an interrupt when the STAT_DONE status flag is asserted. When set to LOW, it prevents INTR_DONE to be asserted. Currently pending interrupts are not affected by clearing this bit.	RW	0x0

### 6.5.1.4 CH\_CTRL

The Channel Control register can be used to configure the type of the transfers and the resources needed by the currently executed DMA command. It also defines how the command shall behave when the command is complete.

The content of this register can be updated during command linking by setting bit[3] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x00C

##### Type

RW

##### Default

0x00200200

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-11: CH\_CTRL register bit descriptions**

Bits	Name	Description	Type	Default
[31:30]	-	Reserved, <b>RAZ/WI</b>	-	-
[29]	USESTREAM	Enable Stream Interface use for this command. <ul style="list-style-type: none"> <li>0: disable</li> <li>1: enable</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: CH_STREAM_EN	RW	0x0
[28]	USEGPO	Enable GPO use for this command. <ul style="list-style-type: none"> <li>0: disable</li> <li>1: enable</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: (CH_GPO_EN * GPO_WIDTH) > 0.	RW	0x0
[27]	USETRIGOUT	Enable Trigger Output use for this command. <ul style="list-style-type: none"> <li>0: disable</li> <li>1: enable</li> </ul>	RW	0x0

Bits	Name	Description	Type	Default
[26]	USEDESTTRIGIN	Enable Destination Trigger Input use for this command.e <ul style="list-style-type: none"> <li>0: disable</li> <li>1: enable</li> </ul>	RW	0x0
[25]	USESRCRTRIGIN	Enable Source Trigger Input use for this command: <ul style="list-style-type: none"> <li>0: disable</li> <li>1: enable</li> </ul>	RW	0x0
[24]	DONEPAUSEEN	Done pause enable. When set to HIGH the assertion of the STAT_DONE flag results in an automatic pause request for the current DMA operation. When the paused state is reached the STAT_RESUMEWAIT flag is also set. When set to LOW the assertion of the STAT_DONE does not pause the progress of the command and the next operation of the channel is started immediately after the STAT_DONE flag is set.	RW	0x0
[23:21]	DONETYPE	Done type selection. This field defines when the STAT_DONE status flag is asserted during the command operation. <ul style="list-style-type: none"> <li>000: STAT_DONE flag is not asserted for this command.</li> <li>001: End of a command, before jumping to the next linked command. (default)</li> <li>011: End of an autorestart cycle, before starting the next cycle.</li> <li>Others : Reserved.</li> </ul>	RW	0x1
[20:18]	REGRELOADTYPE	Automatic register reload type. Defines how the DMA command reloads initial values at the end of a DMA command before autorestarting, ending or linking to a new DMA command: <ul style="list-style-type: none"> <li>000: Reload Disabled.</li> <li>001: Reload source and destination size registers only.</li> <li>011: Reload source address only and all source and destination size registers.</li> <li>101: Reload destination address only and all source and destination size registers.</li> <li>111: Reload source and destination address and all source and destination size registers.</li> <li>Others: Reserved.</li> </ul> <p>NOTE: When CLEARCMD is set, the reloaded registers are also cleared.</p>	RW	0x0
[17:15]	-	Reserved, <b>RAZ/WI</b>	-	-
[14:12]	YTYPE	Operation type for Y direction: <ul style="list-style-type: none"> <li>000: "disable" - Only do 1D transfers. When HAS_2D is 0, meaning 2D capability is not supported in HW, the YTYPE is always "000".</li> <li>001: "continue" - Copy 2D data in a continuous manner from source area to the destination area by using the YSIZE registers. The copy stops when the source runs out of data or the destination runs out of space. Not supported when HAS_2D is 0 in HW.</li> <li>010: "wrap" - Wrap the 2D source area within the destination 2D area by starting to copy data from the beginning of the first source line to the remaining space in the destination area. If the destination area is smaller than the source area then the behavior is UNPREDICTABLE. Not supported when HAS_WRAP or HAS_2D is 0 in HW.</li> <li>011: Fill the remainder of the destination area with FILLVAL when the source area runs out of data. If the destination area is smaller than the source area then the behavior is UNPREDICTABLE. Not supported when HAS_WRAP or HAS_2D is 0 in HW.</li> <li>Others: Reserved The field is <b>RAZ/WI</b> when extended features are disabled for the channel.</li> </ul>	RW	0x0

Bits	Name	Description	Type	Default
[11:9]	XTYPE	<p>Operation type for X direction:</p> <ul style="list-style-type: none"> <li>000: “disable” - No data transfer takes place for this command. This mode can be used to create empty commands that wait for an event or set GPOs.</li> <li>001: “continue” - Copy data in a continuous manner from source to the destination. For 1D operations it is expected that SRCXSIZE is equal to DESXSIZE, other combinations result in UNPREDICTABLE behavior. For 2D operations, this mode can be used for simple 2D to 2D copy but it also allows the reshaping of the data like 1D to 2D or 2D to 1D conversions. If the DESXSIZE is smaller than SRCXSIZE then the read data from the current source line goes to the next destination line. If SRCXSIZE is smaller than DESXSIZE then the reads start on the next line and data is written to the remainder of the current destination line. Note: For 1D to 2D the SRCYSIZE for 2D to 1D conversion the DESYSIZE needs to be set to 1 when using this mode.</li> <li>010: “wrap” - Wrap source data within a destination line when the end of the source line is reached. Read starts again from the beginning of the source line and copied to the remainder of the destination line. If the DESXSIZE is smaller than SRCXSIZE then the behavior is UNPREDICTABLE. Not supported when HAS_WRAP is 0 in HW.</li> <li>011: “fill” - Fill the remainder of the destination line with FILLVAL when the end of the source line is reached. If the DESXSIZE is smaller than SRCXSIZE then the behavior is UNPREDICTABLE. Not supported when HAS_WRAP is 0 in HW.</li> <li>Others: Reserved.</li> </ul>	RW	0x1
[8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	CHPRIO	<p>Channel Priority.</p> <ul style="list-style-type: none"> <li>0: Lowest Priority</li> <li>15: Highest Priority</li> </ul>	RW	0x0
[3]	-	Reserved, <b>RAZ/WI</b>	-	-
[2:0]	TRANSIZE	<p>The width of this field depends on CoreLink DMA-350's DATA_WIDTH config parameter.</p> <p>If DATA_WIDTH is 128, bits[2:0] are present. In this case, SW can program all three bits, though the DMAC does not support TRANSIZE values larger than 128 bits. If DATA_WIDTH is 64 or 32, then bit[2] is RESERVED and only bits[1:0] are present. In this case, SW can program only the two least significant bits as the DMAC handles the most significant bit as <b>RAZ/WI</b>.</p> <p>Transfer Entity Size. Size in bytes = 2<sup>TRANSIZE</sup>.</p> <ul style="list-style-type: none"> <li>000: Byte</li> <li>001: Halfword</li> <li>010: Word</li> <li>011: Doubleword</li> <li>100: 128bits</li> <li>101: 256bits</li> <li>110: 512bits</li> <li>111: 1024bits</li> </ul> <p>Note that DATA_WIDTH limits this field. Address is aligned to TRANSIZE by the DMAC by ignoring the lower bits.</p>	RW	0x0

### 6.5.1.5 CH\_SRCADDR

The Channel Source Address register defines the 32-bit base address of the command to read the data from. When the register is read during the execution of the command it shows an approximate hint at the actual address the DMA channel is going to read from.

The content of this register can be updated during command linking by setting bit[4] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x010

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-12: CH\_SRCADDR register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	SRCADDR	Source Address [31:0].	RW	0x0

### 6.5.1.6 CH\_SRCADDRHI

The Channel Source Address Register High Bits [63:32] defines the upper 32 bits of the read address if more than 32-bit addressing is used in the system. When the register is read during the execution of the command it shows an approximate hint at the actual address the DMA channel is going to read from.

The content of this register can be updated during command linking by setting bit[5] in the command link header.

#### Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x014

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-13: CH\_SRCADDRHI register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	SRCADDRHI	Source Address [63:32]. Allows 64-bit addressing but the system might need less address bits defined by ADDR_WIDTH. The not implemented bits remain reserved. The field is <b>RAZ/WI</b> when the following condition is False: ADDR_WIDTH > 32	RW	0x0

### 6.5.1.7 CH\_DESADDR

The Channel Destination Address register defines the 32-bit base address of the command to write the data to. When the register is read during the execution of the command it shows an approximate hint at the actual address the DMA channel is going to write to.

The content of this register can be updated during command linking by setting bit[6] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x018

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-14: CH\_DESADDR register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	DESADDR	Destination Address[31:0]	RW	0x0

### 6.5.1.8 CH\_DESADDRHI

The Channel Destination Address Register, High Bits [63:32], defines the upper 32 bits of the write address if more than 32-bit addressing is used in the system. When the register is read during the execution of the command it shows an approximate hint at the actual address the DMA channel is going to write to.

The content of this register can be updated during command linking by setting bit[7] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x01C

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-15: CH\_DESADDRHI register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	DESADDRHI	Destination Address[63:32]. Allows 64-bit addressing but the system might need less address bits defined by ADDR_WIDTH. The not implemented bits remain reserved. The field is <b>RAZ/WI</b> when the following condition is False: ADDR_WIDTH > 32	RW	0x0



### 6.5.1.9 CH\_XSIZE

The Channel X Dimension Size Register, Lower Bits [15:0] register defines the number of data units copied during the DMA command up to 16 bits in the X dimension. The source and destination size of the command may be different for some transfer types. When read during the execution of the DMA command, the register shows an approximate hint of the remaining number of lines in both read and write directions.

The content of this register can be updated during command linking by setting bit[8] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x020

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-16: CH\_XSIZE register bit descriptions**

Bits	Name	Description	Type	Default
[31:16]	DESXSIZE	Destination Number of Transfers in the X Dimension lower bits [15:0]. This register along with DESXSIZEHI defines the destination data block size of the DMA operation for or any 1D operation, and defines the X dimension of the 2D destination block for a 2D operation. HAS_WRAP or HAS_STREAM configuration needs to be set to allow writes to this register, otherwise it is read-only and writing to SRCXSIZE also updates the value of this register.	RW	0x0
[15:0]	SRCXSIZE	Source Number of Transfers in the X Dimension lower bits [15:0]. This register along with SRCXSIZEHI defines the source data block size of the DMA operation for any 1D operation, and defines the X dimension of the 2D source block for 2D operation.	RW	0x0

### 6.5.1.10 CH\_XSIZEHI

The Channel X Dimension Size Register, High Bits [31:16] defines the number of data units copied during the DMA command up to 32 bits in the X dimension. The source and destination size of the command may be different for some transfer types. When read during the execution of the DMA

command, the register shows an approximate hint of the remaining number of lines in both read and write directions.

The content of this register can be updated during command linking by setting bit[9] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x024

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-17: CH\_XSIZEHI register bit descriptions**

Bits	Name	Description	Type	Default
[31:16]	DESXSIZEHI	Destination Number of Transfers in the X Dimension high bits [31:16]. This register along with DESXSIZE defines the destination data block size of the DMA operation for or any 1D operation, and defines the X dimension of the 2D destination block for a 2D operation. HAS_WRAP or HAS_STREAM configuration needs to be set to allow writes to this register, otherwise it is read-only and writing to SRCXSIZEHI also updates the value of this register.	RW	0x0
[15:0]	SRCXSIZEHI	Source Number of Transfers in the X Dimension high bits [31:16]. This register along with SRCXSIZE defines the source data block size of the DMA operation for any 1D operation, and defines the X dimension of the 2D source block for 2D operation.	RW	0x0

### 6.5.1.11 CH\_SRCTRANSCFG

The Channel Source Transfer Configuration register provides transfer attribute settings in the read direction of the DMA command.

