HB0739
Handbook
CoreAXI4DMAController v2.1

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

50200739. 2.0 2/21

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 2.0

The following is a summary of the changes in revision 2.0 of this document.

- Updated the document for CoreAXI4DMAController v2.1.
- Updated section Supported Families, page 2.
- Updated section Utilization and Performance, page 3.
- Updated the following tables.
  - Table 11, page 20
  - Table 12, page 25
  - Table 19, page 30
- Added Notes below Table 13, page 27, Table 19, page 30, Table 21, page 31, and Table 27, page 34.
- Removed section Obfuscation, page 40.
- Added section RTL, page 40.
- Updated section SmartDesign, page 40.
- Replaced Figure 11, page 40 and Figure 12, page 41.

1.2 Revision 1.0

The first publication of this document, created for CoreAXI4DMAController v2.0.

2 Introduction

CoreAXI4DMAController is an AXI4 DMA controller designed to perform memory to memory style DMA (Direct Memory Access) transfers in an AXI system. The core provides in-built flow control techniques to ensure that the bandwidth of the AXI interface is optimally utilized. In addition, the core provides a bridge to AXI4 memory-mapped slaves for AXI4-Stream masters.

2.1 Features

Following are the key features of CoreAXI4DMAController:

- AXI4-Lite slave control interface
- AXI4 master DMA interface
- AXI4-Stream slave interface to provide a bridge to AXI4 memory map
- Maximum transfer size of 8 MB
- Maximum operating frequency of approximately 200 MHz
- Circular buffer DMA support
- Scatter-gather DMA support
- 2 internal 4 KB (maximum size) store and forward caches
- 1-4 interrupt outputs
- 4-32 internal descriptors
- External descriptor fetching support
- Fixed priority arbiter for DMA requests with configurable number of priority levels
- Configurable DMA bus width from 32- to 512-bit
- Prevents AXI4 transfers from crossing 4 KB boundaries[1]

2.2 Core Version

This handbook provides information on CoreAXI4DMAController version 2.1.

2.3 Supported Families

- PolarFire® SoC
- PolarFire®
- RTG4™
- IGLOO®2
- SmartFusion®2

1. The AXI4 protocol defines the smallest allocatable slave size as 4 KB. 4 KB boundary protection is required to prevent masters from generating sequential transactions that cross 4 KB boundaries to prevent sequential transactions spanning multiple slaves. Since the address information is sent up-front in an AXI transaction, if a transaction were to span a 4 KB boundary, the slave would be unaware of the type and configuration of the AXI transaction taking place.

2.4 Utilization and Performance

Utilization and performance data are listed in the following tables for their respective device families. The data listed in the following tables are indicative only. The overall device utilization and performance of the core is system dependent.

Table 1 • **CoreAXI4DMAController Device Utilization and Performance for PolarFire SoC Family (AXI4-Stream Disabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
PolarFire SoC (32-bit)	2743	4047	6790	203.25
PolarFire SoC (64-bit)	2975	4213	7188	202.76
PolarFire SoC (128-bit)	3434	4614	8048	198.26
PolarFire SoC (256-bit)	4308	6088	10396	194.33
PolarFire SoC (512-bit)	6088	7080	13168	197.43

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 2 • **CoreAXI4DMAController Device Utilization and Performance for PolarFire SoC Family (AXI4-Stream Enabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
PolarFire SoC (32-bit)	3503	5642	9145	197.01
PolarFire SoC (64-bit)	3891	5992	9883	183.35
PolarFire SoC (128-bit)	4588	6788	11376	184.95
PolarFire SoC (256-bit)	5941	8210	14151	177.49
PolarFire SoC (512-bit)	8735	11212	19947	192.6

The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 3 • **CoreAXI4DMAController Device Utilization and Performance for PolarFire Family (AXI4-Stream Disabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
PolarFire (32-bit)	2743	4047	6790	203.25
PolarFire (64-bit)	2975	4213	7188	202.76
PolarFire (128-bit)	3434	4614	8048	198.26
PolarFire (256-bit)	4308	6088	10396	194.33
PolarFire (512-bit)	6088	7080	13168	197.43

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 4 • **CoreAXI4DMAController Device Utilization and Performance for PolarFire Family (AXI4-Stream Enabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
PolarFire (32-bit)	3503	5642	9145	197.01
PolarFire (64-bit)	3891	5992	9883	183.35
PolarFire (128-bit)	4588	6788	11376	184.95
PolarFire (256-bit)	5941	8210	14151	177.49
PolarFire (512-bit)	8735	11212	19947	192.6

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 5 • **CoreAXI4DMAController Device Utilization and Performance for RTG4 Family (AXI4-Stream Disabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
RTG4 (32-bit)	2974	4198	7172	196.309
RTG4 (64-bit)	3212	4465	7677	190.512
RTG4 (128-bit)	3689	4953	8462	185.322
RTG4 (256-bit)	4628	5754	10382	181.422
RTG4 (512-bit)	6449	7343	13792	185.357

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 6 • **CoreAXI4DMAController Device Utilization and Performance for RTG4 Family (AXI4-Stream Enabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
RTG4 (32-bit)	3748	5672	9420	181.061
RTG4 (64-bit)	4119	6101	10220	173.883
RTG4 (128-bit)	5251	7181	12432	164.76
RTG4 (256-bit)	6384	8261	14645	155.642
RTG4 (512-bit)	9249	11053	20302	165.207

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 7 • **CoreAXI4DMAController Device Utilization and Performance for SmartFusion2 Family (AXI4-Stream Disabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
SmarFusion2 (32-bit)	2964	4187	7151	236.967
SmarFusion2 (64-bit)	3211	4431	7642	243.072
SmarFusion2 (128-bit)	3693	4963	8656	229.621
SmarFusion2 (256-bit)	4634	5777	10411	233.318
SmarFusion2 (512-bit)	6455	7345	13800	226.193

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 8 • **CoreAXI4DMAController Device Utilization and Performance for SmartFusion2 Family (AXI4-Stream Enabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
SmarFusion2 (32-bit)	3736	5634	9370	239.808
SmarFusion2 (64-bit)	4125	6061	10186	234.797
SmarFusion2 (128-bit)	5257	7172	12429	228.333
SmarFusion2 (256-bit)	6389	8283	14672	221.877
SmarFusion2 (512-bit)	9255	11014	20269	211.416

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 9 • **CoreAXI4DMAController Device Utilization and Performance for IGLOO2 Family (AXI4-Stream Disabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
IGLOO2 (32-bit)	2964	4187	7151	236.967
IGLOO2 (64-bit)	3211	4431	7642	243.072
IGLOO2 (128-bit)	3693	4963	8656	229.621
IGLOO2 (256-bit)	4634	5777	10411	233.318
IGLOO2 (512-bit)	6455	7345	13800	226.193

Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

Table 10 • **CoreAXI4DMAController Device Utilization and Performance for IGLOO2 Family (AXI4-Stream Enabled)**

	Logic Elements			
Family	**Sequential**	**Combinatorial**	**Total**	**Performance (MHz)**
IGLOO2 (32-bit)	3736	5634	9370	239.808
IGLOO2 (64-bit)	4125	6061	10186	234.797
IGLOO2 (128-bit)	5257	7172	12429	228.333
IGLOO2 (256-bit)	6389	8283	14672	221.877
IGLOO2 (512-bit)	9255	11014	20269	211.416

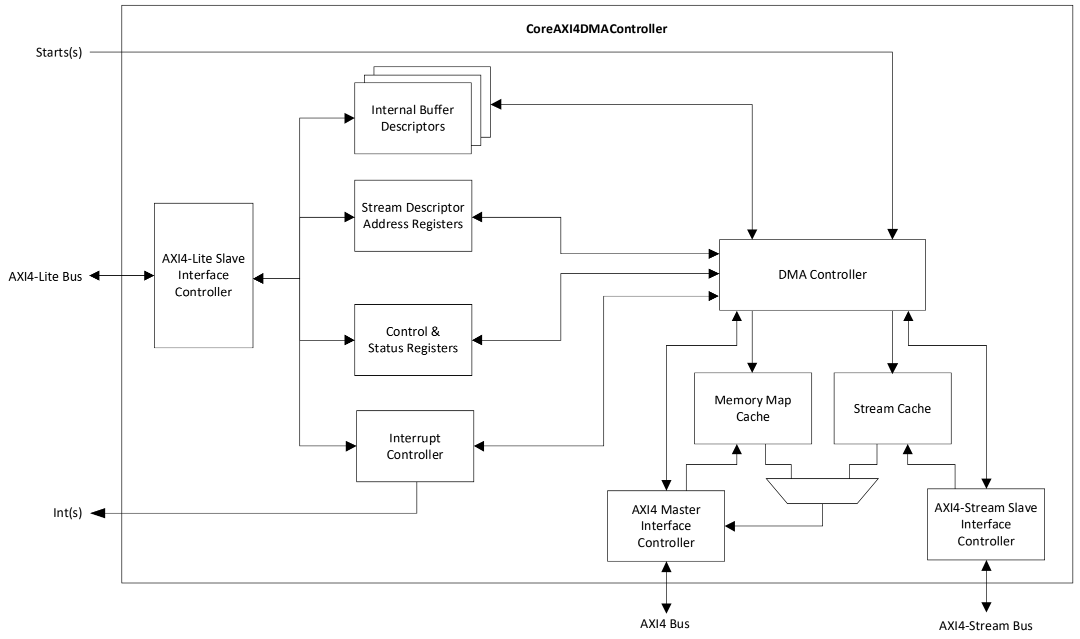
Note: The data in this table is achieved using Verilog RTL, with the following synthesis and layout settings (Timing-driven mode, high-effort) on a -1 speed grade part.

3 Functional Description

3.1 Architecture

Figure 1 displays a high-level view of the internal architecture of CoreAXI4DMAController.

Figure 1 • **CoreAXI4DMAController Internal Architecture**


CoreAXI4DMAController
Starts(s)
Internal Buffer Descriptors
Stream Descriptor Address Registers
AXI4-Lite Slave Interface Controller
AXI4-Lite Bus
DMA Controller
Control & Status Registers
Memory Map Cache
Stream Cache
Interrupt Controller
Int(s)
AXI4 Master Interface Controller
AXI4-Stream Slave Interface Controller
AXI4 Bus
AXI4-Stream Bus

3.1.1 AXI4-Lite Slave Interface Controller

The AXI4-Lite slave interface controller is responsible for translating AXI4-Lite write and read transactions into the internal register interface protocol. This provides access to the internal registers (including buffer descriptors) from an AXI4-Lite master, allowing AXI4-Lite masters to initiate and configure DMA transfers.

3.1.2 Control and Status Registers

The Control and Status Registers block contains the *Version* register to relay the major, minor, and build number of the core to controlling masters along with providing a register for initiating DMA operations.

3.1.3 Buffer Descriptors

The internal buffer descriptors contain the information required to configure a DMA operation including the start and destination addresses, the type of DMA operation and the number of bytes to be transferred. The number of internal descriptors instantiated is configurable through parameter, in the range 4 – 32. Internal descriptors are stored in LSRAM and can be chained together to perform scatter-gather or circular buffer DMA operations. In addition, internal descriptors can be used to point at external descriptors.

3.1.4 DMA Controller

CoreAXI4DMAController provides a dedicated external start bit for each internal buffer descriptor (maximum of 32) which allows DMA operations already configured in the core's internal buffer descriptors to be kicked-off from a simple fabric controller. Alternatively, DMA operations can be kicked-off by writing to the corresponding bit in the *Start Operation* register.

Start bits are queued and processed using a round-robin arbiter kicking off one operation per clock cycle. This loads DMA requests into the DMA controller with the DMA operation commencing once control is granted to that operation on the AXI4 master interface. This setup allows multiple operations to be kicked-off in a single control master write preventing conflicts arising from multiple internal and external start bits being asserted in the same cycle.

The arbiter decides the DMA operation that gets serviced on the AXI4 interface. Buffer descriptors are assigned a fixed priority upfront. Since, the largest permitted DMA operation is 8 MB, DMA operations are divided into multiple transactions with the maximum DMA transaction size determined by the priority of the buffer descriptor to which it is associated and the maximum number of beats permitted for this priority level, configured through parameter. As AXI4 transactions must run to completion once initiated, this mechanism allows transactions with higher priorities to have more bandwidth whilst forcing transactions with lower priorities to enter back into the arbitration sequence more frequently to check for higher priority DMA operations in the queue. Round-robin arbitration is performed to service requests with the same priority level. No bandwidth is given to descriptors with a lower priority level if a higher priority descriptor is being processed. The maximum transactions size for the highest level of priority is 4 KB to prevent AXI transactions from crossing 4 KB address boundaries.

When enabled, AXI4-Stream operations shares the highest priority level, priority level 0. It is possible to associate no internal buffer descriptors with priority level 0 at configuration time to allocate the entire bandwidth of the AXI4 DMA interface to AXI4-Stream operations when they exist. Otherwise, AXI4-Stream to memory map forwarding operations will be interleaved with operations of internal descriptors at the highest priority level.