The content of this register can be updated during command linking by setting bit[10] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x028

### Type

RW

### Default

0x000F0400

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-18: CH\_SRCTRANSCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:20]	-	Reserved, <b>RAZ/WI</b>	-	-
[19:16]	SRCMAXBURSTLEN	Source Max Burst Length. Hint for the DMA on what is the maximum allowed burst size it can use for read transfers. The maximum number of beats sent by the DMA for a read burst is equal to SRCMAXBURSTLEN + 1. Default value is 16 beats, which allows the DMA to set all burst sizes. Note: Limited by the DATA_BUFF_SIZE so larger settings may not always result in larger bursts.	RW	0xf
[15:12]	-	Reserved, <b>RAZ/WI</b>	-	-
[11]	SRCPRIVATTR	Source Transfer Privilege Attribute. <ul style="list-style-type: none"> <li>0: Unprivileged</li> <li>1: Privileged</li> </ul> When a channel is unprivileged this bit is tied to 0.	RW	0x0
[10]	SRCNONSECATTR	Source Transfer Non-secure Attribute. <ul style="list-style-type: none"> <li>0: Secure</li> <li>1: Non-secure</li> </ul> When a channel is Non-secure this bit is tied to 1.	RW	0x1
[9:8]	SRCSHAREATTR	Source Transfer Shareability Attribute. <ul style="list-style-type: none"> <li>00: Non-shareable</li> <li>01: Reserved</li> <li>10: Outer shareable</li> <li>11: Inner shareable</li> </ul>	RW	0x0

Bits	Name	Description	Type	Default
[7:4]	SRCMEMATTRHI	<p>Source Transfer Memory Attribute field [7:4].</p> <ul style="list-style-type: none"> <li>• 0000: Device memory</li> <li>• 0001: Normal memory, Outer Write allocate, Outer Write-through transient</li> <li>• 0010: Normal memory, Outer Read allocate, Outer Write-through transient</li> <li>• 0011: Normal memory, Outer Read/Write allocate, Outer Write-through transient</li> <li>• 0100: Normal memory, Outer non-cacheable</li> <li>• 0101: Normal memory, Outer Write allocate, Outer Write-back transient</li> <li>• 0110: Normal memory, Outer Read allocate, Outer Write-back transient</li> <li>• 0111: Normal memory, Outer Read/Write allocate, Outer Write-back transient</li> <li>• 1000: Normal memory, Outer Write-through non-transient</li> <li>• 1001: Normal memory, Outer Write allocate, Outer Write-through non-transient</li> <li>• 1010: Normal memory, Outer Read allocate, Outer Write-through non-transient</li> <li>• 1011: Normal memory, Outer Read/Write allocate, Outer Write-through non-transient</li> <li>• 1100: Normal memory, Outer Write-back non-transient</li> <li>• 1100: Normal memory, Outer Write-back non-transient</li> <li>• 1101: Normal memory, Outer Write allocate, Outer Write-back non-transient</li> <li>• 1110: Normal memory, Outer Read allocate, Outer Write-back non-transient</li> <li>• 1111: Normal memory, Outer Read/Write allocate, Outer Write-back non-transient</li> </ul>	RW	0x0

Bits	Name	Description	Type	Default
[3:0]	SRCMEMATTRLO	<p>Source Transfer Memory Attribute field [3:0].</p> <p>When SRCMEMATTRHI is Device type (0000) then this field means:</p> <ul style="list-style-type: none"> <li>0000: Device-nGnRnE</li> <li>0100: Device-nGnRE</li> <li>1000: Device-nGRE</li> <li>1100: Device-GRE</li> <li>Others: Invalid resulting in UNPREDICTABLE behavior.</li> </ul> <p>When SRCMEMATTRHI is Normal memory type (other than 0000) then this field means:</p> <ul style="list-style-type: none"> <li>0000: Reserved</li> <li>0001: Normal memory, Inner Write allocate, Inner Write-through transient</li> <li>0010: Normal memory, Inner Read allocate, Inner Write-through transient</li> <li>0011: Normal memory, Inner Read/Write allocate, Inner Write-through transient</li> <li>0100: Normal memory, Inner non-cacheable</li> <li>0101: Normal memory, Inner Write allocate, Inner Write-back transient</li> <li>0110: Normal memory, Inner Read allocate, Inner Write-back transient</li> <li>0111: Normal memory, Inner Read/Write allocate, Inner Write-back transient</li> <li>1000: Normal memory, Inner Write-through non-transient</li> <li>1001: Normal memory, Inner Write allocate, Inner Write-through non-transient</li> <li>1010: Normal memory, Inner Read allocate, Inner Write-through non-transient</li> <li>1011: Normal memory, Inner Read/Write allocate, Inner Write-through non-transient</li> <li>1011: Normal memory, Inner Read/Write allocate, Inner Write-through non-transient</li> <li>1100: Normal memory, Inner Write-back non-transient</li> <li>1101: Normal memory, Inner Write allocate, Inner Write-back non-transient</li> <li>1110: Normal memory, Inner Read allocate, Inner Write-back non-transient</li> <li>1111: Normal memory, Inner Read/Write allocate, Inner Write-back non-transient</li> </ul>	RW	0x0

### 6.5.1.12 CH\_DESTRANSCFG

The Channel Destination Transfer Configuration register provides transfer attribute settings in the write direction of the DMA command.

The content of this register can be updated during command linking by setting bit[11] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

### Offset

0x02C

### Type

RW

### Default

0x000F0400

### Usage constraints

Becomes read-only after ENABLECMD is set.

### Bit descriptions

**Table 6-19: CH\_DESTRANSCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:20]	-	Reserved, <b>RAZ/WI</b>	-	-
[19:16]	DESMAXBURSTLEN	Destination Max Burst Length. Hint for the DMA on what is the maximum allowed burst size it can use for write transfers. The maximum number of beats sent by the DMA for a write burst is equal to DESMAXBURSTLEN + 1. Default value is 16 beats, which allows the DMA to set all burst sizes. Note: Limited by the DATA_BUFF_SIZE so larger settings may not always result in larger bursts.	RW	0xf
[15:12]	-	Reserved, <b>RAZ/WI</b>	-	-
[11]	DESPRIVATTR	Destination Transfer Privilege Attribute. <ul style="list-style-type: none"> <li>0: Unprivileged</li> <li>1: Privileged</li> </ul> When a channel is unprivileged this bit is tied to 0.	RW	0x0
[10]	DESNONSECATTR	Destination Transfer Non-secure Attribute. <ul style="list-style-type: none"> <li>0: Secure</li> <li>1: Non-secure</li> </ul> When a channel is Non-secure this bit is tied to 1.	RW	0x1
[9:8]	DESSHAREATTR	Destination Transfer Shareability Attribute. <ul style="list-style-type: none"> <li>00: Non-shareable</li> <li>01: Reserved</li> <li>10: Outer shareable</li> <li>11: Inner shareable</li> </ul>	RW	0x0

Bits	Name	Description	Type	Default
[7:4]	DESMEMATTRHI	<p>Destination Transfer Memory Attribute field [7:4].</p> <ul style="list-style-type: none"> <li>• 0000: Device memory</li> <li>• 0001: Normal memory, Outer Write allocate, Outer Write-through transient</li> <li>• 0010: Normal memory, Outer Read allocate, Outer Write-through transient</li> <li>• 0011: Normal memory, Outer Read/Write allocate, Outer Write-through transient</li> <li>• 0100: Normal memory, Outer non-cacheable</li> <li>• 0101: Normal memory, Outer Write allocate, Outer Write-back transient</li> <li>• 0110: Normal memory, Outer Read allocate, Outer Write-back transient</li> <li>• 0111: Normal memory, Outer Read/Write allocate, Outer Write-back transient</li> <li>• 1000: Normal memory, Outer Write-through non-transient</li> <li>• 1001: Normal memory, Outer Write allocate, Outer Write-through non-transient</li> <li>• 1010: Normal memory, Outer Read allocate, Outer Write-through non-transient</li> <li>• 1011: Normal memory, Outer Read/Write allocate, Outer Write-through non-transient</li> <li>• 1100: Normal memory, Outer Write-back non-transient</li> <li>• 1101: Normal memory, Outer Write allocate, Outer Write-back non-transient</li> <li>• 1110: Normal memory, Outer Read allocate, Outer Write-back non-transient</li> <li>• 1111: Normal memory, Outer Read/Write allocate, Outer Write-back non-transient</li> </ul>	RW	0x0

Bits	Name	Description	Type	Default
[3:0]	DESMEMATTRLO	<p>Destination Transfer Memory Attribute field [3:0].</p> <p>When DESMEMATTRHI is Device type (0000) then this field means:</p> <ul style="list-style-type: none"> <li>• 0000: Device-nGnRnE</li> <li>• 0100: Device-nGnRE</li> <li>• 1000: Device-nGRE</li> <li>• 1100: Device-GRE</li> <li>• Others: Invalid resulting in UNPREDICTABLE behavior</li> </ul> <p>When DESMEMATTRHI is Normal Memory type (other than 0000) then this field means:</p> <ul style="list-style-type: none"> <li>• 0000: Reserved</li> <li>• 0001: Normal memory, Inner Write allocate, Inner Write-through transient</li> <li>• 0010: Normal memory, Inner Read allocate, Inner Write-through transient</li> <li>• 0011: Normal memory, Inner Read/Write allocate, Inner Write-through transient</li> <li>• 0100: Normal memory, Inner non-cacheable</li> <li>• 0101: Normal memory, Inner Write allocate, Inner Write-back transient</li> <li>• 0110: Normal memory, Inner Read allocate, Inner Write-back transient</li> <li>• 0111: Normal memory, Inner Read/Write allocate, Inner Write-back transient</li> <li>• 1000: Normal memory, Inner Write-through non-transient</li> <li>• 1001: Normal memory, Inner Write allocate, Inner Write-through non-transient</li> <li>• 1010: Normal memory, Inner Read allocate, Inner Write-through non-transient</li> <li>• 1011: Normal memory, Inner Read/Write allocate, Inner Write-through non-transient</li> <li>• 1100: Normal memory, Inner Write-back non-transient</li> <li>• 1101: Normal memory, Inner Write allocate, Inner Write-back non-transient</li> <li>• 1110: Normal memory, Inner Read allocate, Inner Write-back non-transient</li> <li>• 1111: Normal memory, Inner Read/Write allocate, Inner Write-back non-transient</li> </ul>	RW	0x0

### 6.5.1.13 CH\_XADDRINC

The Channel X Dimension Address Increment register sets the increment values used to update the source and destination addresses after each transferred data unit.

The content of this register can be updated during command linking by setting bit[12] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

#### Register frame

DMACH<n>



### Offset

0x030

### Type

RW

### Default

0x00000000

### Usage constraints

Becomes read-only after ENABLECMD is set.

### Bit descriptions

**Table 6-20: CH\_XADDRINC register bit descriptions**

Bits	Name	Description	Type	Default
[31:16]	DESXADDRINC	Destination X dimension Address Increment. This value is used as the increment between each TRANSIZE transfer. When a single bit is used then only 0 and 1 can be set. For wider increment registers, two's complement used with a range between -32768 to 32767 when the counter is 16 bits wide. The width of the register is indicated by the INC_WIDTH parameter. $DESADDR_{next} = DESADDR + 2^{TRANSIZE} * DESXADDRINC$	RW	0x0
[15:0]	SRCXADDRINC	Source X dimension Address Increment. This value is used as the increment between each TRANSIZE transfer. When a single bit is used then only 0 and 1 can be set. For wider increment registers, two's complement used with a range between -32768 to 32767 when the counter is 16 bits wide. The width of the register is indicated by the INC_WIDTH parameter. $SRCADDR_{next} = SRCADDR + 2^{TRANSIZE} * SRCXADDRINC$	RW	0x0

#### 6.5.1.14 CH\_YADDRSTRIDE

The Channel Y Dimension Address Stride register sets the increment values used to update the source and destination line base addresses after each line is transferred.

The content of this register can be updated during command linking by setting bit[13] in the command link header.

### Configurations

See bit descriptions.

### Attributes

#### Register frame

DMACH<n>

### Offset

0x034

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-21: CH\_YADDRSTRIDE register bit descriptions**

Bits	Name	Description	Type	Default
[31:16]	DESYADDRSTRIDE	Destination Address Stride between lines. Calculated in TRANSIZE aligned steps. This value is used to increment the DESADDR after completing the transfer of a destination line. $DESADDR\_next\_line\_base = DESADDR\_line\_base + 2^{TRANSIZE} * DESYADDRSTRIDE$ . Two's complement used with a range between -32768 to 32767. When set to 0 the DESADDR is not incremented after completing one line. Not present when HAS_2D is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0
[15:0]	SRCYADDRSTRIDE	Source Address Stride between lines. Calculated in TRANSIZE aligned steps. This value is used to increment the SRCADDR after completing the transfer of a source line. $SRCADDR\_next\_line\_base = SRCADDR\_line\_base + 2^{TRANSIZE} * SRCYADDRSTRIDE$ . Two's complement used with a range between -32768 to 32767. When set to 0 the SRCADDR is not incremented after completing one line. Not present when HAS_2D is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0

### 6.5.1.15 CH\_FILLVAL

The Channel Fill Pattern Value register provides a predefined value to be used to fill the remaining part of the destination memory area when the source side of the command is finished.

The content of this register can be updated during command linking by setting bit[14] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x038

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-22: CH\_FILLVAL register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	FILLVAL	Fill pattern value. When XTYPE or YTYPE is set to fill mode, then this register value is used on the write data bus when the command starts to fill the memory area. The TRANSIZE defines the width of the FILLVAL used for the command. For byte transfers the FILLVAL[7:0] is used, other bits are ignored. For halfword transfers the FILLVAL[15:0] is used, other bits are ignored. For 64-bit and wider transfers the FILLVAL[31:0] pattern is repeated on the full width of the data bus. Not present when HAS_WRAP is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0

### 6.5.1.16 CH\_YSIZE

The Channel Y Dimensions Size register defines the number of lines copied during the DMA command up to 16 bits in the Y dimension. The read and write size of the command may be different for some transfer types. When read during the execution of the DMA command, the register shows an approximate hint of the remaining number of lines in both read and write directions.

The content of this register can be updated during command linking by setting bit[15] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x03C

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-23: CH\_YSIZE register bit descriptions**

Bits	Name	Description	Type	Default
[31:16]	DESYSIZE	Destination Y dimension or number of lines. Not present when HAS_2D is 0. HAS_WRAP or HAS_STREAM configuration needs to be set to allow writes to this register, otherwise it is read-only. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0

Bits	Name	Description	Type	Default
[15:0]	SRCYSIZE	Source Y dimension or number of lines. Not present when HAS_2D is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0

### 6.5.1.17 CH\_TMPLTCFG

The Channel Template Configuration register provides configuration settings when using template pattern based copy operations.

The content of this register can be updated during command linking by setting bit[16] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x040

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-24: CH\_TMPLTCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:21]	-	Reserved, <b>RAZ/WI</b>	-	-
[20:16]	DESTMPLTSIZE	Destination Template Size in number of transfers plus one. <ul style="list-style-type: none"> <li>0: Destination template is disabled.</li> <li>1 to 31: CH_DESTMPLT. DESTMPLT is used as the destination template. Not present when HAS_TMPLT is 0.</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN.	RW	0x0
[15:13]	-	Reserved, <b>RAZ/WI</b>	-	-

Bits	Name	Description	Type	Default
[12:8]	SRCTMPLTSIZE	Source Template Size in number of transfers plus one. <ul style="list-style-type: none"> <li>0: Source template is disabled.</li> <li>1 to 31: CH_SRCTMPLT. SRCTMPLT is used as the source template. Not present when HAS_TMPLT is 0.</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0
[7:0]	-	Reserved, <b>RAZ/WI</b>	-	-

### 6.5.1.18 CH\_SRCTMPLT

The Channel Source Template Pattern register sets the template pattern used for reading the source memory area.

The content of this register can be updated during command linking by setting bit[17] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x044

##### Type

RW

##### Default

0x00000001

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-25: CH\_SRCTMPLT register bit descriptions**

Bits	Name	Description	Type	Default
[31:1]	SRCTMPLT	Source Packing Template. Bit[0] is read only and always set to 1 as template patterns can only start from the base address of the transfer. Not present when HAS_TMPLT is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0
[0]	SRCTMPLTLSB	Source Packing Template Least Significant Bit. This bit of the template is read only and always set to 1 as template patterns can only start from the base address of the transfer. Not present when HAS_TMPLT is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RO	CH_EXT_FEAT_EN

### 6.5.1.19 CH\_DESTMPLT

The Channel Destination Template Pattern register sets the template pattern used for writing the destination memory area.

The content of this register can be updated during command linking by setting bit[18] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x048

##### Type

RW

##### Default

0x00000001

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-26: CH\_DESTMPLT register bit descriptions**

Bits	Name	Description	Type	Default
[31:1]	DESTMPLT	Destination Packing Template. Bit[0] is read only and always set to 1 as template patterns can only start from the base address of the transfer. Not present when HAS_TMPLT is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RW	0x0
[0]	DESTMPLTLSB	Destination Packing Template Least Significant Bit. This bit of the template is read only and always set to 1 as template patterns can only start from the base address of the transfer. Not present when HAS_TMPLT is 0. The field is <b>RAZ/WI</b> when the following condition is False: CH_EXT_FEAT_EN	RO	CH_EXT_FEAT_EN

### 6.5.1.20 CH\_SRCTRIGINCFG

The Channel Source Trigger In Configuration register provides configuration settings when using source side trigger input for the current DMA command.

The content of this register can be updated during command linking by setting bit[19] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x04C

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-27: CH\_SRCTRIGINCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:24]	-	Reserved, <b>RAZ/WI</b>	-	-
[23:16]	SRCTRIGINBLKSIZE	Source Trigger Input Default Transfer Size. Defined transfer size per trigger + 1. The field is <b>RAZ/WI</b> when the following condition is False: (1) & (NUM_TRIGGER_IN > 0)	RW	0x0
[15:12]	-	Reserved, <b>RAZ/WI</b>	-	-
[11:10]	SRCTRIGINMODE	Source Trigger Input Mode: <ul style="list-style-type: none"> <li>00: Command</li> <li>01: Reserved</li> <li>10: DMA driven Flow control. Only allowed when HAS_TRIGIN is enabled.</li> <li>11: Peripheral driven Flow control. Only allowed when HAS_TRIGIN is enabled.</li> </ul> Note: This field is ignored for Internal triggers as they only support Command triggers.	RW	0x0
[9:8]	SRCTRIGINTYPE	Source Trigger Input Type: <ul style="list-style-type: none"> <li>00: Software only Trigger Request. SRCTRIGINSEL is ignored.</li> <li>01: Reserved</li> <li>10: HW Trigger Request. Only allowed when HAS_TRIGIN is enabled. SRCTRIGINSEL selects between external trigger inputs if HAS_TRIGSEL is enabled.</li> <li>11: Internal Trigger Request. Only allowed when HAS_TRIGSEL is enabled and the DMAC has multiple channels, otherwise treated as HW Trigger Request. SRCTRIGINSEL selects between DMA channels.</li> </ul> Note: SW triggers are also available when HW or Internal types are selected, but is is not recommended and caution must be taken when the these modes are combined.	RW	0x0
[7:0]	SRCTRIGINSEL	Source Trigger Input Select	RW	0x0

### 6.5.1.21 CH\_DESTRIGINCFG

The Channel Destination Trigger In Configuration register provides configuration settings when using destination side trigger input for the current DMA command.

The content of this register can be updated during command linking by setting bit[20] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x050

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-28: CH\_DESTRIGINCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:24]	-	Reserved, <b>RAZ/WI</b>	-	-
[23:16]	DESTRIGINBLKSIZE	Destination Trigger Input Default Transfer Size. Defined transfer size per trigger + 1. The field is <b>RAZ/WI</b> when the following condition is False: (1) & (NUM_TRIGGER_IN > 0)	RW	0x0
[15:12]	-	Reserved, <b>RAZ/WI</b>	-	-
[11:10]	DESTRIGINMODE	Destination Trigger Input Mode: <ul style="list-style-type: none"> <li>00: Command</li> <li>01: Reserved</li> <li>10: DMA driven Flow control. Only allowed when HAS_TRIGGER_IN is enabled.</li> <li>11: Peripheral driven Flow control. Only allowed when HAS_TRIGGER_IN is enabled.</li> </ul> Note: This field is ignored for Internal triggers as they only support Command triggers.	RW	0x0



Bits	Name	Description	Type	Default
[9:8]	DESTRIGINTYPE	Destination Trigger Input Type: <ul style="list-style-type: none"> <li>00: Software only Trigger Request. DESTRIGINSEL is ignored</li> <li>01: Reserved</li> <li>10: HW Trigger Request. Only allowed when HAS_TRIGIN is enabled. DESTRIGINSEL selects between external trigger inputs if HAS_TRIGSEL is enabled.</li> <li>11: Internal Trigger Request. Only allowed when HAS_TRIGSEL is enabled and the DMAC has multiple channels, otherwise treated as HW Trigger Request. DESTRIGINSEL selects between DMA channels.</li> </ul> Note: SW triggers are also available when HW or Internal types are selected, but is is not recommended and caution must be taken when the these modes are combined.	RW	0x0
[7:0]	DESTRIGINSEL	Destination Trigger Input Select	RW	0x0

### 6.5.1.22 CH\_TRIGOUTCFG

The Channel Trigger Out Configuration register provides configuration settings when using trigger output for the current DMA command.

The content of this register can be updated during command linking by setting bit[21] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x054

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-29: CH\_TRIGOUTCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:10]	-	Reserved, <b>RAZ/WI</b>	-	-

Bits	Name	Description	Type	Default
[9:8]	TRIGOUTTYPE	Trigger Output Type <ul style="list-style-type: none"> <li>00: Software only Trigger Acknowledgment.</li> <li>01: Reserved</li> <li>10: HW Trigger Acknowledgment. Only allowed when HAS_TRIGOUT is enabled.</li> <li>11: Internal Trigger Acknowledgment. Only allowed when HAS_TRIGSEL is enabled and the DMAC has multiple channels, otherwise treated as HW Trigger Acknowledgment.</li> </ul>	RW	0x0
[7:6]	-	Reserved, <b>RAZ/WI</b>	-	-
[5:0]	TRIGOUTSEL	Trigger Output Select The field is <b>RAZ/WI</b> when the following condition is False: (1) & (NUM_TRIGGER_OUT > 0) & (1)	RW	0x0

### 6.5.1.23 CH\_GPOEN0

The Channel GPO Driving Enable register 0 enables which GPO ports are enabled to change at the beginning of current DMA command. GPO ports from bit 0 to 31 can be enabled by this register if the port is available to this channel.

The content of this register can be updated during command linking by setting bit[22] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x058

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

**Table 6-30: CH\_GPOEN0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	GPOEN0	Channel General Purpose Output (GPO) bit 0 to 31 enable mask. If bit n is '1', then GPO[n] is selected for driving by GPOVAL0[n]. If bit 'n' is '0', then GPO[n] keeps its previous value. Only [GPO_WIDTH-1:0] are implemented. All unimplemented bits are <b>RAZ/WI</b> . The field is <b>RAZ/WI</b> when the following condition is False: (CH_GPO_EN * GPO_WIDTH) > 0	RW	0x0

### 6.5.1.24 CH\_GPOVAL0

The Channel GPO Value register 0 sets the value to be driven on the GPO ports that are enabled at the beginning of current DMA command. The value of the GPO ports from bit 0 to 31 can be adjusted by this register if the port is available to this channel.

The content of this register can be updated during command linking by setting bit[24] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x060

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after ENABLECMD is set.

#### Bit descriptions

Table 6-31: CH\_GPOVAL0 register bit descriptions

Bits	Name	Description	Type	Default
[31:0]	GPOVAL0	General Purpose Output Value GPO[31:0]. Write to set output value. The actual value on the GPO port becomes active when the command is enabled. Read returns the register value which might be different from the actual GPO port status. Only [GPO_WIDTH-1:0] are implemented. All unimplemented bits are <b>RAZ/WI</b> . The field is <b>RAZ/WI</b> when the following condition is False: (CH_GPO_EN * GPO_WIDTH) > 0	RW	0x0

### 6.5.1.25 CH\_STREAMINTCFG

The Channel Stream Interface Configuration register provides configuration settings for the stream interface used by the current DMA command.

The content of this register can be updated during command linking by setting bit[26] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x068

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-32: CH\_STREAMINTCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:11]	-	Reserved, <b>RAZ/WI</b>	-	-
[10:9]	STREAMTYPE	Stream Interface operation Type <ul style="list-style-type: none"> <li>01: Stream out only</li> <li>00: Stream in and out used.</li> <li>10: Stream in only</li> <li>11: Reserved The field is <b>RAZ/WI</b> when the following condition is False: CH_STREAM_EN</li> </ul>	RW	0x0
[8:0]	-	Reserved, <b>RAZ/WI</b>	-	-

### 6.5.1.26 CH\_LINKATTR

The Channel Link Address Memory Attributes register provides transfer attribute settings for the command link related read transfers. The security and privilege attributes cannot be adjusted, they match the attributes of the channel.

The content of this register can be updated during command linking by setting bit[28] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x070

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-33: CH\_LINKATTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:10]	-	Reserved, <b>RAZ/WI</b>	-	-
[9:8]	LINKSHAREATTR	Link Address Transfer Shareability Attribute. <ul style="list-style-type: none"> <li>00: Non-shareable</li> <li>01: Reserved</li> <li>10: Outer shareable</li> <li>11: Inner shareable</li> </ul>	RW	0x0
[7:4]	LINKMEMATTRHI	Link Address Read Transfer Memory Attribute field [7:4]. <ul style="list-style-type: none"> <li>0000: Device memory</li> <li>0001: Normal memory, Outer Write allocate, Outer Write-through transient</li> <li>0010: Normal memory, Outer Read allocate, Outer Write-through transient</li> <li>0011: Normal memory, Outer Read/Write allocate, Outer Write-through transient</li> <li>0100: Normal memory, Outer non-cacheable</li> <li>0101: Normal memory, Outer Write allocate, Outer Write-back transient</li> <li>0110: Normal memory, Outer Read allocate, Outer Write-back transient</li> <li>0111: Normal memory, Outer Read/Write allocate, Outer Write-back transient</li> <li>1000: Normal memory, Outer Write-through non-transient</li> <li>1001: Normal memory, Outer Write allocate, Outer Write-through non-transient</li> <li>1010: Normal memory, Outer Read allocate, Outer Write-through non-transient</li> <li>1011: Normal memory, Outer Read/Write allocate, Outer Write-through non-transient</li> <li>1100: Normal memory, Outer Write-back non-transient</li> <li>1101: Normal memory, Outer Write allocate, Outer Write-back non-transient</li> <li>1110: Normal memory, Outer Read allocate, Outer Write-back non-transient</li> <li>1111: Normal memory, Outer Read/Write allocate, Outer Write-back non-transient</li> </ul>	RW	0x0

Bits	Name	Description	Type	Default
[3:0]	LINKMEMATTRLO	<p>Link Address Read Transfer Memory Attribute field [3:0]. When LINKMEMATTRHI is Device type (0000) then this field means:</p> <ul style="list-style-type: none"> <li>0000: Device-nGnRnE</li> <li>0100: Device-nGnRE</li> <li>1000: Device-nGRE</li> <li>1100: Device-GRE</li> <li>Others: Invalid resulting in UNPREDICTABLE behavior</li> </ul> <p>When LINKMEMATTRHI is Normal memory type (other than 0000) then this field means:</p> <ul style="list-style-type: none"> <li>0000: Reserved</li> <li>0001: Normal memory, Inner Write allocate, Inner Write-through transient</li> <li>0010: Normal memory, Inner Read allocate, Inner Write-through transient</li> <li>0011: Normal memory, Inner Read/Write allocate, Inner Write-through transient</li> <li>0100: Normal memory, Inner non-cacheable</li> <li>0101: Normal memory, Inner Write allocate, Inner Write-back transient</li> <li>0110: Normal memory, Inner Read allocate, Inner Write-back transient</li> <li>0111: Normal memory, Inner Read/Write allocate, Inner Write-back transient</li> <li>1000: Normal memory, Inner Write-through non-transient</li> <li>1001: Normal memory, Inner Write allocate, Inner Write-through non-transient</li> <li>1010: Normal memory, Inner Read allocate, Inner Write-through non-transient</li> <li>1011: Normal memory, Inner Read/Write allocate, Inner Write-through non-transient</li> <li>1100: Normal memory, Inner Write-back non-transient</li> <li>1101: Normal memory, Inner Write allocate, Inner Write-back non-transient</li> <li>1110: Normal memory, Inner Read allocate, Inner Write-back non-transient</li> <li>1111: Normal memory, Inner Read/Write allocate, Inner Write-back non-transient</li> </ul>	RW	0x0

### 6.5.1.27 CH\_AUTOCFG

The Channel Automatic Command Restart Configuration register configures the automatic restart behavior of the currently running command.