3.1.5 Memory Map Cache

CoreAXI4DMAControl contains two 4 KB SRAM caches to allow the core to complete the forward part of an AXI4-memory map store and forward operation whilst performing the store element of the next store and forward operation. The DMA Controller block is responsible for switching between the internal caches autonomously in a round-robin fashion.

3.1.6 Stream Cache

If the AXI4-Stream configuration is selected, the core contains two additional 4 KB store caches allowing stream data received through the AXI4-Stream interface to be buffered before initiating multi-beat AXI4 forward transactions on the AXI4 interface. This allows stream operations to be received asynchronously to other DMA operations as the AXI4-Stream transfer is initiated by the AXI4-Stream master. The DMA Controller block is responsible for switching between the internal stream caches autonomously in a round-robin fashion.

3.1.7 Interrupt Controller

CoreAXI4DMAController allows users to enable multiple interrupt outputs and to associate each interrupt output with one or more internal buffer descriptors. The Interrupt Controller block is responsible for routing the events of each descriptor to the associated interrupt output. A configurable depth queue for each interrupt output is contained within this block to allow DMA operations to take place whilst waiting on the Control master to handle and clear previous interrupt events. If an interrupt queue backs up all operations of descriptors associated with this interrupt are suspended until space is freed in the interrupt queue by the Control master. DMA operations of other descriptors not associated with this interrupt queue will still be processed. This setup facilitates multiple processors to use the DMA controller concurrently.

Note: The interrupt association for an external descriptor is inherited from the previous internal descriptor in the chain. Interrupt events of Stream descriptors are always associated with Interrupt 0.

3.1.8 AXI4 Master Interface Controller

The AXI4 master interface controller is responsible for performing the DMA operation outlined by the DMA Controller block using AXI4 write and read transactions. This block returns a flag to the DMA controller when the requested store and forward operation is completed. If an AXI4 error is returned by a downstream slave during the read/store operation before the write/forward operation for that operation has commenced, the AXI4 Master Interface Controller continues the read/store to completion and reports the error to the control master without performing the write.

3.1.9 AXI4-Stream Slave Interface Controller

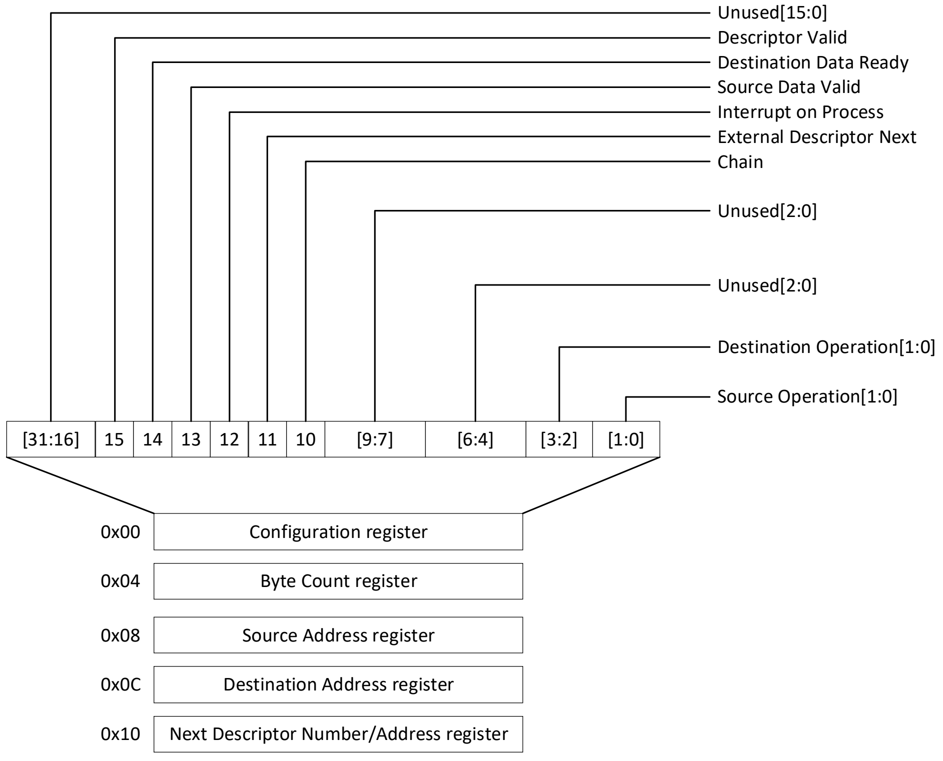
The AXI4-Stream slave interface controller provides a unidirectional bridge to the AXI4 memory map for AXI4 stream masters. The AXI4-Stream slave interface controller notifies the DMA Controller that a stream operation has commenced and of the position of the external descriptor in AXI4 memory describing the AXI4-Stream operation. Caching of the stream data in one of the internal 4 KB stream caches is performed in parallel to the fetching of the stream descriptor over the AXI4 DMA interface and completion of the previous memory mapped or stream DMA operation.

3.2 Buffer Descriptors

3.2.1 Internal Descriptor Support

Internal descriptors are implemented using LSRAM with the number of internal descriptors configurable through parameter in the range 4-32. Each internal descriptor has the ability to operate as a competing DMA channel[1]. The format of an internal descriptor is as shown in Figure 2:

Figure 2 • **Internal Descriptor Format**



Chaining of internal descriptors is supported to perform scatter-gather DMA operations where data is collected from one or more contiguous or non-contiguous locations and forwarded to one or more contiguous or non-contiguous locations. Each descriptor or chain of descriptors can be configured to perform repetitive cyclic operations. Refer to the Scatter-Gather and Cyclic Operations sections of this

1. *Each internal descriptor can operate as a competing DMA channel provided that it's not chained with other internal descriptors.*

document for information on performing such operations. Fetching of internal descriptors is performed in parallel to DMA data transferring and has no impact on DMA throughput performance provided that the operation described by the current descriptor being operated on is sufficiently large.

The priority level and interrupt output associated with operations of any internal descriptor must be configured at the time of instantiation through parameter.

3.2.2 External Descriptor Support

External descriptors can be defined in the AXI4 address space and fetched over the AXI4 DMA interface. The structure of an external descriptor takes the same format as an internal descriptor, as shown in Figure 2.

External descriptors are pointed to using an internal descriptor with the *Chain* and E*xternal Descriptor* bits set in the internal descriptor's *Configuration* register by passing the base address of the external descriptor to the *Next Descriptor Number/Address* register. Fetching of external descriptors over the DMA interface has a minor impact on overall DMA throughput performance provided that DMA operations are sizeable.

Chaining of external descriptors is supported through this mechanism. An external descriptor can be chained back to an internal descriptor by setting the *Chain* bitwith the *External Descriptor* bit cleared in the external descriptor's *Configuration* register and passing the internal descriptor number to the *Next Descriptor* Number/Address register. Cyclic operations of external descriptors can be achieved linking the last external descriptor in the chain back to the first internal descriptor.

The priority level and interrupt association of an external descriptor is inherited from the previous internal descriptor in the chain.

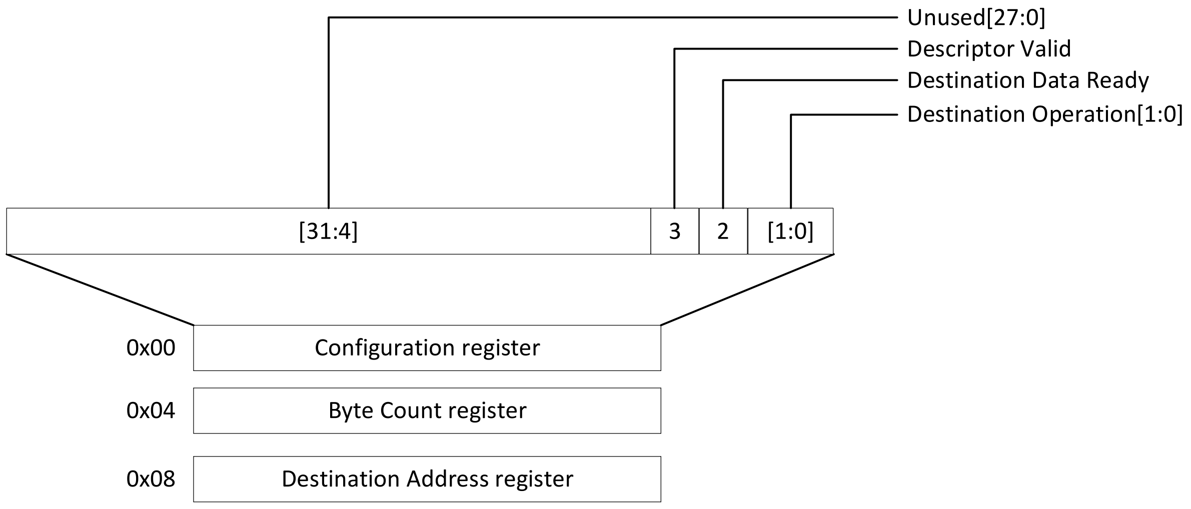
Note: The address of the *Configuration* register of an external descriptor must be aligned to the bus width of the AXI4 DMA interface configured through the *AXI_DMA_DWIDTH* parameter.

3.2.3 Stream Descriptor Support

Stream descriptors describe the AXI4-Stream transaction that is received from the AXI4-Stream master over the AXI4-Stream slave interface. The TDEST signal of the stream interface is used to select the CoreAXI4DMAController *Stream Descriptor Address* register that points to the stream descriptor describing the AXI4-Stream transaction that is in progress. The address of the stream descriptor must be written to the appropriate *Stream Descriptor Address* register prior to the AXI4-Stream transaction being initiated, along with the prior existence of the valid stream descriptor in the AXI4 DMA address space.

The format of stream descriptors is as shown in Figure 3:

Figure 3 • **Stream Descriptor Support**



The *Destination Data Ready bit* is used by control masters to denote when a buffer has been allocated for the reception of the stream data in the AXI4 memory-map address space. The first 4 KB of the stream transaction will be cached whilst the descriptor is being fetched. No further transfers in the AXI4-Stream transaction will be acknowledged until the *Destination Data Ready* bit is asserted, allowing the forwarding operation to the AXI4 memory map to commence when allocated bandwidth by the DMA arbiter.

If the *Descriptor Valid* bit of the stream descriptor that the AXI4-Stream transaction relates to is not set when the stream transaction is initiated and the descriptor is fetched, an invalid descriptor interrupt event will be triggered.

Stream descriptors are fetched over the AXI4 DMA interface when TVALID is asserted for the first transfer in the AXI4-Stream transaction once the current AXI4 or AXI4-Stream transaction completes. The maximum transfer size of a single stream transaction is 8 MB.

Note: The address of the *Configuration* register of a stream descriptor must be aligned to the bus width of the AXI4 DMA interface configured through the *AXI_DMA_DWIDTH* parameter. The bus width of the AXI4 DMA and stream interfaces are synonymous.

3.2.4 Descriptor Management

Since external start inputs exist for each internal descriptor and descriptors can be chained together, it is impossible for firmware to determine if a descriptor is currently involved in a DMA operation as it may have been initiated by a separate or external process. For this reason, it is advised to keep hardware-initiated DMA operations in separate descriptors to dynamically configured, firmware initiated operations. Once a descriptor is initialized by firmware during initial configuration, the *Descriptor Valid* bit must be set in the descriptor's *Configuration* register.

Note: If a UIC script is used to initialize internal descriptors in LSRAM, the *Descriptor Valid* bit will need to be set through firmware due to the order in which the UIC script writes the data into LSRAM (writes the *Configuration* register first, followed by the *Byte Count* register, hence the *Descriptor Valid* bit is cleared automatically by the DMA Controller).

If the start input associated with a descriptor is tied off then it is permitted for firmware to modify the descriptor contents on the fly, provided that no firmware initiated operations are being processed on the particular descriptor or which include the descriptor in a chain/cyclic buffer. Once any field in the descriptor is written to, the *Descriptor Valid* bit is automatically cleared by CoreAXI4DMAController. It is the responsibility of the firmware to set this bit only when the descriptor has been reconfigured.

If a DMA operation is initiated through an external start input assertion or a write to the *Start Operation* register on an uninitialized descriptor, an interrupt will be generated reporting the invalid descriptor error.