The content of this register can be updated during command linking by setting bit[29] in the command link header.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x074

## Type

RW

## Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-34: CH\_AUTOCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:17]	-	Reserved, <b>RAZ/WI</b>	-	-
[16]	CMDRESTARTINFEN	Enable Infinite Automatic Command Restart. When set, CMDRESTARTCNT is ignored and the command is always restarted after it is completed, including output triggering if enabled and autoreloading the registers but it does not perform a link to the next command. This means that the infinite loop of automatic restarts can only be broken by DISABLECMD or STOPCMD. When CMDRESTARTINFEN is set to '0', then the autorestarting of a command depends on CMDRESTARTCNT and when that counter is set to 0 the autorestarting is finished. In this case the next linked command is read or the command is complete.	RW	0x0
[15:0]	CMDRESTARTCNT	Automatic Command Restart Counter. Defines the number of times automatic restarting occurs at end of DMA command. Auto restarting occurs after the command is completed, including output triggering if enabled and autoreloading the registers, but it only performs a link to the next command when CMDRESTARTCNT == 0. When CMDRESTARTCNT and CMDRESTARTINF are both set to '0', autorestart is disabled.	RW	0x0

### 6.5.1.28 CH\_LINKADDR

The Channel Link Address register sets the 32-bit address of the next element in a command link. When the command execution is finished and this register is set then the DMA channel starts loading the next command from this address.

The content of this register can be updated during command linking by setting bit[30] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x078

## Type

RW

## Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-35: CH\_LINKADDR register bit descriptions**

Bits	Name	Description	Type	Default
[31:2]	LINKADDR	Link Address Pointer [31:2]. The DMAC fetches the next command from this address if LINKADDREN is set. NOTE: Commands are fetched with the security and privilege attribute of the channel and cannot be adjusted for the command link reads.	RW	0x0
[1]	-	Reserved, <b>RAZ/WI</b>	-	-
[0]	LINKADDREN	Enable Link Address. When set to '1', the DMAC fetches the next command defined by LINKADDR. When set to '0', the DMAC returns to idle at the end of the current command. NOTE: the linked command fetched by the DMAC needs to clear this field to mark the end of the command chain. Otherwise it may result in an infinite loop of the same command.	RW	0x0

### 6.5.1.29 CH\_LINKADDRHI

The Channel Link Address Register, High Bits [63:32] sets the upper address bits of the next element in a command link.

The content of this register can be updated during command linking by setting bit[31] in the command link header.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x07C

### Type

RW

### Default

0x00000000

## Usage constraints

Becomes read-only after ENABLECMD is set.



## Bit descriptions

**Table 6-36: CH\_LINKADDRHI register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	LINKADDRHI	Link Address Pointer [63:32]. Allows 64-bit addressing but the system might need less address bits. Limited by ADDR_WIDTH and the not implemented bits remain reserved. The field is <b>RAZ/WI</b> when the following condition is False: ADDR_WIDTH > 32	RW	0x0

### 6.5.1.30 CH\_GPOREAD0

The Channel GPO Read Value register 0 shows the current value of the GPO ports from bit 0 to 31 that are available to this channel.

The values here can be different from the values driven to the GPO ports by this channel.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x080

### Type

RO

### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-37: CH\_GPOREAD0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	GPOREAD0	General Purpose Output Read Value for GPO[31:0]. Read returns the actual value of the GPO ports. Only [GPO_WIDTH-1:0] are implemented. All unimplemented bits are set to 0. The field is <b>RAZ/WI</b> when the following condition is False: (CH_GPO_EN * GPO_WIDTH) > 0	RO	0x0

### 6.5.1.31 CH\_WRKREGPTR

The Channel Working Register Pointer register can be used to select an internal work register of the DMA channel that is visible in the CH\_WRKREGVAL register.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x088

##### Type

RW

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-38: CH\_WRKREGPTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:4]	-	Reserved, <b>RAZ/WI</b>	-	-

Bits	Name	Description	Type	Default
[3:0]	WRKREGPTR	<p>Internal Working Register Pointer. L- -</p> <p>This pointer refers to the following:</p> <ul style="list-style-type: none"> <li>0: Reserved</li> <li>1: SRCADDR_INITIAL</li> <li>2: SRCADDRHI_INITIAL</li> <li>3: DESADDR_INITIAL</li> <li>4: DESADDRHI_INITIAL</li> <li>5: SRCXSIZEHI_INITIAL, SRCXSIZE_INITIAL</li> <li>6: DESXSIZEHI_INITIAL, DESXSIZE_INITIAL</li> <li>7: SRCADDR_LINEINITIAL</li> <li>8: SRCADDRHI_LINEINITIAL</li> <li>9: DESADDR_LINEINITIAL</li> <li>10: DESADDRHI_LINEINITIAL</li> <li>11: SRCYSIZE_INITIAL (HAS_2D only)</li> <li>12: DESYSIZE_INITIAL (HAS_2D only)</li> <li>Others: Reserved.</li> </ul> <p>Note: When the selected register is not supported by the DMA then it is read as 0 in the CH_WRKREGVAL register. For 1D modes the INITIAL and LINEINITIAL registers contain the same values.</p>	RW	0x0

### 6.5.1.32 CH\_WRKREGVAL

The Channel - Working Register Value register shows the internal value of a work register of the DMA channel selected by the CH\_WRKREGPTR register.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x08C

##### Type

RO

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-39: CH\_WRKREGVAL register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	WRKREGVAL	Internal Working Register Values. WRKREGPTR points to the register that is visible here. The values are only guaranteed to be stable for the current command when STAT_PAUSED is asserted, otherwise they depend on the current status of the channel which makes them <b>UNPREDICTABLE</b> .	RO	0x0

### 6.5.1.33 CH\_ERRINFO

The Channel Error Information register provides information about internal errors encountered during command execution by the DMA channel.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x090

### Type

RO

### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-40: CH\_ERRINFO register bit descriptions**

Bits	Name	Description	Type	Default
[31:16]	ERRINFO	Error information Register. Additional information for the error that is encountered by the DMA. The meaning of the bits are detailed in the Error Handling section.	RO	0x0
[15:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7]	STREAMERR	Stream Interface Error Flag. Set when the stream interface encountered an error. The field is <b>RAZ/WI</b> when the following condition is False: CH_STREAM_EN	RO	0x0
[6:5]	-	Reserved, <b>RAZ/WI</b>	-	-
[4]	TRIGOUTSELERR	Trigger Output Selection Error Flag. Set when the command selects a trigger output that is not allowed for this channel.	RO	0x0
[3]	DESTRIGINSELERR	Destination Trigger Input Selection Error Flag. Set when the command selects a destination trigger input that is not allowed for this channel.	RO	0x0
[2]	SRCTRIGINSELERR	Source Trigger Input Selection Error Flag. Set when the command selects a source trigger input that is not allowed for this channel.	RO	0x0

Bits	Name	Description	Type	Default
[1]	CFGERR	Configuration Error Flag. Set when the DMA command is enabled or a linked command is read but it is configured in a mode that is not supported by the implementation.	RO	0x0
[0]	BUSERR	Bus Error Flag. Set when the DMA encounters a bus error during data or command read transfers.	RO	0x0

### 6.5.1.34 CH\_IIDR

The Channel Implementation Identification register provides information about the revision of the product implementing the DMA channel.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x0C8

##### Type

RO

##### Default

IMPLEMENTATION DEFINED

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-41: CH\_IIDR register bit descriptions**

Bits	Name	Description	Type	Default
[31:20]	PRODUCTID	Indicates the product ID	RO	0x3a0
[19:16]	VARIANT	Indicates the major revision, or variant, of the product rxy identifier	RO	0x0
[15:12]	REVISION	Indicates the minor revision of the product rxy identifier	RO	0x0
[11:0]	IMPLEMENTER	<p>Contains the JEP106 code of the company that implemented the IP:</p> <ul style="list-style-type: none"> <li>[11:8]: JEP106 continuation code of implementer.</li> <li>[7]: Always 0.</li> <li>[6:0]: JEP106 identity code of implementer</li> </ul> <p>For Arm this field reads as 0x43B.</p>	RO	0x43b

### 6.5.1.35 CH\_AIDR

The Channel Architecture Identification register provides information about the architecture version supported by the DMA channel.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x0CC

##### Type

RO

##### Default

IMPLEMENTATION DEFINED

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-42: CH\_AIDR register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	ARCH_MAJOR_REV	Architecture Major Revision	RO	0x0
[3:0]	ARCH_MINOR_REV	Architecture Minor Revision	RO	0x0

### 6.5.1.36 CH\_ISSUECAP

Used for setting issuing capability threshold.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x0E8

## Type

RW

## Default

IMPLEMENTATION DEFINED

## Usage constraints

Becomes read-only after ENABLECMD is set.

## Bit descriptions

**Table 6-43: CH\_ISSUECAP register bit descriptions**

Bits	Name	Description	Type	Default
[31:3]	-	Reserved, <b>RAZ/WI</b>	-	-
[2:0]	ISSUECAP	ISSUECAP can be used by software to place a constraint on allowed issuing capability in both read and write direction.	RW	0x7

### 6.5.1.37 CH\_BUILDCFG0

The Channel Build Configuration and Capability register 0 contains the configuration parameters and capabilities of the DMA channel.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMACH<n>

### Offset

0x0F8

## Type

RO

## Default

IMPLEMENTATION DEFINED

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-44: CH\_BUILDCFG0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:30]	-	Reserved, <b>RAZ/WI</b>	-	-

Bits	Name	Description	Type	Default
[29:26]	INC_WIDTH	Width of the increment register = INC_WIDTH + 1.  When set to 0, then only 0 and 1 can be set as an increment. When larger values are used, then negative increments can be set using 2's complement: <ul style="list-style-type: none"> <li>INC_WIDTH = 0: 0..1</li> <li>INC_WIDTH = 1: -2..1</li> <li>INC_WIDTH = 2: -4..3</li> <li>INC_WIDTH = 3: -8..7</li> <li>...</li> <li>INC_WIDTH = 15: -32768..32767</li> </ul>	RO	0xf
[25]	-	Reserved, <b>RAZ/WI</b>	-	-
[24:22]	DATA_WIDTH	Data Width: <ul style="list-style-type: none"> <li>000: 8-bit</li> <li>001: 16-bit</li> <li>010: 32-bit</li> <li>011: 64-bit</li> <li>100: 128-bit</li> <li>101: 256-bit</li> <li>110: 512-bit</li> <li>111: 1024-bit</li> </ul>	RO	$\log_2(\text{DATA\_WIDTH} / 8)$
[21:16]	ADDR_WIDTH	Address Width in bits = ADDR_WIDTH + 1	RO	ADDR_WIDTH - 1
[15:8]	CMD_BUFF_SIZE	Command Buffer Size in command words - 1. Total Size = (CMD_BUFF_SIZE + 1) * 4 bytes.	RO	0
[7:0]	DATA_BUFF_SIZE	Data Buffer Size in entries - 1. Total Size = ((DATA_BUFF_SIZE + 1) * DATA_WIDTH / 8) bytes.	RO	(CH_<n>_FIFO_DEPTH) - 1

### 6.5.1.38 CH\_BUILDCFG1

The Channel Build Configuration and Capability register 1 contains the configuration parameters and capabilities of the DMA channel.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMACH<n>

##### Offset

0x0FC

##### Type

RO



## Default

### IMPLEMENTATION DEFINED

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-45: CH\_BUILDCFG1 register bit descriptions**

Bits	Name	Description	Type	Default
[31:26]	-	Reserved, <b>RAZ/WI</b>	-	-
[25:19]	GPO_WIDTH	General Purpose Output Width. 0 to 64. The field is <b>RAZ/WI</b> when the following condition is False: (CH_GPO_EN * GPO_WIDTH) > 0	RO	(CH_GPO_EN * GPO_WIDTH)
[18]	HAS_GPOSEL	Has shared GPO ports between channels.	RO	0x0
[17:13]	-	Reserved, <b>RAZ/WI</b>	-	-
[12]	HAS_STREAMSEL	Has selectable stream interfaces.	RO	0x0
[11]	HAS_STREAM	Has stream interface support.	RO	CH_STREAM_EN
[10]	HAS_WRKREG	Has internal register view capability.	RO	0x1
[9]	HAS_AUTO	Has automatic reload and restart capability.	RO	0x1
[8]	HAS_CMDLINK	Has command link list capability.	RO	0x1
[7]	HAS_TRIGSEL	Has selectable triggers.	RO	0x1
[6]	HAS_TRIGOUT	Has hardware trigger output port.	RO	NUM_TRIGGER_OUT > 0
[5]	HAS_TRIGIN	Has hardware trigger input ports.	RO	NUM_TRIGGER_IN > 0
[4]	HAS_TRIG	Has trigger capability.	RO	0x1
[3]	HAS_TMPLT	Has template based pack and unpack capability.	RO	CH_EXT_FEAT_EN
[2]	HAS_2D	Has 2D capability.	RO	CH_EXT_FEAT_EN
[1]	HAS_WRAP	Has wrap capability.	RO	CH_EXT_FEAT_EN
[0]	HAS_XSIZEHI	Has 32-bit XSIZE counters.	RO	0x1

## 6.5.2 DMANSECCTRL description

DMA Unit Non-secure Control Register Frame.

For an overview of the frame, and a list of constraints that apply to this block, see [DMANSECCTRL summary](#).

### 6.5.2.1 NSEC\_CHINTRSTATUS0

The Non-Secure Channel Interrupt Status register 0 shows the overall interrupt status of every Non-secure channel from Channel 0 to NUM\_CHANNELS - 1.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMANSECCTRL

### Offset

0x000

### Type

RO

### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-46: NSEC\_CHINTRSTATUS0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:NUM_CHANNELS]	-	Reserved	-	-
[NUM_CHANNELS-1:0]	CHINTRSTATUS0	Collated Non-secure Channel Interrupt flags for channel 0 to NUM_CHANNELS - 1.	RO	0x0

## 6.5.2.2 NSEC\_STATUS

The Non-secure Status register provides information about the overall status of the Non-secure channels.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMANSECCTRL

### Offset

0x008

### Type

RW

### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-47: NSEC\_STATUS register bit descriptions**

Bits	Name	Description	Type	Default
[31:20]	-	Reserved, <b>RAZ/WI</b>	-	-
[19]	STAT_ALLCHPAUSED	All Non-secure Channel Paused Status. Set to '1' whenever all Non-secure DMA channel is in paused or inactive state and the ALLCHPAUSE request is active. Not set when channels reach the paused state for other reasons. All Non-secure channels are forced to an immediate pause until ALLCHPAUSE request is asserted even if they were enabled after the request. Cleared by writing '1' to this bit which results in all Non-secure channels to resume their operation. The ALLCHPAUSE request is also cleared when this bit is cleared.	W1C	0x0
[18]	STAT_ALLCHSTOPPED	All Non-secure Channel Stopped Status. Set to '1' whenever all Non-secure DMA channels are in stopped or inactive state and the ALLCHSTOP request is active. Not set when channels reach the stopped state for other reasons. All Non-secure channels are forced to an immediate stopped state until ALLCHSTOP request is asserted even if they were enabled after the request. Cleared by writing '1' to this bit. The ALLCHSTOP request is also cleared when this bit is cleared.	W1C	0x0
[17]	STAT_ALLCHIDLE	All Non-secure Channel Idle Status. Set to '1' whenever all Non-secure DMA channel is in idle state after at least one channel was running. Cleared by writing '1' to this bit.	W1C	0x0
[16:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3]	INTR_ALLCHPAUSED	All Non-secure Channel Paused Interrupt Status. Set to '1' when NSECCTRL.STAT_ALLCHPAUSED is asserted and NSECCTRL.INTREN_ALLCHPAUSED = 1. Cleared automatically when STAT_ALLCHPAUSED is cleared.	RO	0x0
[2]	INTR_ALLCHSTOPPED	All Non-secure Channel Stopped Interrupt Status. Set to '1' when NSECCTRL.STAT_ALLCHSTOPPED is asserted and NSECCTRL.INTREN_ALLCHSTOPPED = 1. Cleared automatically when STAT_ALLCHSTOPPED is cleared.	RO	0x0
[1]	INTR_ALLCHIDLE	All Non-secure Channel Idle Interrupt Status. Set to '1' when NSECCTRL.STAT_ALLCHIDLE is asserted and NSECCTRL.INTREN_ALLCHIDLE = 1. Cleared automatically when STAT_ALLCHIDLE is cleared.	RO	0x0
[0]	INTR_ANYCHINTR	Combined Non-secure Channel Interrupt Flag. Set to '1' when any Non-secure channel has an interrupt request in CHINTRSTATUS0 and NSECCTRL.INTREN_ANYCHINTR = 1. Cleared automatically when the source of the channel interrupt is cleared.	RO	0x0

### 6.5.2.3 NSEC\_CTRL

The Non-secure Control registers can be used to adjust the interrupt generation and general behavior of the Non-secure channels.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMANSECCTRL

##### Offset

0x00C

## Type

RW

## Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-48: NSEC\_CTRL register bit descriptions**

Bits	Name	Description	Type	Default
[31:30]	DISMINPWR	Minimum Power state of the DMAC when at least one Non-secure channel is present. <ul style="list-style-type: none"> <li>00: OFF</li> <li>01: Retention</li> <li>10: ON</li> <li>Others: Reserved</li> </ul>	RW	0x0
[29]	IDLERETEN	Idle Channel Retention Enable. Allows retention for Non-secure channels that are enabled and waiting for an event in IDLE state. <ul style="list-style-type: none"> <li>0: disabled</li> <li>1: enabled</li> </ul>	RW	0x0
[28]	DBGHALTEN	Debug Halt Enabled. <ul style="list-style-type: none"> <li>0: the DMA ignores the halt request from an external debugger. Clearing this bit while halt is ongoing results in continuing the operation.</li> <li>1: the debugger request to halt the DMA is allowed.</li> </ul> <p>This field is common for Non-secure and Secure side of the DMA, but access to this register is limited by the DBGHALTNSRO register field.</p>	RW	0x0
[27]	DBGHALTNSRO	Debug Halt Enable Non-secure Read Only. <ul style="list-style-type: none"> <li>0: NSEC_CTRL.DBGHALTEN has read-write access.</li> <li>1: The NSEC_CTRL.DBGHALTEN register becomes read-only.</li> </ul> <p>This register is read-only for the Non-secure SW.</p>	RO	0x0
[26:10]	-	Reserved, <b>RAZ/WI</b>	-	-
[9]	ALLCHPAUSE	Non-secure All Channel Pause Request. When set to '1', all Non-secure channels get a pause request. Stays asserted until the channels are paused and the STAT_ALLCHPAUSED status flag is set. Cleared automatically when the STAT_ALLCHPAUSED status flag is cleared.	W1S	0x0
[8]	ALLCHSTOP	Non-secure All Channel Stop Request. When set to '1', all Non-secure channels get a stop request. Stays asserted until the STAT_ALLCHSTOPPED status flag is set. Cleared automatically when the STAT_ALLCHSTOPPED status flag is cleared.	W1S	0x0
[7:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3]	INTREN_ALLCHPAUSED	All Non-secure Channel Paused Interrupt Enable	RW	0x0
[2]	INTREN_ALLCHSTOPPED	All Non-secure Channels Stopped Interrupt Enable	RW	0x0
[1]	INTREN_ALLCHIDLE	All Non-secure Channel Idle Interrupt Enable	RW	0x0

Bits	Name	Description	Type	Default
[0]	INTREN_ANYCHINTR	Combined Non-secure Channel Interrupt Enable	RW	0x0

#### 6.5.2.4 NSEC\_CHPTR

The Non-secure Channel Pointer register is used to select one channel that needs to be configured through the following registers through the NSEC\_CHCFG.

##### Configurations

See bit descriptions.

##### Attributes

##### Register frame

DMANSECCTRL

##### Offset

0x014

##### Type

RW

##### Default

0x00000000

##### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

##### Bit descriptions

**Table 6-49: NSEC\_CHPTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:6]	-	Reserved, <b>RAZ/WI</b>	-	-
[5:0]	CHPTR	Non-secure Channel Pointer. Selects which channel settings can be adjusted by the following registers.	RW	0x0

#### 6.5.2.5 NSEC\_CHCFG

The Non-secure Channel Configuration register provides configuration fields for channel attributes for the channel selected by NSEC\_CHPTR.

##### Configurations

See bit descriptions.

##### Attributes

##### Register frame

DMANSECCTRL

**Offset**

0x018

**Type**

RW

**Default**

0x00000000

**Usage constraints**

Becomes read-only after ENABLECMD is set for the channel selected by CHPTR.

**Bit descriptions****Table 6-50: NSEC\_CHCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:18]	-	Reserved, <b>RAZ/WI</b>	-	-
[17]	CHPRIV	Non-secure Channel Privilege Enable. NSEC_CHPTR selects the channel. When set to '1' it allows the channel to send transfers marked as Privileged only. The configuration registers of the selected channel are also given privileged only access rights. When set to '0' the channel is only allowed to send unprivileged transfers and the channel registers can be accessed by both privileged and unprivileged register accesses.	RW	0x0
[16]	CHIDVLD	Non-secure Channel ID valid. NSEC_CHPTR selects the channel. Set to '1' to drive the channel ID value in CHID for all the transfers by the selected channel. The field is <b>RAZ/WI</b> when the following condition is False: CHID_WIDTH > 0	RW	0x0
[15:0]	CHID	Non-secure Channel ID value. NSEC_CHPTR selects the channel ID value to be read or written via this register. CHID_WIDTH limits this field, unused bits are <b>RAZ/WI</b> . The field is <b>RAZ/WI</b> when the following condition is False: CHID_WIDTH > 0	RW	0x0

### 6.5.2.6 NSEC\_STATUSPTR

The Non-secure Unit Status Pointer register can set a pointer to an internal status register that can show the global state of the Non-secure channels.

**Configurations**

See bit descriptions.

**Attributes****Register frame**

DMANSECCTRL

**Offset**

0x0F0

**Type**

RW

**Default**

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-51: NSEC\_STATUSPTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3:0]	NSECSTATUSPTR	<p>Non-secure DMA Unit status pointer used to select which status value to view using STATUSVALUE register.</p> <p>Pointer values are:</p> <ul style="list-style-type: none"> <li>0: Channel Enabled Status for channel numbers [31:0].</li> <li>1: Reserved</li> <li>2: Channel Stopped Status for channel numbers [31:0].</li> <li>3: Reserved</li> <li>4: Channel Paused Status for channel numbers [31:0].</li> <li>Others: Reserved.</li> </ul>	RW	0x0

### 6.5.2.7 NSEC\_STATUSVAL

The Non-secure Unit Status Value register shows the current value of a status register on Non-secure channels selected by the pointer in NSEC\_STATUSPTR.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMANSECCTRL

### Offset

0x0F4

### Type

RO

### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-52: NSEC\_STATUSVAL register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	NSECSTATUSVAL	Non-secure DMA Unit status value. Can be used for reading internal status values of the DMA Unit for debug purposes. Values shown here are dependent on STATUSPTR. Note that inputs are masked by security mapping before being presented here. This means that only status values mapped to Non-secure world are visible as non-Zero values. All unimplemented bits are <b>RAZWI</b> .	RO	0x0

## 6.5.2.8 NSEC\_SIGNALPTR

The Non-secure Unit Signal Pointer register can set a pointer to an internal register that shows the state of the Non-secure interfaces attached to the DMAC.

### Configurations

See bit descriptions.

### Attributes

#### Register frame

DMANSECCTRL

#### Offset

0x0F8

#### Type

RW

#### Default

0x00000000

### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-53: NSEC\_SIGNALPTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3:0]	NSECSIGNALPTR	Non-secure DMA Unit signal pointer used to select which inputs or outputs to view using SIGNALVAL register.  Pointer values, x, are: <ul style="list-style-type: none"> <li>0 to 7: Trigger Input Requests [31+32x : 32x]</li> <li>8 to 9: Trigger output Acknowledges [31+32(x-8) : 32(x-8)]</li> <li>10 to 11: GPO output value [31+32(x-10) : 32(x-10)], only present when HAS_GPOSEL is set</li> <li>Others: Reserved</li> </ul>	RW	0x0



### 6.5.2.9 NSEC\_SIGNALVAL

The Non-secure Unit Signal Value register shows the current value of the Non-secure interfaces selected by the pointer in NSEC\_SIGNALPTR.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMANSECCTRL

##### Offset

0x0FC

##### Type

RW

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-54: NSEC\_SIGNALVAL register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	NSECSIGNALVAL	<p>Non-secure DMA Unit signal status. Can be used for reading signal values of triggers and GPOs for debug. Values shown here are dependent on SIGNALPTR.</p> <p>Note that inputs are masked by security mapping before being presented here. This means that only signals mapped to Non-secure world are visible as non-zero values. Writing to this register has the following effect:</p> <ul style="list-style-type: none"> <li>Writing '1' to a Non-secure Trigger Input Request that are not selected by any DMA channel causes a deny response for it. This can attempt to clear an unwanted trigger input.</li> <li>Writing '1' to any Trigger Output Acknowledges status, GPO status and all Trigger Input Request already selected by any DMA channel is ignored.</li> </ul> <p>All unimplemented bits are <b>RAZWI</b>.</p>	W1C	0x0

### 6.5.3 DMASECCTRL description

DMA Unit Secure Control Register Frame.

For an overview of the frame, and a list of constraints that apply to this block, see [DMASECCTRL summary](#).

### 6.5.3.1 SEC\_CHINTRSTATUS0

The Secure Channel Interrupt Status register 0 shows the overall interrupt status of every Secure channel from Channel 0 to NUM\_CHANNELS - 1.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCTRL

##### Offset

0x000

##### Type

RO

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-55: SEC\_CHINTRSTATUS0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:NUM_CHANNELS]	-	Reserved	-	-
[NUM_CHANNELS-1:0]	CHINTRSTATUS0	Collated Secure Channel Interrupt flags for channel 0 to NUM_CHANNELS - 1. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RO	0x0

### 6.5.3.2 SEC\_STATUS

The Secure Status register provides information about the overall status of the Secure channels.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCTRL

##### Offset

0x008

## Type

RW

## Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-56: SEC\_STATUS register bit descriptions**

Bits	Name	Description	Type	Default
[31:20]	-	Reserved, <b>RAZ/WI</b>	-	-
[19]	STAT_ALLCHPAUSED	All Secure Channel Paused Status. Set to '1' whenever all Secure DMA channel is in paused or inactive state and the ALLCHPAUSE request is active. Not set when channels reach the paused state for other reasons. All Secure channels are forced to an immediate pause until ALLCHPAUSE request is asserted even if they were enabled after the request. Cleared by writing '1' to this bit which results in all Secure channels to resume their operation. The ALLCHPAUSE request is also cleared when this bit is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	W1C	0x0
[18]	STAT_ALLCHSTOPPED	All Secure Channel Stopped Status. Set to '1' whenever all Secure DMA channels are in stopped or inactive state and the ALLCHSTOP request is active. Not set when channels reach the stopped state for other reasons. All Secure channels are forced to an immediate stopped state until ALLCHSTOP request is asserted even if they were enabled after the request. Cleared by writing '1' to this bit. The ALLCHSTOP request is also cleared when this bit is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	W1C	0x0
[17]	STAT_ALLCHIDLE	All Secure Channel Idle Status. Set to '1' whenever all Secure DMA channel is in idle state after at least one channel was running. Cleared by writing '1' to this bit. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	W1C	0x0
[16:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3]	INTR_ALLCHPAUSED	All Secure Channel Paused Interrupt Status. Set to '1' when SECCTRL.STAT_ALLCHPAUSED is asserted and SECCTRL.INTREN_ALLCHPAUSED = 1. Cleared automatically when STAT_ALLCHPAUSED is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RO	0x0
[2]	INTR_ALLCHSTOPPED	All Secure Channel Stopped Interrupt Status. Set to '1' when SECCTRL.STAT_ALLCHSTOPPED is asserted and SECCTRL.INTREN_ALLCHSTOPPED = 1. Cleared automatically when STAT_ALLCHSTOPPED is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RO	0x0
[1]	INTR_ALLCHIDLE	All Secure Channel Idle Interrupt Status. Set to '1' when SECCTRL.STAT_ALLCHIDLE is asserted and SECCTRL.INTREN_ALLCHIDLE = 1. Cleared automatically when STAT_ALLCHIDLE is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RO	0x0
[0]	INTR_ANYCHINTR	Combined Secure Channel Interrupt Flag. Set to '1' when any Secure channel has an interrupt request in CHINTRSTATUS0 and SECCTRL.INTREN_ANYCHINTR = 1. Cleared automatically when the source of the channel interrupt is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RO	0x0