3.3 Flow Control/Throttling

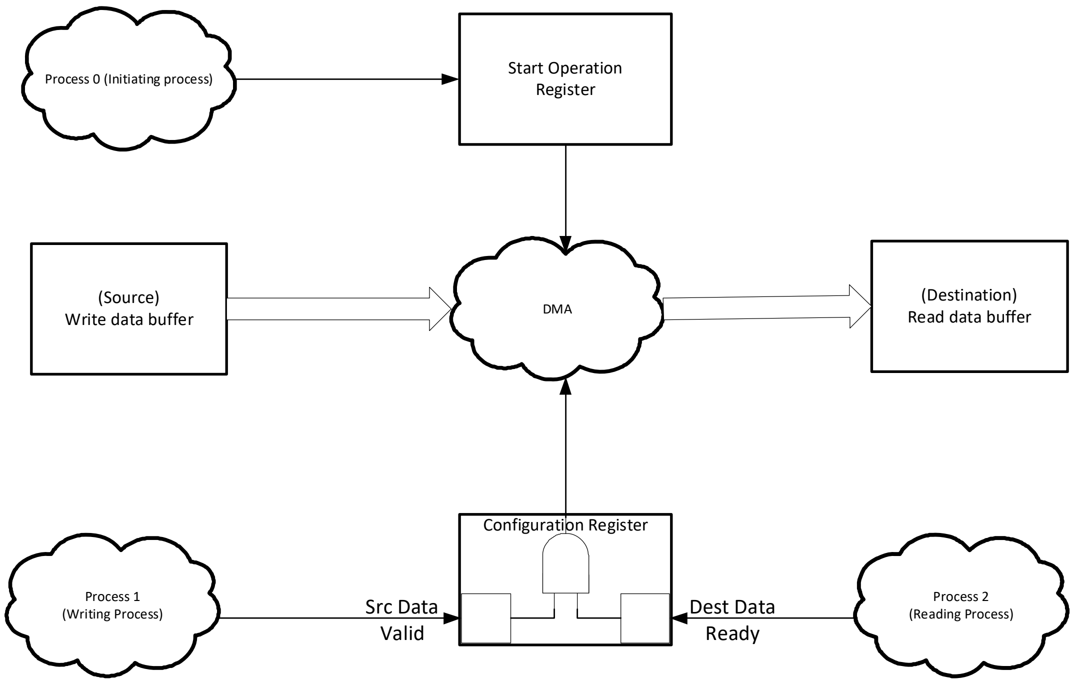
Once an AXI transaction is initiated by a master sending the write/read address it must run to completion. A slave can force a master to wait before sending additional write data or receiving read data using the valid-ready flow control signals built into the AXI4 protocol. However, masters cannot determine up-front that a slave is unable to handle more data and thereby may hog the bus.

To prevent CoreAXI4DMAController relying on AXI slaves always having data in place and room for write data, CoreAXI4DMAController implements separate *Source Data Ready and Destination Data Valid* bits in both internal and external descriptors to support firmware flow control management. In order for a descriptor to be operated on, firmware must set both flow control bits whether or not the descriptor is part of a descriptor chain.

Note: Both the *Source Data Valid* and *Destination Data Ready* flow control bits must be set again every time that a descriptor is processed as part of a cyclic chain.

To support the separation of the application into multiple processes, a descriptor can be kicked-off in the *Start Operation* register before either or both flow control bits are set, but no bandwidth will be granted to this descriptor until both bits are subsequently set either together or separately in the same process or in separate processes altogether. The conceptual flow control management of CoreAXI4DMAController is as shown in Figure 4:

Figure 4 • **Multi Process Flow Control Management**


Process 0 (Initiating process)
Start Operation Register
(Source) Write data buffer
DMA
(Destination) Read data buffer
Configuration Register
Process 1 (Writing Process)
Src Data Valid
Dest Data Ready
Process 2 (Reading Process)

Clearing of the *Source Data Valid* and *Destination Data Ready* flow control bits is performed automatically by CoreAXI4DMAController for both internal and external descriptors every time that an operation of the descriptor is completed.

If either the *Source Data Valid* or D*estination Data Ready* flow control bits are low when an external descriptor is fetched, the external descriptor's *Configuration* register will be fetched again over the AXI4 interface until both bits are detected high every time that this DMA channel is apportioned a time slot by the DMA arbiter. The allotted bandwidth will be passed-over if either bit remains low when fetched. No DMA transfer will be performed whilst arbitration will be re-entered immediately.

Stream descriptors support a *Destination Data Ready* bit for flow control purposes. As with internal and external descriptors, the *Destination Data Ready* bit is cleared by the DMA controller when the AXI4-Stream transaction data described in the stream descriptor has been forwarded to the AXI4 memory-map address space. The *Destination Data Ready* flow control bit will be fetched every time that the stream descriptor is allocated bandwidth by the DMA arbiter in the manner described for external descriptors above.

3.4 DMA Operations

3.4.1 Chain Operations (Scatter-gather)

Chain operations are typically used to implement scatter-gather DMA operations where data is collected from multiple non-concurrent memory blocks and spread to a number of concurrent or non-concurrent locations. A chain can be formed using a single internal descriptor and multiple external descriptors, multiple internal descriptors or using a mix of internal and external descriptors. The starting point for a descriptor chain must be an internal descriptor.

To configure a chain operation, create a number of buffer descriptors with the *Chain* bit set in each buffer descriptor except the last. The *Next Descriptor* field in all buffer descriptors other than the last should point to the next descriptor in the chain. This field will be ignored for the last descriptor in the chain (denoted by a descriptor without the *Chain* bit set). The *External Descriptor Next* bit is used to denote the type of the next descriptor in the chain. Chain operations can be initiated by writing to the internal start bit or by triggering the start input signal associated with the first descriptor in the chain.

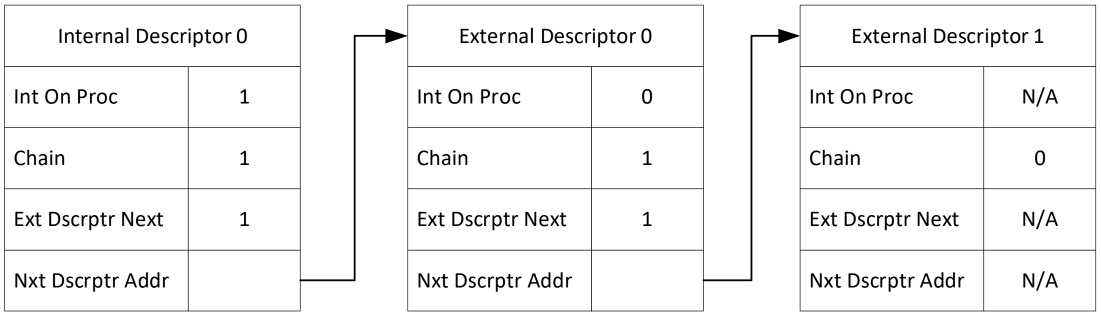
If a chain operation is configured with all descriptors having the *Interrupt on Process* bit cleared, an interrupt will only be generated on error or when the operation described by the last descriptor in the chain has completed.

Alternatively, if the *Interrupt on Process* bit is set for one or more descriptors in a chain, an interrupt will be generated when the operation described in each descriptor with its *Interrupt on Process* bit set is completed, however, this will not stall the next descriptor in the chain from being processed by the DMA controller, unless the associated interrupt queue has backed-up.

Note: Descriptor chaining should only be performed with buffer descriptors of the same priority level.

Figure 5 shows the format of one possible descriptor chain. Notice that the *Chain* bit is set for every descriptor other than the last. In this case, the *Interrupt on Process* bit is set for the first internal descriptor in the chain, Internal BD0 and External BD1. This will generate an interrupt when each of these descriptors are processed. No interrupt will be generated for the processing of External BD0. An interrupt will always be generated when the last descriptor in a chain has been processed, regardless of the Interrupt on Process configuration.

Figure 5 • **Example Descriptor Chain**

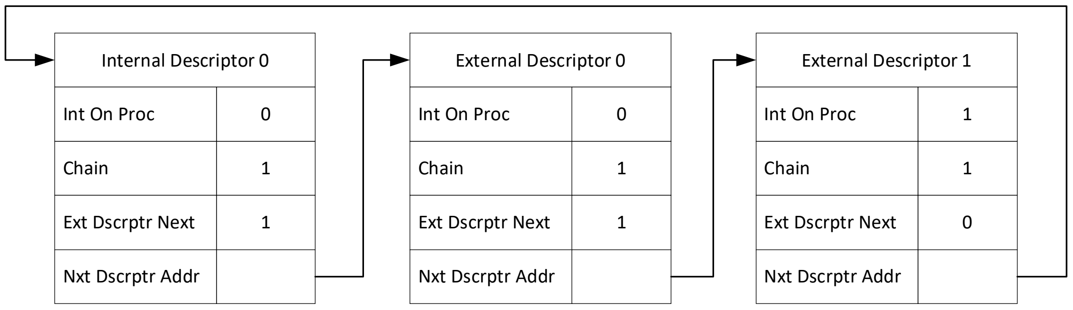


CoreAXI4DMAController supports a source address and destination no-op feature to allow an internal descriptor to point at an external descriptor without the internal descriptor performing a DMA data transfer. Refer to the *Source* and D*estination Operation* fields of the descriptor's *Configuration* register.

3.4.2 Cyclic Operations

Cyclic operations can be set up by creating a number of buffer descriptors with the *Chain* bit set in each buffer descriptor. The *Next Descriptor* field for each descriptor should be assigned the number of the next descriptor in the chain. Unlike standard chain operations, the last descriptor should have the *Chain* bit set and should point back to the first descriptor in order to make the operation cyclic. The cyclic operation is initiated by writing to the associated start bit for the first buffer descriptor in the cyclic chain. The flow control bits must be set again for each descriptor every time that the descriptor has been processed provided that there is valid data at the source and room for data at the destination. If a descriptor is reached without one or both flow control bits set, the DMA operation will be passed over in the DMA arbiter, allowing other DMA operations to proceed while waiting on buffer allocation.

Figure 6 • **Example Cyclic Descriptor Chain**



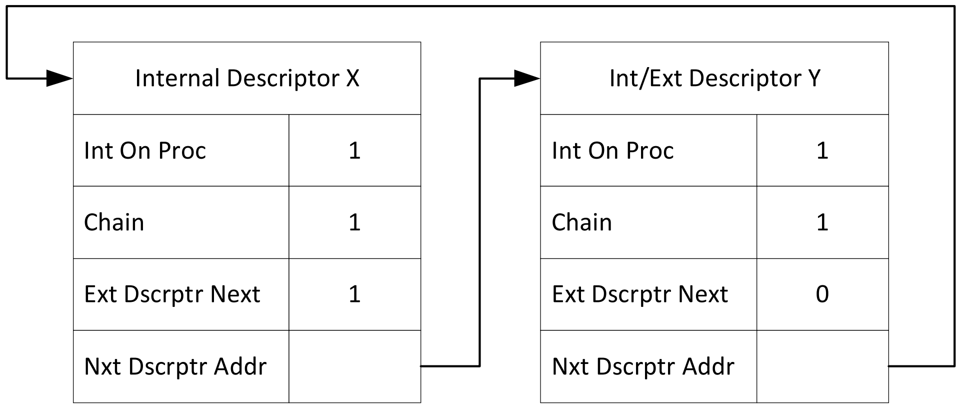
Cyclic operations can be terminated by clearing the *Chain* bit in the descriptor to stop at when next reached. When this descriptor is reached and the DMA operation within the descriptor has been carried out, an interrupt will be asserted signifying the completion of the DMA cyclic operation. If external buffer descriptors are included in a descriptor chain, the cycle can only be stopped on an internal descriptor from the Control interface of CoreAXI4DMAController. The chain can terminate on an external descriptor if the control master clears the *Chain* bit in the external descriptor's *Configuration* register in the AXI4 memory map address space.

It is recommend that the *Interrupt on Process* bit is asserted for at least the last or tail descriptor in a descriptor chain to provide notice to the firmware application that the chain has been processed and that the *Source Data Valid* and D*estination Data Ready* flow control bit will need to be set for each descriptor in the chain before another complete cycle of the chain can take place. The DMA arbiter allows the DMA transfer to continue on each descriptor in the chain provided that both flow control bits are set and that the descriptor is valid. The status of the next descriptor's flow control bits will not impact the current descriptors service.

3.4.2.1 Ping-Pong

Support for ping-pong cyclic DMA operations where data is read from one location and written to two or more memory buffers with an interrupt generated every time that a buffer is filled is provided through use of descriptor chaining and the *Interrupt on Process* bit within buffer descriptors. The *Interrupt on Process* bit causes an interrupt to be raised every time that a descriptor has been processed. This feature is supported for both internal and external buffer descriptors. The resetting of the data valid bit signifies that the buffer can be re-written to by the DMA controller.