### 6.5.3.3 SEC\_CTRL

The Secure Control registers can be used to adjust the interrupt generation and general behavior of the Secure channels.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCTRL

##### Offset

0x00C

##### Type

RW

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-57: SEC\_CTRL register bit descriptions**

Bits	Name	Description	Type	Default
[31:30]	DISMINPWR	Minimum Power state of the DMAC when at least one Secure channel is present. <ul style="list-style-type: none"> <li>00: OFF</li> <li>01: Retention</li> <li>10: ON</li> <li>Others: Reserved</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0
[29]	IDLERETEN	Idle Channel Retention Enable. Allows retention for Secure channels that are enabled and waiting for an event in IDLE state. <ul style="list-style-type: none"> <li>0: disabled</li> <li>1: enabled</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT & (1)	RW	0x0
[28]	DBGHALTEN	Debug Halt Enabled. When set to '0', the DMA ignores the halt request from an external debugger. Clearing this bit while halt is ongoing results in continuing the operation. When set to '1', the debugger request to halt the DMA is allowed for all channels. This field is common for Non-secure and Secure side of the DMA, but control can be limited by the SEC_CTRL.DBGHALTNSRO register field. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0

Bits	Name	Description	Type	Default
[27]	DBGHALTNSRO	Debug Halt Enable Non-secure Read Only. When set to '1', the NSEC_CTRL.DBGHALTEN register becomes read-only. When set to '0', NSEC_CTRL.DBGHALTEN has read-write access. This register allows the Secure software to limit the Non-secure software adjusting the common DBGHALTEN register bit. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0
[26:10]	-	Reserved, <b>RAZ/WI</b>	-	-
[9]	ALLCHPAUSE	Secure All Channel Pause Request. When set to '1', all Secure channels get a pause request. Stays asserted until the channels are paused and the STAT_ALLCHPAUSED status flag is set. Cleared automatically when the STAT_ALLCHPAUSED status flag is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	W1S	0x0
[8]	ALLCHSTOP	Secure All Channel Stop Request. When set to '1', all Secure channels get a stop request. Stays asserted until the STAT_ALLCHSTOPPED status flag is set. Cleared automatically when the STAT_ALLCHSTOPPED status flag is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	W1S	0x0
[7:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3]	INTREN_ALLCHPAUSED	All Secure Channel Paused Interrupt Enable The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0
[2]	INTREN_ALLCHSTOPPED	All Secure Channels Stopped Interrupt Enable The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0
[1]	INTREN_ALLCHIDLE	All Secure Channel Idle Interrupt Enable The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0
[0]	INTREN_ANYCHINTR	Combined Secure Channel Interrupt Enable The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0

### 6.5.3.4 SEC\_CHPTR

The Secure Channel Pointer register is used to select one channel that needs to be configured through the following registers that have SEC\_CH prefix.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCTRL

##### Offset

0x014

##### Type

RW

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-58: SEC\_CHPTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:6]	-	Reserved, <b>RAZ/WI</b>	-	-
[5:0]	CHPTR	Secure Channel Pointer. Selects which channel settings can be adjusted by the following registers. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0

## 6.5.3.5 SEC\_CHCFG

The Secure Channel Configuration register provides configuration fields for channel attributes.

### Configurations

See bit descriptions.

### Attributes

#### Register frame

DMASECCTRL

#### Offset

0x018

#### Type

RW

#### Default

0x00000000

### Usage constraints

Becomes read-only after ENABLECMD is set for the channel selected by CHPTR.

## Bit descriptions

**Table 6-59: SEC\_CHCFG register bit descriptions**

Bits	Name	Description	Type	Default
[31:18]	-	Reserved, <b>RAZ/WI</b>	-	-
[17]	CHPRIV	Secure Channel Privilege Enable. SECCHPTR selects the channel. When set to '1' it allows the channel to send transfers marked as Privileged only. The configuration registers of the selected channel are also given privileged only access rights. When set to '0' the channel is only allowed to send unprivileged transfers and the channel registers can be accessed by both privileged and unprivileged register accesses. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0
[16]	CHIDVLD	Secure Channel ID valid. SECCHPTR selects the channel. Set to '1' to drive the channel ID value in CHID for all the transfers by the selected channel. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT & (CHID_WIDTH > 0)	RW	0x0
[15:0]	CHID	Secure Channel ID value. SECCHPTR selects the channel ID value to be read or written via this register. CHID_WIDTH limits this field, unused bits are <b>RAZ/WI</b> . The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT & (CHID_WIDTH > 0)	RW	0x0

### 6.5.3.6 SEC\_STATUSPTR

The Secure Unit Status Pointer register can set a pointer to an internal status register that can show the global state of the Secure channels.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCTRL

##### Offset

0x0F0

##### Type

RW

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-60: SEC\_STATUSPTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3:0]	SECSTATUSPTR	Secure DMA Unit status pointer used to select which status value to view using STATUSVALUE register.  Pointer values are: <ul style="list-style-type: none"> <li>0: Channel Enabled Status for channel numbers [31 : 0]</li> <li>1: Reserved</li> <li>2: Channel Stopped Status for channel numbers [31 : 0]</li> <li>3: Reserved</li> <li>4: Channel Paused Status for channel numbers [31 : 0].</li> <li>Others: Reserved.</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0

### 6.5.3.7 SEC\_STATUSVAL

The Secure Unit Status Value register shows the current value of a status register on Secure channels selected by the pointer in SEC\_STATUSPTR.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCTRL

##### Offset

0x0F4

##### Type

RO

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-61: SEC\_STATUSVAL register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	SECSTATUSVAL	Secure DMA Unit status value. Can be used for reading internal status values of the DMA Unit for debug purposes. Values shown here are dependent on STATUSPTR. Note that inputs are masked by security mapping before being presented here. This means that only status values mapped to Secure world are visible as non-Zero values.  All unimplemented bits are <b>RAZ/WI</b> . The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RO	0x0

### 6.5.3.8 SEC\_SIGNALPTR

The Secure Unit Signal Pointer register can set a pointer to an internal register that shows the state of the Secure interfaces attached to the DMAC.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCTRL



### Offset

0x0F8

### Type

RW

### Default

0x00000000

### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

### Bit descriptions

**Table 6-62: SEC\_SIGNALPTR register bit descriptions**

Bits	Name	Description	Type	Default
[31:4]	-	Reserved, <b>RAZ/WI</b>	-	-
[3:0]	SECSIGNALPTR	Secure DMA Unit signal pointer used to select which inputs to view using SIGNALVAL register.  Pointer values, x, are: <ul style="list-style-type: none"> <li>0 to 7: Trigger Input Requests [31+32x : 32x]</li> <li>8 to 9: Trigger output Acknowledges [31+32(x-8) : 32(x-8)]</li> <li>10 to 11: GPO output value [31+32(x-10) : 32(x-10)], only present when HAS_GPOSEL is set.</li> <li>Others: Reserved</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0

### 6.5.3.9 SEC\_SIGNALVAL

The Secure Unit Signal Value register shows the current value of the Secure interfaces selected by the pointer in SEC\_SIGNALPTR.

### Configurations

See bit descriptions.

### Attributes

#### Register frame

DMASECCTRL

#### Offset

0x0FC

#### Type

RW

#### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-63: SEC\_SIGNALVAL register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	SECSIGNALVAL	<p>Secure DMA Unit signal status.</p> <p>Can be used for reading signal values of triggers and GPOs for debug. Values shown here are dependent on SIGNALPTR.</p> <p>Note that inputs are masked by security mapping before being presented here. This means that only signal values mapped to Secure world are visible as non-zero values. Writing to this register has the following effect:</p> <ul style="list-style-type: none"> <li>Writing '1' to a Trigger Input Request that are not selected by any DMA channel causes a deny response for it. This can attempt to clear an unwanted trigger input.</li> <li>Writing '1' to any Trigger Output Acknowledges status, GPO value and all Trigger Input Request selected by any DMA channel is ignored.</li> </ul> <p>All unimplemented bits are <b>RAZ/WI</b>. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT</p>	W1C	0x0

## 6.5.4 DMASECCFG description

DMA Unit Security Configuration Register Frame.

For an overview of the frame, and a list of constraints that apply to this block, see [DMASECCFG summary](#).

### 6.5.4.1 SCFG\_CHSECO

The Secure Configuration Channel Security Mapping register is used to define the security attribute of each channel from Channel 0 to NUM\_CHANNELS - 1.

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMASECCFG

### Offset

0x000

### Type

RW

### Default

0x00000000

### Usage constraints

Becomes read-only after SEC\_CFG\_LCK is set.

### Bit descriptions

**Table 6-64: SCFG\_CHSECO register bit descriptions**

Bits	Name	Description	Type	Default
[31:NUM_CHANNELS]	-	Reserved	-	-
[NUM_CHANNELS-1:0]	SCFGCHSECO	Secure Configuration Channel Security Mapping for channel 0 to NUM_CHANNELS - 1. When [i] set to '1', CH[i] is Non-secure world, else Secure world. The NUM_CHANNELS parameter limits this field, unused bits are <b>RAZ/WI</b> . Value for bit[i] can only be changed if the selected channel is not enabled, so reading back the register content is needed to check the success of the change.  The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0

## 6.5.4.2 SCFG\_TRIGINSECO

The Secure Configuration Trigger Input Security Mapping register 0 is used to define the security attribute of each trigger input.

### Configurations

See bit descriptions.

### Attributes

#### Register frame

DMASECCFG

#### Offset

0x008

#### Type

RW

#### Default

0x00000000

### Usage constraints

Becomes read-only after SEC\_CFG\_LCK is set.

## Bit descriptions

**Table 6-65: SCFG\_TRIGINSEC0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	SCFGTRIGINSEC0	Secure Configuration Trigger Input Security Mapping. When [i] set to '1', Trigger Input <i> is Non-secure world, else Secure world. The NUM_TRIGGER_IN parameter limits this field, unused bits are <b>RAZ/WI</b> . Value for bit[i] can only be changed if the trigger input port is not in use, so reading back the register content is needed to check the success of the change. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT & (1) & (NUM_TRIGGER_IN > 0)	RW	0x0

### 6.5.4.3 SCFG\_TRIGOUTSEC0

The Secure Configuration Trigger Output Security Mapping register 0 is used to define the security attribute of each trigger output.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCFG

##### Offset

0x028

##### Type

RW

##### Default

0x00000000

#### Usage constraints

Becomes read-only after SEC\_CFG\_LCK is set.

## Bit descriptions

**Table 6-66: SCFG\_TRIGOUTSEC0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:0]	SCFGTRIGOUTSEC0	Secure Configuration Trigger Output Security Mapping. When [i] set to '1', Trigger Output <i> is Non-secure world, else Secure world. The NUM_TRIGGER_OUT parameter limits this field, unused bits are RAZWI. Value for bit[i] can only be changed if the trigger output port is not in use, so reading back the register content is needed to check the success of the change. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT & (1) & (NUM_TRIGGER_OUT > 0)	RW	0x0

#### 6.5.4.4 SCFG\_CTRL

The Secure Configuration Control register defines the behavior of the DMAC when security violation occurs and also allows the security configuration to be locked until the next reset.

### Configurations

See bit descriptions.

### Attributes

#### Register frame

DMASECCFG

#### Offset

0x040

#### Type

RW

#### Default

0x00000000

### Usage constraints

Becomes read-only after SEC\_CFG\_LCK is set.

### Bit descriptions

**Table 6-67: SCFG\_CTRL register bit descriptions**

Bits	Name	Description	Type	Default
[31]	SEC_CFG_LCK	Security Configuration Lock. When set to '1', only SCFG.STAT_SECACCVIO can be cleared, all other register fields in the this block become read-only. Once set to '1', this field can only be set back to '0' by reset. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RW	0x0
[30:2]	-	Reserved, <b>RAZ/WI</b>	-	-
[1]	RSPTYPE_SECACCVIO	Secure Access Violation response type configuration. <ul style="list-style-type: none"> <li>0: <b>RAZ/WI</b></li> <li>1: bus error</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT.	RW	0x0
[0]	INTREN_SECACCVIO	Secure Access Violation Interrupt Enable. <ul style="list-style-type: none"> <li>0: no interrupt</li> <li>1: interrupt raised for security violation</li> </ul> The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT.	RW	0x0

### 6.5.4.5 SCFG\_INTRSTATUS

The Secure Configuration Interrupt Status register shows the interrupt status for security access violations.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMASECCFG

##### Offset

0x044

##### Type

RW

##### Default

0x00000000

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-68: SCFG\_INTRSTATUS register bit descriptions**

Bits	Name	Description	Type	Default
[31:17]	-	Reserved, <b>RAZ/WI</b>	-	-
[16]	STAT_SECACCVIO	Secure Access Violation Status. Set to '1' when a security violation occurred. Write '1' to clear. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	W1C	0x0
[15:1]	-	Reserved, <b>RAZ/WI</b>	-	-
[0]	INTR_SECACCVIO	Secure Access Violation Interrupt Status. Set to '1' when SCFG.STAT_SECACCVIO is asserted and SCFG.INTREN_SECACCVIO = 1. Cleared automatically when STAT_SECACCVIO is cleared. The field is <b>RAZ/WI</b> when the following condition is False: SECEXT_PRESENT	RO	0x0

### 6.5.5 DMAINFO description

DMA Unit Information Register Frame.