Figure 7 • **Ping-Pong Cyclic Descriptor Chain**

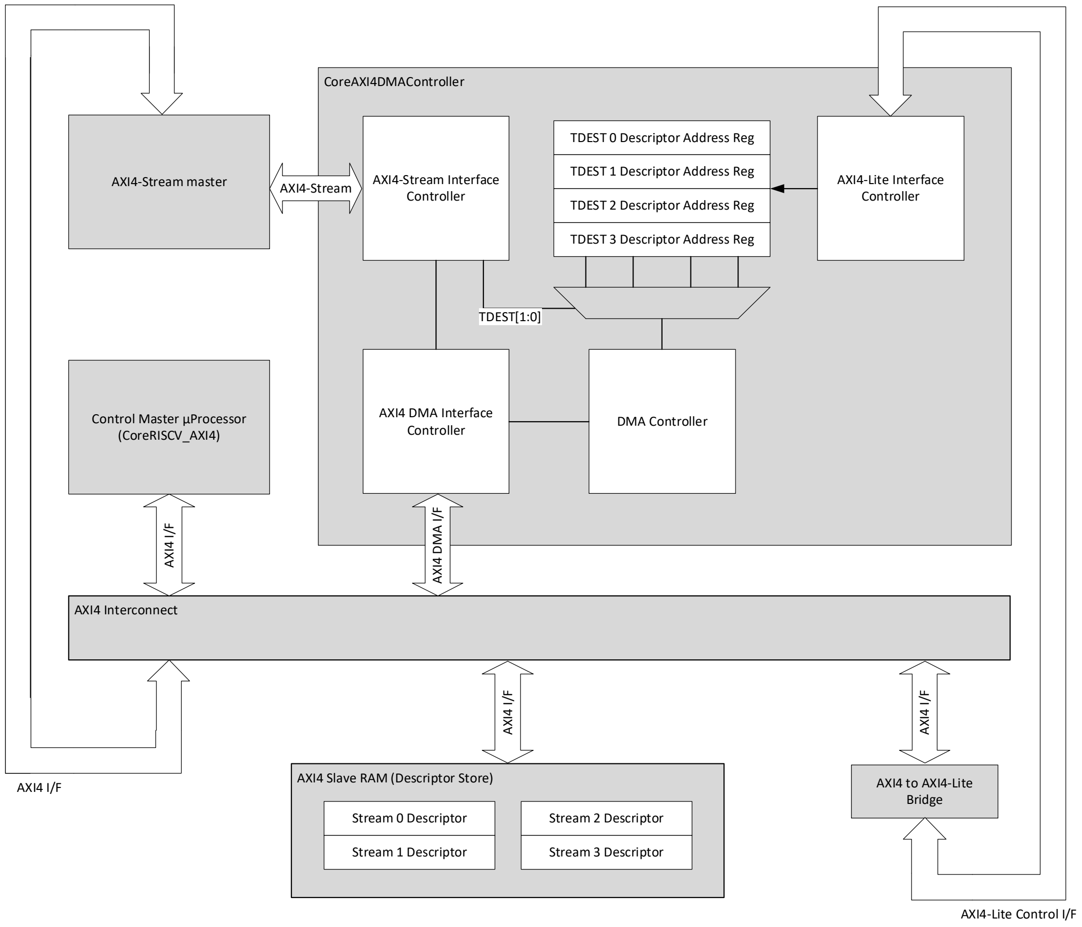


3.5 AXI4-Stream Bridge

In addition to AXI4 memory-map to memory-map DMA operations, CoreAXI4DMAController provides a bridge to the AXI4 memory-map address space for AXI4-Stream masters. Before a stream master initiates a stream transaction, the Control master must first write a description of the imminent AXI4-Stream transaction to the AXI4 memory map in the stream descriptor format.

Next the address of the *Configuration* register of this stream descriptor needs to be written to the *Stream Descriptor Address* register within CoreAXI4DMAController associated with the TDEST signal value of the AXI4-Stream transaction. The TDEST signal provides destination routing information, used to multiplex between the 4 *Stream Descriptor Address* registers inside CoreAXI4DMAController, as shown in Figure 8.

Figure 8 • **AXI4-Stream to AXI4 Memory Map Bridging**


CoreAXI4DMAController
AXI4-Stream master
AXI4-Stream
AXI4-Stream Interface Controller
TDEST 0 Descriptor Address Reg
TDEST 1 Descriptor Address Reg
TDEST 2 Descriptor Address Reg
TDEST 3 Descriptor Address Reg
AXI4-Lite Interface Controller
TDEST[1:0]
Control Master µProcessor (CoreRISCV_AXI4)
AXI4 DMA Interface Controller
DMA Controller
AXI4 I/F
AXI4 DMA I/F
AXI4 Interconnect
AXI4 I/F
AXI4 I/F
AXI4 I/F
AXI4 Slave RAM (Descriptor Store)
Stream 0 Descriptor
Stream 1 Descriptor
Stream 2 Descriptor
Stream 3 Descriptor
AXI4 to AXI4-Lite Bridge
AXI4-Lite Control I/F

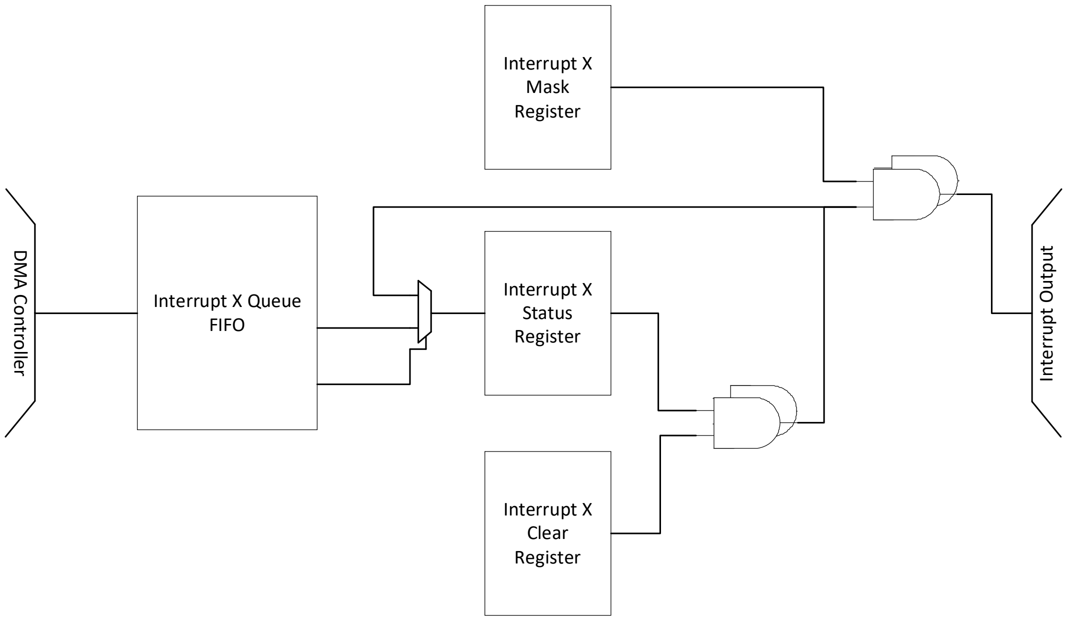
Once the first transfer of the AXI4-Stream transaction is received, the TDEST signal is used to select the *Stream Descriptor Address* register that contains the address of the stream descriptor describing the AXI4-Stream transaction. The stream descriptor is then fetched over the AXI4 DMA interface once the current AXI4 DMA transaction completes, whilst the stream data is cached in parallel. If the stream descriptor is valid then it is loaded into the DMA Arbiter where it competes for bandwidth with other priority 0 requests. Otherwise, an invalid descriptor interrupt event is generated. Once bandwidth is allocated to the stream request, if the *Destination Data Ready* flow control bit was set when the descriptor was fetched, the AXI4 memory map forward operation commences on the AXI4 DMA interface. If the flow control bit is not set then the flow control bit is re-fetched from the stream descriptor over the AXI4 DMA interface until it is detected as begin set, at which point the AXI4 memory map forward commences.

Once the AXI4 memory-map forward related to the AXI4-Stream read is completed, an interrupt is generated on the Interrupt 0 output.

3.6 Interrupts

CoreAXI4DMAController provides a configurable number of interrupt outputs, each with independent status, mask and clear registers along with independent configurable depth queues. Descriptors must be associated with interrupt outputs at instantiation time. Multiple descriptors may be associated with a single interrupt output. Figure 9 shows the logic implemented for each enabled interrupt output.

Figure 9 • **Interrupt Queue Logic**


Interrupt X Mask Register
DMA Controller
Interrupt X Queue FIFO
Interrupt X Status Register
Interrupt X Clear Register
Interrupt Output

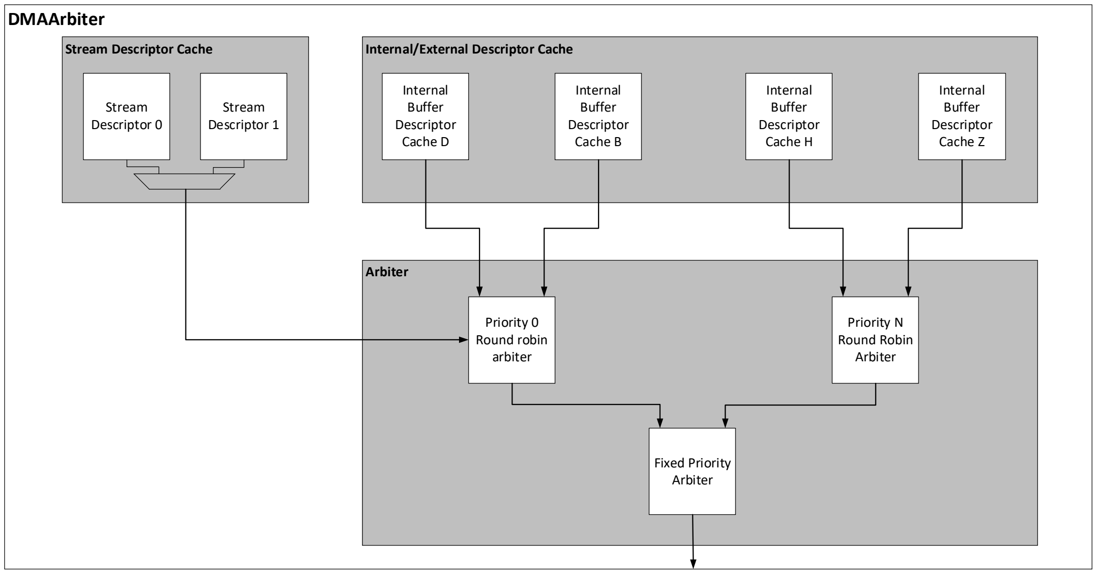
Interrupts are generated on error or completion. When configuring a descriptor, the *Interrupt on Process* field provides the user with a means to generate an interrupt every time that a particular descriptor is processed in a chain or circular buffer configuration, provided that this descriptor is not the last descriptor in the chain. DMA operations are not halted for interrupts of this type and will proceed in the background provided that the associated interrupt queue has not backed-up.

The current interrupt status will remain valid in the *Interrupt X Status* register associated with the interrupt output until all unmasked bits (masked-off using the *Interrupt Mask* register) in the *Interrupt Status* register have been cleared through the *Interrupt Clear* register.

3.7 Arbitration

CoreAXI4DMAController supports fixed priority arbitration between multiple priority levels. If multiple requests exist of the same priority level, round-robin arbitration is performed between these requests. Higher priority requests are allowed to perform larger AXI4 burst transfers. This forces faster re-entrance into the arbitration logic by lower priority requests to ensure that higher priority requests are detected sooner.

Figure 10 • Descriptor Arbitration


DMAArbiter
Stream Descriptor Cache
Stream Descriptor 0
Stream Descriptor 1
Internal/External Descriptor Cache
Internal Buffer Descriptor Cache D
Internal Buffer Descriptor Cache B
Internal Buffer Descriptor Cache H
Internal Buffer Descriptor Cache Z
Arbiter
Priority 0 Round robin arbiter
Priority N Round Robin Arbiter
Fixed Priority Arbiter

Note: No bandwidth is allotted to lower priority descriptor requests when a request from a higher priority descriptor with both its flow control bits set exists.

Since, a total transfer size of 8 MB is permitted for an operation of a single descriptor, DMA operations are divided into multiple AXI4 transactions, with a maximum permitted transaction size of 4 KB to provide AXI4 boundary protection. Access is re-evaluated every time that an AXI4 transaction completes. If multiple large descriptor requests of the same priority level exist, AXI4 transactions of each descriptor will be interleaved in a round-robin fashion.

Note: CoreAXI4DMAController does not support interleaving of transfers within AXI4 transactions (xID signal unused). Only 1 descriptor is serviced per AXI4 transaction.

When enabled, AXI4-Stream requests are interleaved with operations of the highest priority level, priority 0. If the performance of the AXI4-Stream master is paramount, no internal descriptors should be assigned priority 0 priority association at instantiation time to allocate the entire bandwidth of the AXI4 memory map interface to the AXI4-Stream operation. Unlike standard AXI4 DMA operations which enter back into the arbitration algorithm every time that an AXI4 transaction is performed provided that the destination transaction does not span a 4 KB boundary, which is not aligned to the source address (in which case the core will perform multiple AXI writes to empty the read cache which was filled by a single AXI read transaction before entering back into arbitration), whether or not the AXI4 transaction completely services the descriptor request, AXI4-Stream operations do not enter back into arbitration until the entire stream request has been serviced, even if this requires multiple AXI4 write transactions to prevent crossing a 4 KB boundary in a sequential transaction and to empty the stream read cache.

Priority level assignment should be performed in a descending, contiguous fashion. If a priority level is enabled with no descriptors allocated to it (apart from priority 0 when Stream support is enabled), the underlying logic related to this priority level will be removed from the core. For this reason, no priority level should be enabled without descriptors allocated to it when lower priority levels are enabled with descriptors allocated to them.

To reduce the setup overhead on DMA transfers, the core allows multiple descriptors to be kicked-off in a single write to the *Start Operation* register. It is difficult to determine the order that DMA operations will take place in as the core has the facility for multiple start bits to get kicked-off in the same cycle, each with shared or separate priority levels. All start bits are maintained in an internal register and loaded into the descriptor cache using a non-weighted round-robin arbiter, with only one start bit loaded in any given cycle. The weighted round-robin DMA arbiter then reads out of the descriptor cache and fills a two-stage pipeline to ensure that DMA transfers happen in a back-to-back fashion on the DMA interface and to ensure that read-ahead can be performed for the next DMA operation as there are two internal caches. To ensure that DMA operations happen in order, either wait for an interrupt to be received on the first operation before kicking off the next operation, else form a chain with the descriptors of interest.

3.8 AXI Transactions

The source and destination addresses of all DMA operations must be aligned to AXI bus width instantiated.

Note: Unaligned AXI transactions and data realignment are not performed by CoreAXI4DMAController.

The size of the AXI transaction generated is dependent upon the width of the AXI DMA interface instantiated. CoreAXI4DMAController generates transactions with the AxSIZE field set at the maximum for the instantiated bus width exclusively.

The length of the AXI transaction is dependent upon the priority level of the descriptor which the AXI transaction is related to, the maximum permitted burst size at this priority level and the source address, destination address and number of bytes in the DMA operation. If the store DMA operation spans a 4 KB boundary, CoreAXI4DMAController may generate multiple shorter read transactions to prevent crossing the 4 KB boundary in a sequential transfer and vice versa for the forward transactions if the forward DMA operation spans a 4 KB boundary. Arbitration is performed on completion of each read transaction to determine if a higher priority requests exists or if there are other requests at this priority level that need to be serviced. The last transfer of the last transaction of a DMA operation may be narrow by way of the AXI write strobes to provide N-byte DMA transfer support.

A single AXI read transaction may result in multiple AXI write transactions if the source operation is confined within a 4 KB slot and the destination operations spans a 4 KB boundary.

The core supports DMA read ahead to read in the next data to the internal cache whilst the previous write operation is completing. This increases the throughput of the AXI interface significantly for queued AXI DMA operations in busy AXI systems. AXI outstanding address transactions are not generated by CoreAXI4DMAController.

3.9 AXI4-Stream Transactions

CoreAXI4DMAController supports AXI4-Stream transactions where each bit of TSTRB mirrors TKEEP, indicating either data or null bytes. Position bytes occur when a TKEEP bit is asserted and the corresponding TSTRB bit is de-asserted. Position bytes are not supported by CoreAXI4DMAController.

The core expects all AXI4-Stream transactions to start from an address aligned to bus width, set through the *AXI_DMA_DWIDTH* parameter. The first transfer in the transaction cannot be unaligned through the use of TKEEP and TSTRB.

In addition, the core permits the last transfer in an AXI4-Stream transaction to be narrow by keeping the TKEEP and TSTRB bits low for the associated byte lanes, making N-byte AXI4-Stream to AXI4 memory map bridging possible. An example of this would be where TKEEP and TSTRB are both 0x7F for the last transfer in the transaction and 0xFF for all other transfers in the transaction on a 64-bit bus.

It is expected that all TSTRB and TKEEP bits are asserted for every transfer other than the last transfer in an AXI4-Stream transaction. Sparse strobe assertion is not supported where TSTRB and TKEEP are both 0x55 for instance.

Note: AXI4 Stream Transactions should be performed whenever there is no pending request for memory map to memory map data transfer and vice-versa.

3.10 Cache Coherence

Data integrity issues can arise where a DMA read is performed on a shared, cacheable memory where the cached memory contents has been updated but not yet written back to memory. To cover such an event, the application must ensure that the contents of the cache is written back to memory before kicking of a DMA read operation in a cacheable region. Processor operations to this memory space should be suspended until the DMA read has completed.

Similarly, where a DMA write is performed on cacheable memory, the application needs to ensure that the contents of the cache is flushed and refilled to prevent the processor from operating on out-of-date data once the memory has been written via the DMA operation. Processor operations to this memory space should be suspended until the DMA write has completed.

Note: Cache coherence is not handled by CoreAXI4DMAController and remains the responsibility of the Firmware application.

4 Interface Descriptions

4.1 Signal Descriptions

Signal descriptions for CoreAXI4DMAController are defined in Table 11.

Table 11 • **CoreAXI4DMA Controller I/O Signals**

Port Name	Width	Direction	Description
General Ports			
CLOCK	1	Input	Clock signal to all sequential elements within the core.
RESETN	1	Input	Active-low reset signal to all sequential elements within the core. Reset de-assertion must be synchronous to CLOCK rising edge as per AXI4 specification. This reset should be synchronized in the CLOCK clock domain externally.
AXI4-Lite Control Interface Ports			
CTRL_AWVALID	1	Input	Write address valid. Indicates that the control master is presenting valid write address information.
CTRL_AWREADY	1	Output	Write address ready. Indicates that CoreAXI4DMAController is ready to receive write address information.
CTRL_AWADDR[10:0]	11	Input	AXI4-Lite write address bus.
CTRL_WVALID	1	Input	Write data valid. Indicates that the control master is presenting write data.
CTRL_WLAST	1	Input	Indicates that the current transfer is the last transfer in the write transaction.
CTRL_WREADY	1	Output	Indicates that CoreAXI4DMAController is ready to receive write data.
CTRL_WSTRB[3:0]	4	Input	Write strobes. Indicates the byte lanes of the WDATA bus, which contain valid write data.
CTRL_WDATA[31:0]	32	Input	AXI4-Lite write data bus.
CTRL_BVALID	1	Output	Write response valid. Indicates that CoreAXI4DMAController is presenting valid write response information. Only occurs at the end of a write transaction.
CTRL_BREADY	1	Input	Indicates that the control master is ready to receive write response information.

Table 11 • **CoreAXI4DMA Controller I/O Signals *(continued)***

CTRL_BRESP[1:0]	2	Output	Write response. Indicates the status of a write transaction. **Okay** and **SLVERR** responses are returned by the core. The core returns SLVERR when an invalid address is latched on AXI4-Lite write address bus.
CTRL_ARVALID	1	Input	Read address valid. Indicates that the control master is presenting valid address information.
CTRL_ARREADY	1	Output	Read address ready. Indicates that CoreAXI4DMAController is ready to receive address information.
CTRL_ARADDR[10:0]	11	Input	AXI4-Lite read address bus.
CTRL_RVALID	1	Output	Read data valid. Indicates that CoreAXI4DMAController is presenting valid read data.
CTRL_RREADY	1	Input	Indicates that the control master is ready to receive read data.
CTRL_RDATA[31:0]	32	Output	AXI4-Lite read data bus.
CTRL_RLAST	1	Output	Indicates that the current transfer is the last transfer in the read transaction.
CTRL_RRESP[1:0]	2	Output	Read response valid. Indicates that CoreAXI4DMAController is presenting valid read response information. Valid for every transfer in a transaction. **Okay** and **SLVERR** responses are returned by the core. The core returns SLVERR when an invalid address is latched on the AXI4-Lite read address bus.
	AXI4 DMA Interface Ports		
DMA_AWVALID	1	Output	Write address valid. Indicates that CoreAXI4DMAController is presenting valid address information.
DMA_AWREADY	1	Input	Write address ready. Indicates that the AXI4 slave is ready to receive address information.
DMA_AWADDR[31:0]	32	Output	DMA write address bus.
DMA_AWID[ID_WIDTH-1:0]	ID_WIDTH	Output	Write address ID. Identification tag for write address group of signals. Driven out as zeros as ID support not implemented by CoreAXI4DMAController. All transactions are processed in order.

Table 11 • **CoreAXI4DMA Controller I/O Signals *(continued)***

DMA_AWLEN[7:0]	8	Output	Indicates the number of transfers in the AXI write transaction. Allows for between 1 and 256 transfers per transaction.
DMA_AWSIZE[2:0]	3	Output	Indicates the size of transfers in the AXI write transaction. CoreAXI4DMAController generates transactions of the data bus width size exclusively. DMA_AWSIZE = $\log_2(\frac{AXI\_DMA\_DWIDTH}{8})$
DMA_AWBURST[1:0]	2	Output	Write burst address type. Only fixed and incrementing DMA transfers are generated by CoreAXI4DMAController.
DMA_WVALID	1	Output	Write data valid. Indicates that CoreAXI4DMAController is presenting valid write data.
DMA_WLAST	1	Output	Indicates that the current transfer is the last transfer in the write transaction.
DMA_WREADY	1	Input	Indicates that the AXI4 slave is ready to receive write data.
DMA_WSTRB[(AXI_DMA_DWIDTH/8)-1:0]	AXI_DMA_DWIDTH/8	Output	Write strobes. Indicates the byte lanes of the WDATA bus which contain valid write data.
DMA_WDATA[AXI_DMA_DWIDTH-1:0]	AXI_DMA_DWIDTH	Output	DMA write data bus.
DMA_BVALID	1	Input	Write response valid. Indicates that the AXI4 slave is presenting valid write response information. Only occurs at the end of a transaction.
DMA_BREADY	1	Output	Indicates that CoreAXI4DMAController is ready to receive write response information.
DMA_BID[ID_WIDTH:0]	ID_WIDTH	Input	Write response ID. Identification tag for write response group of signals.
DMA_BRESP[1:0]	2	Input	Write response. Indicates the status of a write transaction.
DMA_ARVALID	1	Output	Read address valid. Indicates that CoreAXI4DMAControlleris presenting valid address information.
DMA_ARREADY	1	Input	Read address ready. Indicates that the control master is ready to receive address information.
DMA_ARADDR[31:0]	32	Output	DMA read address bus.

Table 11 • **CoreAXI4DMA Controller I/O Signals *(continued)***

DMA_ARID[ID_WIDTH-1:0]	ID_WIDTH	Output	Read ID. Identification tag for read address group of signals. Driven out as zeros as ID support not implemented byCoreAXI4DMAController. All transactions are processed in order.
DMA_ARLEN[7:0]	8	Output	Indicates the number of transfers in the AXI read transaction. Allows for between 1 and 256 transfers per transaction.
DMA_ARSIZE[2:0]	3	Output	Indicates the size of transfers in the AXI read transaction. CoreAXI4DMAController generates transactions of the data bus width size exclusively. DMA_ARSIZE = $\log_2(\frac{AXI\_DMA\_DWIDTH}{8})$
DMA_ARBURST[1:0]	2	Output	Read burst address type. Only fixed and incrementing DMA transfers are generated by CoreAXI4DMAController.
DMA_RVALID	1	Input	Read data valid. Indicates that the AXI4 slave is presenting read data.
DMA_RREADY	1	Output	Indicates that the CoreAXI4DMAController is ready to receive read data.
DMA_RDATA[AXI_DMA_DWIDTH-1:0]	AXI_DMA_DWIDTH	Input	DMA read data bus.
DMA_RLAST	1	Input	Indicates that the current transfer is the last transfer in the read transaction.
DMA_RRESP[1:0]	2	Input	Read response data. Read response data returned by the AXI4 slave. Valid for every transfer in a transaction.
DMA_RID[ID_WIDTH-1:0]	ID_WIDTH	Input	Read ID. Identification tag for read group of signals.
	AXI4-Stream Interface		
TVALID	1	Input	AXI4-Stream master is driving a valid transfer.
TREADY	1	Output	CoreAXI4DMAController is ready to receive stream data.
TDATA[AXI_DMA_DWIDTH-1:0]	AXI_DMA_DWIDTH	Input	Stream data bus.
TSTRB[(AXI_DMA_DWIDTH/8)-1:0]	AXI_DMA_DWIDTH/8	Input	Indicates whether the associated byte lane within TDATA should be treated as a data byte or a position byte. TSTRB[0] is associated with TDATA[7:0], and so on.

Table 11 • **CoreAXI4DMA Controller I/O Signals *(continued)***

TKEEP[(AXI_DMA_DWIDTH)-1:0]	AXI_DMA_DWIDTH/8	Input	Indicates the byte lanes of the TDATA bus which are processed as part of the stream. Byte lanes that have the associated TKEEP bit de-asserted contain null bytes. TKEEP[0] is associated with TDATA[7:0], and so on.
TLAST	1	Input	Indicates the boundary of a packet.
TID[ID_WIDTH-1:0]	ID_WIDTH	Input	Data stream identifier.
TDEST[1:0]	2	Input	Provides routing information for the data stream.
	Interrupt Interface Ports		
Interrupt0	1	Output	Interrupt 0 output. Remains asserted until all unmasked bits in the Interrupt 0 Status register are cleared via the Interrupt 0 Clear register. Interrupt events of buffer descriptors associated with this Interrupt output are queued.
…			
Interrupt3	1	Output	Interrupt 3 output, enabled at instantiation time when NUM_OF_INTS = 3. Remains asserted until all unmasked bits in the Interrupt 3 Status register are cleared via the Interrupt 3 Clear register. Interrupt events of buffer descriptors associated with this Interrupt output are queued.
	Start Interface Ports		
STRTDMAOP [NUM_INT_BDS-1:0]	NUM_INT_BDS	Input	Input to allow the DMA operation described in an internal buffer descriptor, to be initiated by a fabric controller. The DMA operation is kicked-off when this bit is asserted for one clock cycle. This signal should be synchronized in the CLOCK clock domain externally.