For an overview of the frame, and a list of constraints that apply to this block, see [DMAINFO summary](#).

### 6.5.5.1 DMA\_BUILDCFG0

The DMA Build Configuration register 0 is used to provide information about the configuration parameters of the DMAC.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0B0

##### Type

RO

##### Default

IMPLEMENTATION DEFINED

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-69: DMA\_BUILDCFG0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:25]	-	Reserved, <b>RAZ/WI</b>	-	-
[24:20]	CHID_WIDTH	Channel ID Width. '0' means CHID is not present.	RO	CHID_WIDTH
[19]	-	Reserved, <b>RAZ/WI</b>	-	-
[18:16]	DATA_WIDTH	Data Width. <ul style="list-style-type: none"> <li>000: 8-bit</li> <li>001: 16-bit</li> <li>010: 32-bit</li> <li>011: 64-bit</li> <li>100: 128-bit</li> <li>101: 256-bit</li> <li>110: 512-bit</li> <li>111: 1024-bit</li> </ul>	RO	log2(DATA_WIDTH / 8)
[15:10]	ADDR_WIDTH	Address Width in bits = ADDR_WIDTH + 1	RO	ADDR_WIDTH - 1
[9:4]	NUM_CHANNELS	Number of Channels + 1 supported by the DMAC	RO	NUM_CHANNELS - 1
[3]	-	Reserved, <b>RAZ/WI</b>	-	-

Bits	Name	Description	Type	Default
[2:0]	FRAMETYPE	<p>Register Frame Type.</p> <ul style="list-style-type: none"> <li>000: Combined Frame</li> <li>001: Security Configuration Frame</li> <li>010: Secure Control Frame, or if TrustZone not implemented, Control Frame.</li> <li>011: Non-secure Control Frame</li> <li>100: DMA Channel Frame.</li> </ul> <p>Note that each frame replicates the Information fields.</p>	RO	0x0

### 6.5.5.2 DMA\_BUILDCFG1

The DMA Build Configuration register 1 is used to provide information about the configuration parameters of the DMAC.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0B4

##### Type

RO

##### Default

IMPLEMENTATION DEFINED

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-70: DMA\_BUILDCFG1 register bit descriptions**

Bits	Name	Description	Type	Default
[31:25]	-	Reserved, <b>RAZ/WI</b>	-	-
[24]	-	Reserved, <b>RAZ/WI</b>	-	-
[23:17]	-	Reserved, <b>RAZ/WI</b>	-	-
[16]	HAS_TRIGSEL	Has Selectable Trigger Support	RO	0x1
[15:9]	NUM_TRIGGER_OUT	Number of Triggers Outputs. '0' means that no output triggers are present. The field is <b>RAZ/WI</b> when the following condition is False: NUM_TRIGGER_OUT > 0	RO	NUM_TRIGGER_OUT



Bits	Name	Description	Type	Default
[8:0]	NUM_TRIGGER_IN	Number of Triggers Inputs. '0' means that no input triggers are present. The field is <b>RAZ/WI</b> when the following condition is False: NUM_TRIGGER_IN > 0	RO	NUM_TRIGGER_IN

### 6.5.5.3 DMA\_BUILDCFG2

The DMA Build Configuration register 2 is used to provide information about the configuration parameters of the DMAC.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0B8

##### Type

RO

##### Default

IMPLEMENTATION DEFINED

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-71: DMA\_BUILDCFG2 register bit descriptions**

Bits	Name	Description	Type	Default
[31:12]	-	Reserved, <b>RAZ/WI</b>	-	-
[11]	-	Reserved, <b>RAZ/WI</b>	-	-
[10]	-	Reserved, <b>RAZ/WI</b>	-	-
[9]	HAS_RET	Has Retention Support	RO	0x1
[8]	HAS_TZ	Has TrustZone Support	RO	SECEXT_PRESENT
[7]	HAS_GPOSEL	Has Shared and hence Selectable GPO Support.	RO	0x0
[6:0]	-	Reserved, <b>RAZ/WI</b>	-	-

### 6.5.5.4 IIDR

The Implementation Identification register provides information about the implementer of the DMAC.

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0C8

##### Type

RO

##### Default

IMPLEMENTATION DEFINED

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-72: IIDR register bit descriptions**

Bits	Name	Description	Type	Default
[31:20]	PRODUCTID	Indicates the product ID	RO	0x3a0
[19:16]	VARIANT	Indicates the major revision, or variant, of the product rxy identifier	RO	0x0
[15:12]	REVISION	Indicates the minor revision of the product rxy identifier	RO	0x0
[11:0]	IMPLEMENTER	<p>Contains the JEP106 code of the company that implemented the IP:</p> <ul style="list-style-type: none"> <li>[11:8]: JEP106 continuation code of implementer.</li> <li>[7]: Always 0.</li> <li>[6:0]: JEP106 identity code of implementer.</li> </ul> <p>For Arm this field reads as 0x43B.</p>	RO	0x43b

### 6.5.5.5 AIDR

The Architecture Identification register provides information about the architecture revision supported by the DMAC.

#### Configurations

See bit descriptions.

## Attributes

### Register frame

DMAINFO

### Offset

0x0CC

### Type

RO

### Default

IMPLEMENTATION DEFINED

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-73: AIDR register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	ARCH_MAJOR_REV	Architecture Major Revision.	RO	0x0
[3:0]	ARCH_MINOR_REV	Architecture Minor Revision.	RO	0x0

### 6.5.5.6 PIDR4

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

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMAINFO

### Offset

0x0D0

### Type

RO

### Default

IMPLEMENTATION DEFINED

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-74: PIDR4 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	SIZE	4KB Count - the number of 4K pages used.< <ul style="list-style-type: none"> <li>0x00: 4K</li> <li>0x01: 8K</li> <li>0x02: 16K</li> <li>0x03: 32K</li> </ul>	RO	ceil(NUM_CHANNELS / 16)
[3:0]	DES_2	JEP106 Continuation Code	RO	0x4

### 6.5.5.7 PIDR0

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

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMAINFO

### Offset

0x0E0

### Type

RO

### Default

IMPLEMENTATION DEFINED

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-75: PIDR0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:0]	PART_0	Part Number [7:0]	RO	0xA0

### 6.5.5.8 PIDR1

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

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0E4

##### Type

RO

##### Default

IMPLEMENTATION DEFINED

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-76: PIDR1 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	DES_0	JEP106 Identity Code [3:0]	RO	0xb
[3:0]	PART_1	Part Number [11:8].	RO	0x3

### 6.5.5.9 PIDR2

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

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0E8

##### Type

RO

## Default

IMPLEMENTATION DEFINED

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-77: PIDR2 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	REVISION	Revision Code	RO	0x0
[3]	JEDEC	JEDEC	RO	0x1
[2:0]	DES_1	JEP106 Identity Code [6:4]	RO	0x3

### 6.5.5.10 PIDR3

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

## Configurations

See bit descriptions.

## Attributes

### Register frame

DMAINFO

### Offset

0x0EC

### Type

RO

### Default

0x00000000

## Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

## Bit descriptions

**Table 6-78: PIDR3 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	REVAND	Manufacturer revision number	RO	0x0
[3:0]	CMOD	Customer Modified	RO	0x0

### 6.5.5.11 CIDR0

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

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0F0

##### Type

RO

##### Default

0x0000000D

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-79: CIDR0 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:0]	PRMBL_0	Preamble	RO	0xd

### 6.5.5.12 CIDR1

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

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0F4

##### Type

RO

### Default

0x000000F0

### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

### Bit descriptions

**Table 6-80: CIDR1 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:4]	CLASS	Component class	RO	0xf
[3:0]	PRMBL_1	Preamble	RO	0x0

### 6.5.5.13 CIDR2

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

### Configurations

See bit descriptions.

### Attributes

#### Register frame

DMAINFO

#### Offset

0x0F8

#### Type

RO

#### Default

0x00000005

### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

### Bit descriptions

**Table 6-81: CIDR2 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:0]	PRMBL_2	Preamble	RO	0x5



### 6.5.5.14 CIDR3

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

#### Configurations

See bit descriptions.

#### Attributes

##### Register frame

DMAINFO

##### Offset

0x0FC

##### Type

RO

##### Default

0x000000B1

#### Usage constraints

There are no usage constraints apart from what is described for the block as a whole.

#### Bit descriptions

**Table 6-82: CIDR3 register bit descriptions**

Bits	Name	Description	Type	Default
[31:8]	-	Reserved, <b>RAZ/WI</b>	-	-
[7:0]	PRMBL_3	Preamble	RO	0xb1

# Appendix A Signal descriptions

This appendix contains all signal descriptions.

## CLK AND RESET

### Connection

Clock and reset controller

**Table A-1: Clock and reset signals**

Signal	Direction	Description
clk	input	Clock input.
resetsn	input	Active-LOW reset. Reset can go LOW asynchronously but must go HIGH synchronously.
aclken_m0	input	AXI5 Clock enable for M0 interface. Inputs sampled when HIGH, outputs are stable when LOW.
aclken_m1	input	AXI5 Clock enable for M1 interface. Inputs sampled when HIGH, outputs are stable when LOW.
pclken	input	APB4 Clock enable. Inputs sampled when HIGH, outputs are stable when LOW.

## Q-CHANNEL

### Connection

Clock controller

**Table A-2: Q-Channel signals**

Signal	Direction	Description
clk_qreqn	input	This signal indicates when the clock controller issues a clock quiescence entry or exit request to the DMAC.
clk_qacceptn	output	This signal indicates when the DMAC accepts the clock quiescence request.
clk_qdeny	output	This signal indicates when the DMAC denies the clock quiescence request.
clk_qactive	output	This signal indicates when the DMAC is active and also when it requests to exit from clock quiescence.

## P-CHANNEL

### Connection

Power controller

**Table A-3: P-Channel signals**

Signal	Direction	Description
preq	input	This signal indicates when the power controller issues a power state change request to the DMAC.
pstate[3:0]	input	This signal indicates the requested power state for the DMAC.
paccept	output	This signal indicates when the DMAC accepts the power state change request.
pdeny	output	This signal indicates when the DMAC denies the power state change request.

Signal	Direction	Description
pactive[9:0]	output	<p>This signal indicates which DMAC power states are active.</p> <p>The following bits can be asserted by the DMAC:</p> <ul style="list-style-type: none"> <li>[8] - ON state</li> <li>[5] - FULL_RET state</li> </ul> <p>All other bits are tied to 0.</p>

## APB4

### Connection

APB4 interconnect

**Table A-4: APB4 signals**

Signal	Direction	Description
psel	input	Select. It indicates that the subordinate device is selected and that a data transfer is required.
penable	input	Enable. This signal indicates the second and subsequent cycles of an APB4 transfer.
pprot[2:0]	input	Protection type. This signal indicates the normal, privileged, or Secure protection level of the transaction and whether the transaction is a data access or an instruction access.
pwrite	input	Direction. This signal indicates an APB Write-Access when HIGH and an APB4 read access when LOW.
paddr[12:0]	input	Address. This is the APB4 address bus.
pwdata[31:0]	input	Write data. This bus is driven during write cycles when pwrite is HIGH.
pstrb[3:0]	input	Write strobes. This signal indicates which byte lanes to update during a write transfer. Write strobes must not be active during a read transfer. Note that individual byte lane update is not supported, that is, pstrb is expected to be either 4'b0 or 4'b1, otherwise an error response is sent.
pready	output	Ready. The subordinate uses this signal to extend an APB4 transfer.
pslverr	output	This signal indicates a transfer failure.
prdata[31:0]	output	Read Data. The selected subordinate drives this bus during read cycles when pwrite is LOW.
pwakeup	input	Pending APB4 activity indicator, used as an input to clock and power controllers to wake the DMAC up.
pdebug	input	Additional sideband signal to identify debugger access. Valid together with PSEL.

## AXI5 M0

### Connection

AXI5 interconnect

**Table A-5: AXI5 M0 signals**

Signal	Direction	Description
awakeup_m0	output	Pending AXI5 activity indicator
awvalid_m0	output	Write address valid signal.
awready_m0	input	Write address ready signal.
awaddr_m0[ADDR_WIDTH-1:0]	output	Write address signal.
awburst_m0[1:0]	output	Write burst type signal.
awid_m0[ID_WIDTH-1:0]	output	Write request ID signal.