4.2 Configuration Parameters

There are a number of configurable options, which are applied to CoreAXI4DMAController (as shown in Table 12). If a configuration other than the default is required, then the configuration dialog box in SmartDesign should be used to select appropriate values for the configurable options.

Table 12 • **CoreAXI4DMAController Configuration Options**

Name	Valid Range	Default	Description
ECC	0-1	0	**Note:** This feature is only valid for RTG4, PolarFire, and PolarFire SoC device families. For more information on ECC Flags (SB_CORRECT and DB_DETECT), refer to Table 19, page 30. Error Correcting Code: • 0: ECC is disabled for uSRAM/LSRAM RAM • 1: ECC is enabled for LSRAM RAM
AXI4_STREAM_IF	0-1	0	AXI Stream slave interface. Provides bridge for AXI4-Stream masters to access the AXI4 memory map.
AXI_DMA_DWIDTH	32, 64,128, 256, 512	32	Data width of the AXI DMA and optional AXI4-Stream interfaces.
ID_DWIDTH	1-8	1	ID width for AXI DMA transactions. ID will always be driven out as zeros (unused – no transaction interleaving support).
NUM_PRI_LVLS	1-8	1	Number of fixed priority levels supported.
PRI_0_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	256	Maximum number of transfers in an AXI transaction of this priority level.
PRI_1_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	128	Maximum number of transfers in an AXI transaction of this priority level. Must be ≤ PRI_0_NUM_OF_BEATS.
PRI_2_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	64	Maximum number of transfers in an AXI transaction of this priority level. Must be ≤ PRI_1_NUM_OF_BEATS.
PRI_3_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	32	Maximum number of transfers in an AXI transaction of this priority level. Must be ≤ PRI_2_NUM_OF_BEATS.
PRI_4_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	16	Maximum number of transfers in an AXI transaction of this priority level. Must be ≤ PRI_3_NUM_OF_BEATS.
PRI_5_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	8	Maximum number of transfers in an AXI transaction of this priority level. Must be ≤ PRI_4_NUM_OF_BEATS.
PRI_6_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	4	Maximum number of transfers in an AXI transaction of this priority level. Must be ≤ PRI_5_NUM_OF_BEATS.
PRI_7_NUM_OF_BEATS	1, 4, 8, 16, 32, 64, 128, 256	1	Maximum number of transfers in an AXI transaction of this priority level. Must be ≤ PRI_6_NUM_OF_BEATS.

Table 12 • **CoreAXI4DMAController Configuration Options *(continued)***

NUM_OF_INTS	1-4	1	Number of interrupt outputs enabled.
INT_0_QUEUE_DEPTH	1-8	1	Number of interrupt events that can be queued for the interrupt 0 output.
INT_1_QUEUE_DEPTH	1-8	1	Number of interrupt events that can be queued for the interrupt 1 output.
INT_2_QUEUE_DEPTH	1-8	1	Number of interrupt events that can be queued for the interrupt 2 output.
INT_3_QUEUE_DEPTH	1-8	1	Number of interrupt events that can be queued for the interrupt 3 output.
NUM_INT_BDS	4, 8, 16, 32	4	Number of internal buffer descriptors supported.
DSCRPTR_0_PRI_LVL	0-7	0	Buffer Descriptor 0 priority level. Fixed priority level associated with operations of this descriptor.
DSCRPTR_0_INT_ASSOC	0-3	0	Buffer Descriptor 0 interrupt association. Associate interrupt events of this buffer descriptor with an interrupt output.
…			
DSCRPTR_31_PRI_LVL	0-7	0	Buffer Descriptor 31 priority level. Fixed priority level associated with operations of this descriptor.
DSCRPTR_31_INT_ASSOC	0-3	0	Buffer Descriptor 31 interrupt association. Associate interrupt events of this buffer descriptor with an interrupt output.

Note: When ECC is enabled, the core generates all the memories into LSRAM.

5 Register Map and Descriptions

The CoreAXI4DMAController registers are listed in Table 13.

Table 13 • **CoreAXI4DMAController Registers**

Offset	Register Name	Type	Width	Res	Description
0x00	Version Register	Read-only	4	N/A	Register holding the major, minor, and build number of the core.
0x04	Start Operation Register	Write-only	4	N/A	Start bits to kick-off DMA operations.
0x10	Interrupt 0 Status Register	Read-only	4	0x0	Status register containing status information for Interrupt 0 events.
0x14	Interrupt 0 Mask Register	Read-write	4	0x0	Configure flags in the Interrupt 0 Status register that can generate an interrupt.
0x18	Interrupt 0 Clear Register	Write-only	4	N/A	Clear flags asserted in the Interrupt 0 Status register.
0x1C	Interrupt 0 External Address Register	Read-only	4	0x0	Address of the external buffer descriptor with which the contents of the Interrupt Status register is associated.
0x20	Interrupt 1 Status Register	Read-only	4	0x0	Status register containing status information for Interrupt 1 events.
0x24	Interrupt 1 Mask Register	Read-write	4	0x0	Configure flags in the Interrupt 1 Status register that can generate an interrupt.
0x28	Interrupt 1 Clear Register	Write-only	4	N/A	Clear flags asserted in the Interrupt 1 Status register.
0x2C	Interrupt 1 External Address Register	Read-only	4	0x0	Address of the external buffer descriptor with which the contents of the Interrupt Status register is associated.
0x30	Interrupt 2 Status Register	Read-only	4	0x0	Status register containing status information for Interrupt 2 events.
0x34	Interrupt 2 Mask Register	Read-write	4	0x0	Configure flags in the Interrupt 2 Status register that can generate.
0x38	Interrupt 2 Clear Register	Write-only	4	N/A	Clear flags asserted in the Interrupt 2 Status register.
0x3C	Interrupt 2 External Address Register	Read-only	4	0x0	Address of the external buffer descriptor with which the contents of the Interrupt Status register is associated.

Table 13 • **CoreAXI4DMAController Registers *(continued)***

0x40	Interrupt3Status Register	Read-write	4	0x0	Status register containing status information for Interrupt 3 events.
0x44	Interrupt 3 Mask Register	Read-write	4	0x0	Configure flags in the Interrupt 3 Status register that can generate an interrupt.
0x48	Interrupt 3 Clear Register	Write-only	4	N/A	Clear flags asserted in the Interrupt 3 Status register.
0x4C	Interrupt 3 External Address Register	Read-write	4	0x0	Address of the external buffer descriptor with which the contents of the Interrupt Status register is associated.
0x60	Internal Descriptor 0 Configuration Register	Read-write	4	N/A	Register for configuring the DMA operation performed from by descriptor.
0x64	Internal Descriptor 0 Byte Count Register	Read-write	4	N/A	Number of bytes to be transferred in the DMA operation.
0x68	Internal Descriptor 0 Source Address Register	Read-write	4	N/A	Start address to source data from for the DMA store operation.
0x6C	Internal Descriptor 0 Destination Address Register	Read-write	4	N/A	Start address to forward data to for the DMA forward operation.
0x70	Internal Descriptor 0 Next Descriptor Address Register	Read-write	4	N/A	Descriptor number/address of the next internal/external descriptor in the chain.
0x80	Internal Descriptor 1 Configuration Register	Read-write	32	N/A	Buffer Descriptor 1.
…					
0x450	Internal Descriptor 31 Next Descriptor Address Register	Read-write	32	N/A	Buffer Descriptor 31.
0x460	Stream 0 Address Register	Read-write	32	0x0	External stream descriptor associated with Stream transactions with TDEST = 0b00.
0x464	Stream 1 Address Register	Read-write	32	0x0	External stream descriptor associated with Stream transactions with TDEST = 0b01.
0x468	Stream 2 Address Register	Read-write	32	0x0	External stream descriptor associated with Stream transactions with TDEST = 0b10.
0x46C	Stream 3 Address Register	Read-write	32	0x0	External stream descriptor associated with Stream transactions with TDEST = 0b11.

Note: CoreAXI4DMAController generates error response (SLVERR) on AXI4-Lite write response channel and on read response, when register address other than mentioned in Table 13 (invalid address) is accessed.

5.1 Version Register

The *Version* register provides control masters with the CPZ version number of the CoreAXI4DMAController instance. Table 14 describes the *Version* register.

Table 14 • **Version Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
0x00	Version Register	Read-only	32	N/A	Register holding the major, minor, and build number of the core.

Table 15 • **Version Register Bit Definitions**

Bit(s)	Name	Type	Description
7:0	Build Number	Read-only	Build number of the CPZ.
15:8	Minor Number	Read-only	Minor version number of the CPZ.
23:16	Major Number	Read-only	Major version number of the CPZ.
31:24	Reserved	Read-only	Reserved – Returns all zeros if read.

5.2 Start Operation Register

The *Start Operation* register allows control masters to kick-off DMA operations described in internal buffer descriptors. Table 16 describes the *Start Operation* register. Multiple DMA operations can be initiated with a single write operation by setting multiple bits simultaneously in the *Start Operation* register.

Note: DMA operations may not get initiated on the DMA interface in the order in which they were loaded into the *Start Operation* register. The order is dependent upon the previous descriptor kicked-off and the number and size of descriptor operations held in DMA controller's internal two-stage pipeline.

Table 16 • **Start Operation Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
0x04	Start Operation	Write-only	32	N/A	Start bits to kick-off DMA operations.

Table 17 • **Start Operation Register Bit Definitions**

Bit(s)	Name	Type	Description
0	Start Bit 0	Write-only	Kicks off the DMA operation described in Internal Descriptor 0.
		…	
31	Start Bit 31	Write-only	Kicks off the DMA operation described in Internal Descriptor 31.

5.3 Interrupt X Status Register

The *Interrupt X Status* register provides status information associated with Interrupt X events.

Table 18 • **Interrupt X Status Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x0	Interrupt X Status Register	Read-only	32	0x00	Status register containing status information for Interrupt X events. Returns the status information for the event at the head of the Interrupt X Queue.

Table 19 • **Interrupt X Status Register Bit Definitions**

Bit(s)	Name	Type	Description
0	Operation complete	Read-only	Indicates that the DMA operation described in this descriptor or a chain ending with this descriptor has completed successfully.
1	DMA Write Transaction error	Read-only	An AXI error was returned by the target slave during an AXI write transaction.
2	DMA Read Transaction error	Read-only	An AXI error was returned by the target slave during an AXI read transaction.
3	Invalid buffer descriptor	Read-only	An attempt was made to initiate a DMA operation on an invalid descriptor.
9:4	Descriptor Number	Read-only	0-31: Internal descriptor number that the status information is related to. 32: Refer to the *External Descriptor Address* register to determine the address of the external buffer descriptor to which the status information is associated. 33: Stream operation complete. Refer to the *External Descriptor Address* register to determine the address of the stream descriptor to which the status information is associated.
10	Error Flag	Read only	SB_CORRECT Flag from Buffer descriptors Memory instance.
11	Error Flag	Read only	DB_DETECT Flag from Buffer descriptors Memory instance.
12	Error Flag	Read only	SB_CORRECT Flag from DMA Arbitration Memory instance.
13	Error Flag	Read only	DB_DETECT Flag from DMA Arbitration Memory instance.
14	Error Flag	Read only	SB_CORRECT Flag from Memory Map Cache Memory instance.
15	Error Flag	Read only	DB_DETECT Flag from Memory Map Cache Memory instance.
16	Error Flag	Read only	SB_CORRECT Flag from Stream Cache Memory instance.
17	Error Flag	Read only	DB_DETECT Flag from Stream Cache Memory instance.
18	Error Flag	Read only	SB_CORRECT Flag from Interrupt FIFO Memory instance.
19	Error Flag	Read only	DB_DETECT Flag from Interrupt FIFO Memory instance.
31:20	Reserved	Read only	Reserved – Returns all zeros if read.

Note:

- SB_CORRECT flag signifies single bit error corrected and DB_DETECT flag signifies double-bit error detected. The ECC flags (SB_CORRECT and DB_DETECT) will be only available when the ECC option is selected.
- The buffer descriptors memory contains the information of descriptors configured through the AXI4-lite interface.
- The cache memory is used to store the data to be transferred from source to destination. Similarly, stream cache memory stores the stream of data from the AXI4-stream interface.
- The Interrupt FIFO memory is used to store status information of interrupt associated with a particular descriptor.

5.4 Interrupt X Mask Register

The *Interrupt X Mask* register allows control masters to configure the flags in the *Interrupt X Status* register which generate an Interrupt X interrupt to the control master when asserted. Table 20 describes the *Interrupt X Mask* register.

Table 20 • **Interrupt X Mask Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x04	Interrupt X Mask Register	Read-write	32	0x00	Configure flags in the *Interrupt X Status* register that can generate an interrupt.

Table 21 • **Interrupt X Mask Register Bit Definitions**

Bit(s)	Name	Type	Description
0	Operation complete	Read-write	0: No interrupt generated when the corresponding bit in the *Interrupt X status* register is asserted. 1: Assert an interrupt when a DMA operation completes.
1	DMA Write Transaction error	Read-write	0: No interrupt generated when the corresponding bit in the *Interrupt X status* register is asserted. 1: Assert an interrupt when an AXI write error is detected.
2	DMA Read Transaction error	Read-write	0: No interrupt generated when the corresponding bit in the *Interrupt X status* register is asserted. 1: Assert an interrupt when an AXI read error is detected.
3	Invalid buffer descriptor	Read-write	0: No interrupt generated when the corresponding bit in the *Interrupt X status* register is asserted. 1: Assert an interrupt when an invalid descriptor is processed.
31:4	Reserved	Read-only	Reserved – Returns all zeros if read.

Note: It is mandatory to configure an interrupt mask register for a particular descriptor.

Note: Invalid buffer descriptor interrupt will be asserted if the DB_DETECT flag of Buffer descriptor memory is asserted.

5.5 Interrupt X Clear Register

The *Interrupt X Clear* register provides control masters with a means to clear all status bits asserted in the *Interrupt X Status* register. The Interrupt X interrupt output will remain asserted until all asserted, unmasked status bits have been cleared by writing to the corresponding bit in this register. Table 22 describes the *Interrupt X Clear* register.

Table 22 • **Interrupt X Clear Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x08	Interrupt X Clear Register	Write-only	32	0x00	Clear flags asserted in the *Interrupt X Status* register.

Table 23 • **Interrupt X Clear Register Bit Definitions**

Bit(s)	Name	Type	Description
0	Operation complete	Write-only	Writing a ‘1’ to this bit clears the corresponding bit in the *Interrupt X Status* register.
1	DMA Write Transaction	Write-only	Writing a ‘1’ to this bit clears the corresponding bit in the *Interrupt X Status* register.
2	DMA Read Transaction	Write-only	Writing a ‘1’ to this bit clears the corresponding bit in the *Interrupt X Status* register.
3	Invalid buffer descriptor	Write-only	Writing a ‘1’ to this bit clears the corresponding bit in the *Interrupt X Status* register.
31:4	Reserved	Read-only	Reserved – Returns all zeros if read.

5.6 Interrupt X External Descriptor Register

The *Interrupt X External Descriptor* register provides the address of the external descriptor to which the Interrupt X Status data is related. Table 24 describes the *Interrupt X External Descriptor* register. This register should only be referenced if the contents of the *Interrupt X Status* register *Descriptor Number* field is either 32 or 33.

Table 24 • **Interrupt X External Descriptor Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x0C	Interrupt X External Descriptor Register	Read-only	32	0x00	Address of the external or stream descriptor with which the contents of the *Interrupt Status* register is associated.

Table 25 • **Interrupt X External Descriptor Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Descriptor Address	Read-only	Address of the external or stream descriptor to which the contents of the *Interrupt Status* register is related. The contents of this register should be ignored and will return all zeros if the contents of the *Descriptor Number* field in the *Interrupt Status* register is something other than 32 or 33.

5.7 Descriptor X Configuration Register

The *Descriptor X Configuration* register holds the configuration for internal descriptor X. Table 26 describes the *Interrupt X Configuration* register. The descriptor configuration is maintained between operations on this descriptor, with the exception of the *source data valid* and *destination data ready* flow control bits, which must be reset each time the descriptor is processed.

Table 26 • **Descriptor X Configuration Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x00	Descriptor X Configuration Register	Read-write	32	N/A	Register for configuring the DMA operation performed from by descriptor.

Table 27 • **Descriptor X Configuration Register Bit Definitions**

Bit(s)	Name	Type	Description
1:0	Source Operation	Read-write	Store operation type: 0: No operation – Suits pointing at external descriptors 1: Incrementing address 2: Fixed address 3: Unused
3:2	Destination Operation	Read-write	Forward operation type: 0: No operation – Suits pointing at external descriptors 1: Incrementing address 2: Fixed address 3: Unused
9:4	Unused	Read-only	Reserved – Returns all zeros if read.
10	Chain	Read-write	0: Buffer descriptor describes a single DMA operation or the last descriptor in a non-cyclic chain. 1: Descriptor is part of a chain
11	External Descriptor	Read-write	The content of this field is irrelevant if the chain bit is not set. 0: Internal buffer descriptor 1: External buffer descriptor
12	Interrupt on Process	Read-write	Generates an interrupt when this descriptor has been processed as part of a chain. Setting this bit for the last descriptor in a non-cyclic chain has no effect. **Note:** DMA operations are not halted on this channel when an interrupt is generated via this bit. DMA operations will only be paused as a result of this bit being set if the Interrupt Status bit associated with this descriptor is not cleared in the *Interrupt Clear* register after an interrupt has been fired resulting in the interrupt queue associated with this descriptor (and possibly other channels) becoming full.
13	Source Data Valid	Read-write	Indicates that data is valid at the source. Useful for chain and cyclic operations to prevent the DMA controller from hogging the AXI DMA bus waiting for valid data at the source. The operation defined in this descriptor will not be performed until this bit is set. The *Source Data Valid* bit is cleared by the DMA controller every time that the operation defined in this descriptor has been completed. The control master (firmware) must set this bit again for cyclic operations.
14	Destination Data Ready	Read-write	Indicates that there is room for data at the destination. Useful for chain and cyclic operations to prevent the DMA controller from hogging the AXI DMA bus waiting for a buffer to be allocated at the destination. The operation defined in this descriptor will not be performed until this bit is set. The *Destination Data Ready* bit is cleared by the DMA controller every time that the operation defined in this descriptor has been completed. The control master (firmware) must set this bit again for cyclic operations.

Table 27 • **Descriptor X Configuration Register Bit Definitions *(continued)***

15	Descriptor Valid	Read-write	Indicates that this is a valid descriptor. Provides firmware with a mechanism to safely de-allocate buffer descriptors and prevents lock ups from external start inputs being triggered at startup before firmware has defined a valid descriptor. This bit is automatically cleared when any field of this *Configuration* register is written. The bit should be subsequently reset by firmware when the descriptor is valid. This field needs to be set from a UIC script if a UIC script is used to predefine descriptors in memory. **Note:** It is not recommended to modify descriptors that are kicked-off using an external start input. Once a DMA operation has been initiated this bit is no longer referenced. Therefore an *invalid descriptor* interrupt will not be generated if a descriptor in operation is modified.
31:15	Reserved	Read-only	Reserved – Returns all zeros if read.

Note: Descriptor X configuration register must be written after writing all registers for a particular Descriptor X.

5.8 Descriptor X Byte Count Register

The *Descriptor X Byte Count* register is used to specify the number of bytes to transfer by the DMA operation described in descriptor X. Table 28 describes the *Descriptor X Byte Count* register.

Table 28 • **Descriptor X Byte Count Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x04	Descriptor X Byte Count Register	Read-write	32	N/A	Number of bytes to be transferred in the DMA operation.

Table 29 • **Descriptor X Byte Count Register Bit Definitions**

Bit(s)	Name	Type	Description
22:0	Number of Bytes	Read-write	Number of bytes to be transferred as part of the DMA operation. Maximum permitted is ~ 8 MB (8,388,608 bytes).
31:22	Reserved	Read-only	Reserved – Returns all zeros if read.

5.9 Descriptor X Source Address Register

The *Descriptor X Source Address* register is used to specify the start address of the store part of the DMA operation described in descriptor x. Table 30 describes the *Descriptor X Source Address* register.

Table 30 • **Descriptor X Source Address Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x08	Descriptor X Source Address Register	Read-write	32	N/A	Start address to source data from for the DMA store operation.

Table 31 • **Descriptor X Source Address Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Source address	Read-write	Address to start the AXI read operation from during a store and forward operation.

5.10 Descriptor X Destination Address Register

The *Descriptor X Destination Address* register is used to specify the start address of the forward part of the DMA operation described in descriptor x. Table 32 describes the *Descriptor X Destination Address* register.

Table 32 • **Descriptor X Destination Address Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x0C	Descriptor X Destination Address Register	Read-write	32	N/A	Start address to forward data to for the DMA forward operation.

Table 33 • **Descriptor X Destination Address Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Destination address	Read-write	Address to start the AXI write operation to during a store and forward operation.

5.11 Descriptor X Next Descriptor Number/Address Register

The *Descriptor X Next Descriptor Number/Address* register is used to specify the address/number of the next descriptor in the chain, if descriptor x forms part of a cyclic chain or is part of a non-cyclic chain and is not the last descriptor in the chain. Table 34 describes the *Descriptor X Next Descriptor Number/Address* register.

Table 34 • **Descriptor X Next Descriptor Number/Address Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
Offset 0x10	Descriptor X Next Descriptor Number/Address Register	Read-write	32	N/A	Descriptor number/address of the next internal/external descriptor in the chain.

Table 35 • **Descriptor X Next Descriptor Number/Address Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Descriptor Number/Address	Read-write	Address/ number of the next descriptor in the chain. This register is ignored if the *Chain* bit is not set in the *Configuration* register of this descriptor. The value specified to this register depends on the *External Descriptor* bit in the *Configuration* register: 0: Specify the internal descriptor number 1: Specify the address of the external descriptor

5.12 Stream Address 0 Register

The *Stream Address 0* register is used to store the address of the stream descriptor associated with AXI4-Stream transactions with TDEST[1:0] set to 0b00. Table 36 describes the *Stream Address 0* register.

Table 36 • **Stream Address 0 Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
0x460	Stream Address 0 Register	Read-write	32	0x00	External stream descriptor address associated with AXI4-Stream transactions of with TDEST = 0b00.

Table 37 • **Stream Address 0 Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Stream Descriptor Address	Read-write	Address of the *Configuration* register of the stream descriptor in the AXI4 memory-map address space.

5.13 Stream Address 1 Register

The *Stream Address 1* register is used to store the address of the stream descriptor associated with AXI4-Stream transactions with TDEST[1:0] set to 0b01. Table 38 describes the *Stream Address 1* register.

Table 38 • **Stream Address 1 Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
0x464	Stream Address 1 Register	Read-write	32	0x00	External stream descriptor address associated with AXI4-Stream transactions of with TDEST = 0b01.

Table 39 • **Stream Address 1 Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Stream Descriptor Address	Read-write	Address of the *Configuration* register of the stream descriptor in the AXI4 memory-map address space.

5.14 Stream Address 2 Register

The *Stream Address 2* register is used to store the address of the stream descriptor associated with AXI4-Stream transactions with TDEST[1:0] set to 0b10. Table 40 describes the *Stream Address 2* register.

Table 40 • **Stream Address 2 Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
0x468	Stream Address 2 Register	Read-write	32	0x00	External stream descriptor address associated with AXI4-Stream transactions of with TDEST = 0b10.

Table 41 • **Stream Address 2 Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Stream Descriptor Address	Read-write	Address of the *Configuration* register of the stream descriptor in the AXI4 memory-map address space.

5.15 Stream Address 3 Register

The *Stream Address 3* register is used to store the address of the stream descriptor associated with AXI4-Stream transactions with TDEST[1:0] set to 0b11. Table 42 describes the *Stream Address 3* *r*egister.

Table 42 • **Stream Address 3 Register**

AxADDR [10:0]	Register Name	Type	Width	Reset Value	Description
0x46C	Stream Address 3 Register	Read-write	32	0x00	*External stream descriptor address* associated with AXI4-Stream transactions of with TDEST = 0b11.

Table 43 • **Stream Address 3 Register Bit Definitions**

Bit(s)	Name	Type	Description
31:0	Stream Descriptor Address	Read-write	Address of the *Configuration* register of the stream descriptor in the AXI4 memory-map address space.

6 Tool Flows

6.1 Licensing

No license is required for the use of this core.

6.2 RTL

Complete RTL source code is provided for the core and testbench.

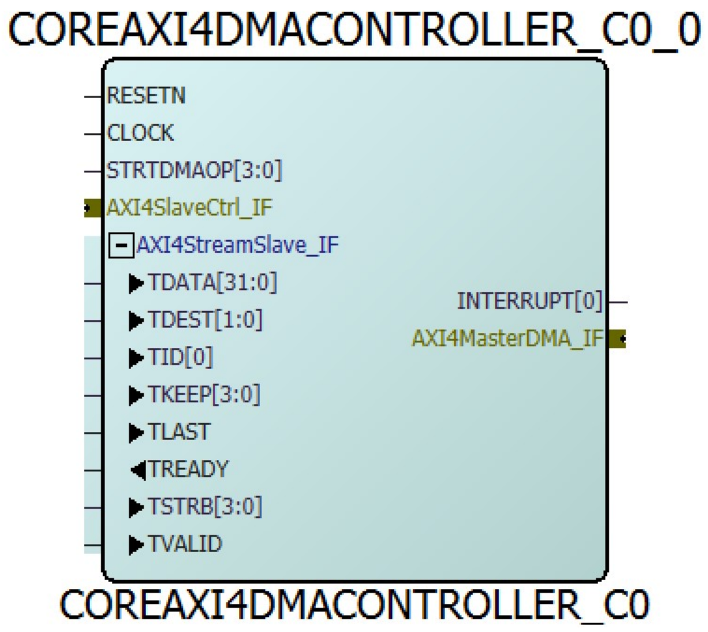
6.3 SmartDesign

CoreAXI4DMAController is preinstalled in the SmartDesign IP deployment design environment. An example instantiated view is shown in the following figure. The core can be configured using the configuration GUI within SmartDesign, as shown in Figure 7, page 14. To know how to create a SmartDesign project using the IP cores, refer to the *Libero SoC documents page* and use the latest SmartDesign user guide.