Signal	Direction	Description
awlen_m0[7:0]	output	Write burst length signal.
awsize_m0[2:0]	output	Write burst size signal.
awqos_m0[3:0]	output	Write quality-of-service signal.
awprot_m0[2:0]	output	Write protection type signal.
awcache_m0[3:0]	output	Indicates how transactions are required to progress through a system.
awdomain_m0[1:0]	output	Indicates the Shareability domain of a write transaction
awinner_m0[3:0]	output	Write address User signal for inner domain cache attributes for writes on the M0 interface.
awchid_m0[CHID_WIDTH-1:0]	output	Write address User signal for SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
awchidvalid_m0	output	Write address User signal for the validity of the SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
arvalid_m0	output	Read address valid signal.
aready_m0	input	Read address ready signal.
araddr_m0[ADDR_WIDTH-1:0]	output	Read address signal.
arburst_m0[1:0]	output	Read burst type signal.
arid_m0[ID_WIDTH-1:0]	output	Read request ID signal.
arlen_m0[7:0]	output	Read address burst length signal.
arsize_m0[2:0]	output	Read burst size signal.
arqos_m0[3:0]	output	Read quality-of-service signal.
arprot_m0[2:0]	output	Read protection type signal.
arcache_m0[3:0]	output	Indicates how transactions are required to progress through a system.
ardomain_m0[1:0]	output	Indicates the Shareability domain of a read transaction
arinner_m0[3:0]	output	Read address User signal for inner domain cache attributes for reads on the M0 interface.
archid_m0[CHID_WIDTH-1:0]	output	Read address User signal for SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
archidvalid_m0	output	Read address User signal for the validity of the SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
arcmdlink_m0	output	Read address User signal for indicating that the current read operation is a command link read.
wvalid_m0	output	Write data valid signal.
wready_m0	input	Write data ready signal.
wlast_m0	output	Indicates last transfer in a write burst.
wstrb_m0[STRB_WIDTH-1:0]	output	Write byte lane strobes.
wdata_m0[DATA_WIDTH-1:0]	output	Write data signal.
rvalid_m0	input	Read data valid signal.
rready_m0	output	Read data ready signal.
rid_m0[ID_WIDTH-1:0]	input	Read data ID.
rlast_m0	input	Indicates last transfer in read data.
rdata_m0[DATA_WIDTH-1:0]	input	Read data.
rpoison_m0[POIS_WIDTH-1:0]	input	Read data poison signal.
rresp_m0[1:0]	input	Read data response.
bvalid_m0	input	Write response valid signal.

Signal	Direction	Description
bready_m0	output	Write response ready signal.
bid_m0[ID_WIDTH-1:0]	input	Write response ID signal.
bresp_m0[1:0]	input	Write response signal.

## AXI5 M1

This whole group is not present when AXI5\_M1\_PRESENT is set to 0.

### Connection

AXI5 interconnect

**Table A-6: AXI5 M1 signals**

Signal	Direction	Description
awakeup_m1	output	Pending AXI5 activity indicator
awvalid_m1	output	Write address valid signal.
awready_m1	input	Write address ready signal.
awaddr_m1[ADDR_WIDTH-1:0]	output	Write address signal.
awburst_m1[1:0]	output	Write burst type signal.
awid_m1[ID_WIDTH-1:0]	output	Write request ID signal.
awlen_m1[7:0]	output	Write burst length signal.
awsize_m1[2:0]	output	Write burst size signal.
awqos_m1[3:0]	output	Write quality-of-service signal.
awprot_m1[2:0]	output	Write protection type signal.
awcache_m1[3:0]	output	Indicates how transactions are required to progress through a system.
awdomain_m1[1:0]	output	Indicates the Shareability domain of a write transaction
awinner_m1[3:0]	output	Write address User signal for inner domain cache attributes for writes on the M1 interface.
awchid_m1[CHID_WIDTH-1:0]	output	Write address User signal for SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
awchidvalid_m1	output	Write address User signal for the validity of the SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
arvalid_m1	output	Read address valid signal.
arready_m1	input	Read address ready signal.
araddr_m1[ADDR_WIDTH-1:0]	output	Read address signal.
arburst_m1[1:0]	output	Read burst type signal.
arid_m1[ID_WIDTH-1:0]	output	Read request ID signal.
arlen_m1[7:0]	output	Read address burst length signal.
arsize_m1[2:0]	output	Read burst size signal.
arqos_m1[3:0]	output	Read quality-of-service signal.
arprot_m1[2:0]	output	Read protection type signal.
arcache_m1[3:0]	output	Indicates how transactions are required to progress through a system.
ardomain_m1[1:0]	output	Indicates the Shareability domain of a read transaction
arinner_m1[3:0]	output	Read address User signal for inner domain cache attributes for reads on the M1 interface.

Signal	Direction	Description
archid_m1[CHID_WIDTH-1:0]	output	Read address User signal for SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
archidvalid_m1	output	Read address User signal for the validity of the SW configurable channel ID indication. Not present when CHID_WIDTH is set to 0.
arcmdlink_m1	output	Read address User signal for indicating that the current read operation is a command link read.
wvalid_m1	output	Write data valid signal.
wready_m1	input	Write data ready signal.
wlast_m1	output	Indicates last transfer in a write burst.
wstrb_m1[STRB_WIDTH-1:0]	output	Write byte lane strobes.
wdata_m1[DATA_WIDTH-1:0]	output	Write data signal.
rvalid_m1	input	Read data valid signal.
rready_m1	output	Read data ready signal.
rid_m1[ID_WIDTH-1:0]	input	Read data ID.
rlast_m1	input	Indicates last transfer in read data.
rdata_m1[DATA_WIDTH-1:0]	input	Read data.
rpoison_m1[POIS_WIDTH-1:0]	input	Read data poison signal.
rresp_m1[1:0]	input	Read data response.
bvalid_m1	input	Write response valid signal.
bready_m1	output	Write response ready signal.
bid_m1[ID_WIDTH-1:0]	input	Write response ID signal.
bresp_m1[1:0]	input	Write response signal.

## TRIGGER

The number of trigger ports are set base on NUM\_TRIGGER\_IN and NUM\_TRIGGER\_OUT. The selected trigger port type is not present on the entity when these are set to 0.

### Connection

Trigger capable peripherals

**Table A-7: Trigger signals**

Signal	Direction	Description
trig_in_<TI>_req	input	Trigger input request for trigger port TI.
trig_in_<TI>_req_type[1:0]	input	Trigger input request type for trigger port TI.
trig_in_<TI>_ack	output	Trigger input acknowledge for trigger port TI.
trig_in_<TI>_ack_type[1:0]	output	Trigger input acknowledge type trigger port TI.
trig_out_<TO>_req	output	Trigger output request for channel TO.
trig_out_<TO>_ack	input	Trigger output acknowledge for channel TO.

## IRQ

### Connection

Interrupt controller

**Table A-8: Interrupt request signals**

Signal	Direction	Description
irq_channel[ <b>NUM_CHANNELS</b> -1:0]	output	Individual interrupt signal for every channel. Interrupt sources are defined in the register map
irq_comb_nonsec	output	Common Non-secure channel interrupt for functional interrupt sources only.
irq_comb_sec	output	Common Secure channel interrupt interrupt for functional interrupt sources only. Not present when <b>SECEXT_PRESENT</b> is set to 0.
irq_sec_viol_err	output	Interrupt request for Secure register access violations. Not present when <b>SECEXT_PRESENT</b> is set to 0.

## STREAM

### Connection

Stream capable HW accelerator

**Table A-9: Stream signals**

Signal	Direction	Description
str_out_<N>_tvalid	output	Channel N.stream out interface transfer valid indication
str_out_<N>_tready	input	Channel N stream out interface transfer ready indication which shows that the subordinate can accept the transfer.
str_out_<N>_tdata[ <b>DATA_WIDTH</b> -1:0]	output	Channel N stream out interface data bus.
str_out_<N>_tstrb[ <b>STRB_WIDTH</b> -1:0]	output	Channel N stream out interface byte qualifier that indicates whether the content of the associated byte of tdata is processed as a data byte or a position byte.
str_out_<N>_tlast	output	Channel N stream out interface packet boundary indication.
str_in_<N>_tvalid	input	Channel N.stream in interface transfer valid indication
str_in_<N>_tready	output	Channel N stream in interface transfer ready indication which shows that the DMAC can accept the transfer.
str_in_<N>_tdata[ <b>DATA_WIDTH</b> -1:0]	input	Channel N stream in interface data bus.
str_in_<N>_tstrb[ <b>STRB_WIDTH</b> -1:0]	input	Channel N stream in interface byte qualifier that indicates whether the content of the associated byte of tdata is processed as a data byte or a position byte.
str_in_<N>_tlast	input	Channel N stream in interface packet boundary indication.
str_in_<N>_flush	output	This signal is a hint to the stream engine that the following data transactions are thrown away by the DMAC until the packet is finished with [tlast].{signal}.

## Status/Control

### Connection

System control

**Table A-10: Status and Control signals**

Signal	Direction	Description
gpo_ch_<N>[ <b>GPO_WIDTH</b> -1:0]	output	General purpose output for channel N. Not present when <b>CH_GPO_MASK</b> bit N is set to 0.
allch_stop_req_nonsec	input	All channel stop request for Non-secure channels. When asserted, all Non-secure channels that are not in IDLE are stopped. Channels can be enabled by SW, but are immediately stopped if this request is asserted.

Signal	Direction	Description
allch_stop_ack_nonsec	output	All channel stopped acknowledge for Non-secure channels. When asserted, all Non-secure channels are stopped or inactive. Four-phase handshake pair of allch_stop_req_nonsec.
allch_pause_req_nonsec	input	All channel pause request for Non-secure channels. When asserted, then all Non-secure channels that are not in IDLE are paused. Channels can be enabled by SW, but are immediately paused if this request is asserted. When the request is deasserted, the operation continues.
allch_pause_ack_nonsec	output	All channel pause acknowledge for Non-secure channels. When asserted, all Non-secure channels are paused or inactive. Four-phase handshake pair of allch_pause_req_nonsec.
allch_stop_req_sec	input	All channel stop request for Secure channels. When asserted, all Secure channels that are not in IDLE are stopped. Channels can be enabled by SW, but are immediately stopped if this request is asserted. Not present when SECEXT_PRESENT is set to 0.
allch_stop_ack_sec	output	All channel stopped acknowledge for Secure channels. When asserted, all Secure channels are stopped or inactive. Four-phase handshake pair of allch_stop_req_sec. Not present when SECEXT_PRESENT is set to 0.
allch_pause_req_sec	input	All channel pause request for Secure channels. When asserted, then all Secure channels that are not in IDLE are paused. Channels can be enabled by SW, but are immediately paused if this request is asserted. When the request is deasserted, the operation continues. Not present when SECEXT_PRESENT is set to 0.
allch_pause_ack_sec	output	All channel pause acknowledge for Secure channels. When asserted, all Secure channels are paused or inactive. Four-phase handshake pair of allch_pause_req_sec. Not present when SECEXT_PRESENT is set to 0.
halt_req	input	Level input to pause all operations and go into a halted state for both Secure and Non-secure channels. Compatible with the Cross Trigger Interface.
restart_req	input	Pulse input to resume the operation from the halted state. Compatible with the Cross Trigger Interface.
halted	output	Pulse indication that the DMAC reached the halted state. Compatible with the Cross Trigger Interface.
ch_enabled[NUM_CHANNELS-1:0]	output	Status indication per channel to show when a channel is active. When SECEXT_PRESENT is set to 1, then channels must be separated based on the ch_nonsec signals.
ch_err[NUM_CHANNELS-1:0]	output	Status indication per channel to show when a channel has encountered an error. When SECEXT_PRESENT is set to 1, then channels must be separated based on the ch_nonsec signals.
ch_stopped[NUM_CHANNELS-1:0]	output	Status indication per channel to show when the channel is stopped. When SECEXT_PRESENT is set to 1, then channels must be separated based on the ch_nonsec signals.
ch_paused[NUM_CHANNELS-1:0]	output	Status indication per channel to show when the channel is paused. When SECEXT_PRESENT is set to 1, then channels must be separated based on the ch_nonsec signals.
ch_priv[NUM_CHANNELS-1:0]	output	Status indication per channel to show the privilege setting of the channel. When SECEXT_PRESENT is set to 1, then channels must be separated based on the ch_nonsec signals.
ch_nonsec[NUM_CHANNELS-1:0]	output	Status indication per channel to show the security setting of the channel. Not present when SECEXT_PRESENT is set to 0.

## CONFIG

### Connection

Static configuration values



**Table A-11: Configuration signals**

Signal	Direction	Description
boot_en	input	Enables the channel 0 to load a first command after reset from the specified input address set by boot_addr.
boot_addr[ADDR_WIDTH-1:2]	input	word-aligned address for the boot process. When SECEXT_PRESENT is set to 1, then the boot_addr must point to a Secure memory region because channel 0 boots as Secure.
boot_memattr[7:0]	input	Boot address memory attributes. Has the same encoding as the LINKMEMATTRHI and LINKMEMATTRLO registers combined.
boot_shareattr[1:0]	input	Boot address memory Shareability attributes. Has the same encoding as the LINKSHAREATTR register.

# Appendix B Revisions

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

**Table B-1: Issue 0000-01**

Change	Location
First release	-

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

Change	Location
Master has been changed to manager and slave to subordinate	Throughout the document
AXI bandwidth utilization	<a href="#">AXI bandwidth utilization</a>
Cross Trigger Interface	<a href="#">Cross Trigger Interface</a>
Command execution status reporting	<a href="#">Command execution status reporting</a>
Added Warm reset power state	<a href="#">LPI power P-Channel</a> and <a href="#">Power P-Channel</a>
Content in Programming considerations	<a href="#">Programming considerations</a>

**Table B-3: Differences between issue 0000-02 and 0000-03**

Change	Location
Name of relevant configuration added	<a href="#">Configurable options</a>
Rewrite of section	<a href="#">AXI bandwidth utilization</a>
Addition of sections	<a href="#">Example of an unaligned start address and</a> <a href="#">Example of an unaligned start address</a>
Change to Non-secure and Secure DMA level interrupt signal sources	<a href="#">Interrupt operation</a>
Additional information added and change to figure	<a href="#">Programming considerations</a>
Memory map table updated	<a href="#">Memory map</a>
Description of channels rewritten	<a href="#">SCFG_CHSEC0</a>
Description of channels rewritten	<a href="#">SEC_CHINTRSTATUS0</a>
Description of channels rewritten	<a href="#">NSEC_CHINTRSTATUS0</a>

**Table B-4: Differences between issue 0000-03 and 0000-04**

Change	Location
Removed typo	<a href="#">Interworking with Trigger Interface</a>