After configuring and generating the core instance, basic functionality can be simulated using the testbench supplied with CoreAXI4DMAController. The testbench parameters automatically adjust to the CoreAXI4DMAController configuration.

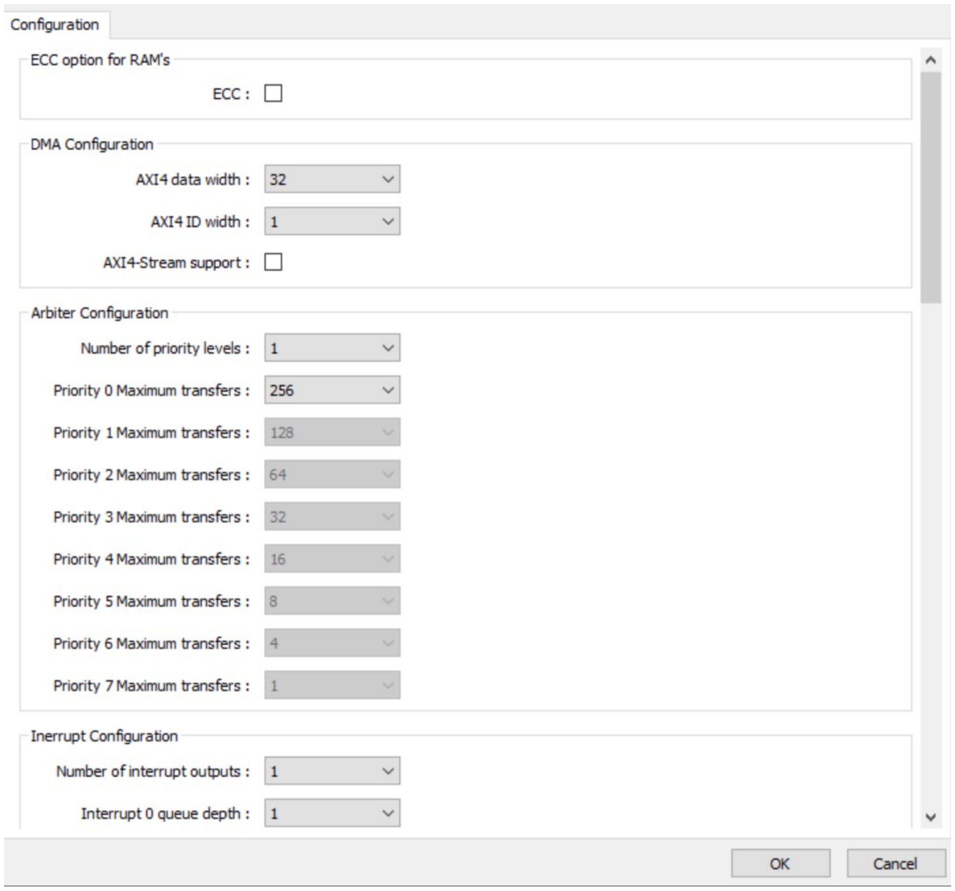
Note: Certain RTL within CoreAXI4DMAController is automatically created when the core instance is generated in SmartDesign based upon the Priority and Interrupt associations of descriptors. For this reason, top-level parameters should only be modified in the CoreAXI4DMAController configurator in SmartDesign to ensure that the underlying RTL reflects the parameter modifications for both simulation and synthesis flows.

Figure 11 • **SmartDesign CoreAXI4DMAController Instance View**



The core is configured using the configuration GUI within SmartDesign, as shown in Figure 12.

Figure 12 • **SmartDesign CoreAXI4DMAController Configuration Dialog Box**


Configuration

ECC option for RAM's

ECC : ☐

DMA Configuration

AXI4 data width : 32

AXI4 ID width : 1

AXI4-Stream support : ☐

Arbiter Configuration

Number of priority levels : 1

Priority 0 Maximum transfers : 256

Priority 1 Maximum transfers : 128

Priority 2 Maximum transfers : 64

Priority 3 Maximum transfers : 32

Priority 4 Maximum transfers : 16

Priority 5 Maximum transfers : 8

Priority 6 Maximum transfers : 4

Priority 7 Maximum transfers : 1

Inerrupt Configuration

Number of interrupt outputs : 1

Interrupt 0 queue depth : 1

6.4 Simulation Flows

The user testbench for CoreAXI4DMAController is included in all releases.

To run simulations:

1. Select the user testbench flow within SmartDesign.
2. Click **Save and Generate** in the Generate pane. The user testbench is selected through the Core Testbench Configuration GUI.

When SmartDesign generates the Libero project, it installs the user testbench files.

To run the user testbench:

1. Set the design root to the CoreAXI4DMAController instantiation in the Libero design hierarchy pane.
2. Click **Simulation** in the Libero Design Flow window. This invokes ModelSim and automatically run the simulation.

6.5 Synthesis

To run synthesis on the CoreAXI4DMAController, set the design root to the IP component instance and run the synthesis tool from the Libero design flow pane.

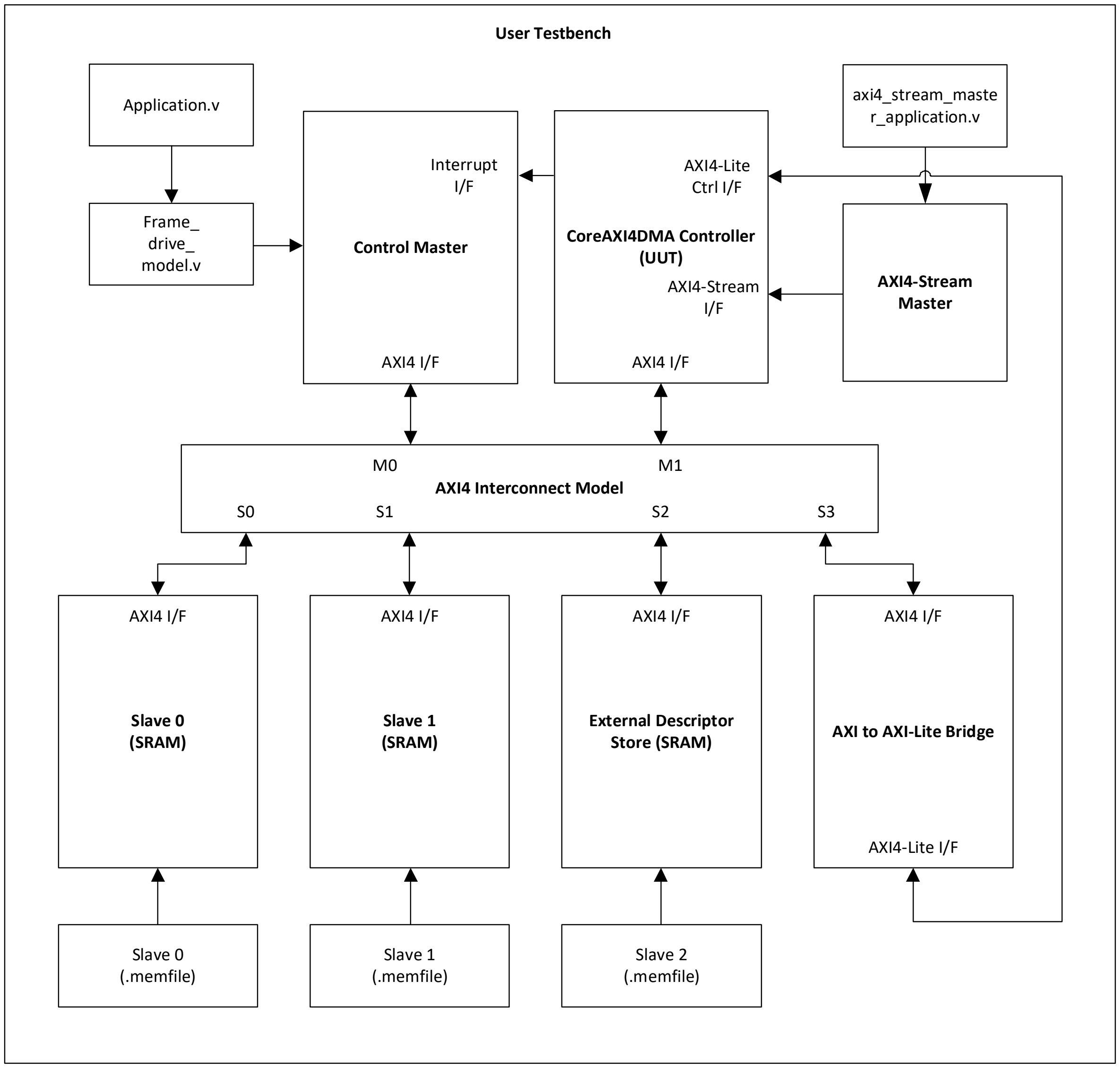
6.6 Place-and-Route

After the design is synthesized, run the compilation and then place-and-route the tools. CoreAXI4DMAController requires no special place-and-route settings.

7 Test-bench Operation and Modification

The testbench packaged along with CoreAXI4DMAController includes an instantiation of CoreAXI4DMAController, connected up to an AXI4 subsystem, as shown in Figure 13.

Figure 13 • **User Testbench**



The CoreAXI4DMAController testbench environment consists of the following components:

- **Control Master:** The Control Master component emulates the operation of an AXI4 bus master. The operations carried out by the Control master are issued from the application.v file, which calls the functions implemented in the firmware_driver_model.v file to model the intended operation of CoreAXI4DMAController from a firmware driver and application perspective. Users can modify the application.v script in order to simulate custom cases.
- **CoreAXI4DMAController:** An instance of CoreAXI4DMAController, the unit under test (UUT). In the user testbench, the DMA interface of the core is connected to a master slot on the AXI4 interconnect model. External fetches are fetched from the External Descriptor store over the AXI4 DMA interface. The interrupt output of the core is connected to the Control Master to inform the master when a DMA operation has completed or failed. The test cases provided in the application.v file perform data transfers between the Slave0 and Slave1 AXI4 slave RAM blocks.
- **AXI4 Interconnect Model:** A lightweight AXI4 interconnect simulation model facilitating the connection of two masters to four slaves. Round-robin arbitration is performed to share control between the two masters. The interconnect implements separate channels for the write address, read address, write data, read data, and write response AXI4 channels.
- **Slave 0 (SRAM):** AXI4 RAM wrapper simulation model. Configurable address width supported to allow larger RAM blocks to be implemented. The contents of the RAM is initialized by slave0.mem.
- **Slave 1 (SRAM):** AXI4 RAM wrapper simulation model. Configurable address width supported to allow larger RAM blocks to be implemented. The contents of the RAM is initialized by slave1.mem.
- **External Descriptor Store (SRAM):** The purpose of this AXI4 RAM wrapper simulation model is to store external descriptors describing DMA operations. When pointed to via an internal descriptor, CoreAXI4DMAController fetches the descriptor from the External Descriptor Store through the AXI4 DMA interface and executes the operation described in the descriptor, before clearing the *source data valid* and *destination data ready* bits for each executed descriptor in the External descriptor store on completion of the DMA operation. The Control Master must then determine if the slave memory related to the external descriptor DMA operation is ready before re-setting the *source data valid* and *destination data ready* bits for that descriptor in the External Descriptor Store. The contents of the External Descriptor Store is initialized by DscrptrStore.mem.
- **AXI4 to AXI4-Lite Bridge:** An AXI4 to AXI4-Lite protocol bridge simulation model to facilitate an AXI4 master connecting to the AXI4-Lite control interface of CoreAXI4DMAController. Buffering is performed within the simulation model to break sequential burst transactions into non-sequential, single beat transactions on the AXI4-Lite side.
- **AXI4-Stream Master:** An AXI-Stream master model is included in the user testbench when the *AXI4_STREAM_IF* parameter is set to 1 when CoreAXI4DMAController is instantiated. This generates the AXI4-Stream transactions defined in the axi4_stream_master_application.v file.

Note: The CoreAXI4DMAController testbench only executes for the default configuration of CoreAXI4DMAController as RTL is generated based on user configuration parameters when the SmartDesign sheet containing this instance of CoreAXI4DMAController gets *generated*. In addition, no data width translation logic is included in the testbench between the interconnect slave port 3 and the AXI4-Lite bridge to the AXI4-Lite Control interface of CoreAXI4DMAController (fixed data width of 32-bit on the Control interface.

Note: Ignore the warnings during simulation of user testbench, if any.