DYNANIC

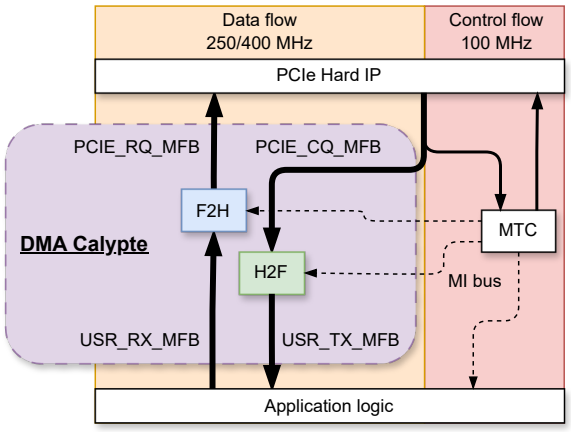
DMA Calypte IP Core Datasheet

1 Features

- **Low-latency data transfer:** Designed to minimize latency, with a focus on both Host-to-FPGA (H2F) and FPGA-to-Host (F2H) directions.
- **FPGA vendor independence:** Compatible with various PCIe IP cores
- **Configurable bus data width and PCIe interface:** Allows for customization according to specific requirements
- **Multiple DMA channels:** Enables the transmission of packets to be divided into a number of queues (DMA channels)
- Custom transmission protocol Multi-Frame bus(MFB), Multi-Value bus (MVB) and Memory-Interface bus (MI)
- Open-source driver and tools (NDK-SW)
- DPDK support

2 General Description

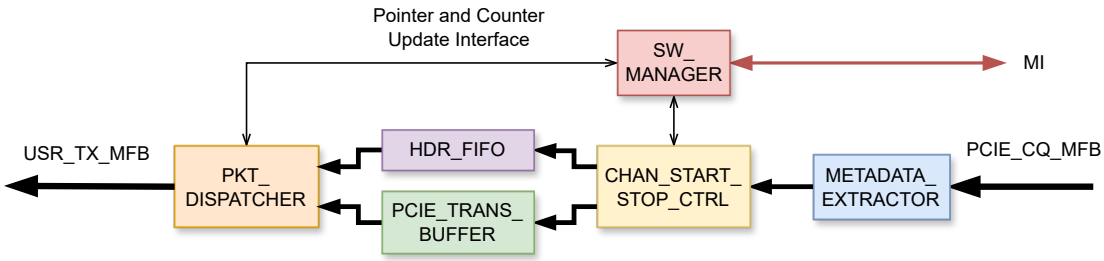
This module allows simple DMA access to the host memory over the PCI Express interface for both, Host-to-FPGA (H2F) and FPGA-to-Host (F2H) directions. The design was primarily focused on the lowest latency possible. The module contains two controllers for each direction: the F2H (formerly named RX) and the H2F (formerly named TX) controller. These allow for the full-duplex transmission of packet data from/to the host memory. The controllers connect to the surrounding infrastructure using the MFB bus. The packet transmission is part of the Data Flow which is distinguished from the Control Flow. The Control Flow is established using the MI Bus that accesses internal Control and Status (C/S) registers. Each of the controllers contains multiple virtual channels that share the same MFB bus but have separate address spaces in the host memory. This allows for concurrent access from the host system. The block scheme of the DMA module is provided in the following figure:



Block Diagram of DMA Calypte

3 TX DMA Calypte

The TX DMA Calypte controller is a standalone component of the DMA Calypte module designed for data transfers in the H2F (Host-to-FPGA) direction. The following figure shows the internal layout of the controller:



Schematic view of the internal blocks within the TX DMA Calypte controller

The component accepts PCIe transactions on the PCIE_CQ_MFB input and dispatches packets towards the application logic from the USR_TX_MFB output. Each packet consists of multiple PCIe transactions of various lengths that are merged in the internal data buffer. Every packet is followed by its DMA header that identifies it in the data buffer. The packets are transferred over multiple virtual channels, each comprising a separate buffer space for its packets.

3.1 Control/Status Registers

To enable software control, the controller utilizes an address space with configuration/status (C/S) registers. Currently, each channel has its own register space, each having a size of 128 bytes. The first channel's registers are located at address 0x00, the second at 0x80, the third at 0x100, and so on. These registers are connected to the MI Bus. The C/S register set for one channel is shown in the following table:

Address	Name	Access Permission (FPGA/Host)	Description
0x00	Control	R/W	Bit 0: Set to 1 to request the enable of a channel; set to 0 to request a stop.
0x04	Status	W/R	Bit 0: Set to 1 if a channel is enabled; 0 if disabled.
0x08	Reserved	N/A	-
0x0C	Reserved	N/A	-
0x10	Software data pointer (SDP)	R/W	Write pointer for data (up to 16 bits)
0x14	Software header pointer (SHP)	R/W	Write pointer for headers (up to 16 bits)

0x18	Hardware data pointer (HDP)	R/W	Read pointer for data (up to 16 bits)
0x1C	Hardware header pointer (HHP)	R/W	Read pointer for headers (up to 16 bits)
0x20	Reserved	N/A	-
0x24	Reserved	N/A	-
0x28	Reserved	N/A	-
0x2C	Reserved	N/A	-
0x30	Reserved	N/A	-
0x34	Reserved	N/A	-
0x38	Reserved	N/A	-
0x3C	Reserved	N/A	-
0x40	Reserved	N/A	-
0x44	Reserved	N/A	-
0x48	Reserved	N/A	-
0x4C	Reserved	N/A	-
0x50	Reserved	N/A	-
0x54	Reserved	N/A	-
0x58	Data pointer mask (DPM)	W/R	Determines data buffer size
0x5C	Header pointer mask (HPM)	W/R	Determines header buffer size
0x60	Received packetsL	W/RW (Strobe)	Counter of received packets (lower part)
0x64	Received packetsH	W/RW (Strobe)	Counter of received packets (upper part)
0x68	Received bytesL	W/RW (Strobe)	Counter of received bytes (lower part)
0x6C	Received bytesH	W/RW (Strobe)	Counter of received bytes (upper part)
0x70	Discarded packetsL	W/RW (Strobe)	Counter of discarded packets (lower part)
0x74	Discarded packetsH	W/RW (Strobe)	Counter of discarded packets (upper part)
0x78	Discarded bytesL	W/RW (Strobe)	Counter of discarded bytes (lower part)
0x7C	Discarded bytesH	W/RW (Strobe)	Counter of discarded bytes (upper part)

Note

Some registers have a strobe functionality in which case specific writes on a counter's address need to be issued from the host in order to manipulate a counter's register:

- **0x0:** Resets a counter and its register
- **0x1:** Samples a value of a counter to its register
- **0x2:** Combination of the two previous, e.g. a value of a counter is sampled to its register and the counter is put to reset.

3.2 Start sequence

The *Control* and *Status* registers are the most important ones in terms of ensuring the channel's activity. When a start of a channel is requested, some registers need to be initialized from the MI bus:

- The SDP and SHP pointer registers need to be initialized to 0.
- A write of value *0b1* to the Control register is issued. This immediately starts the required channel that responds by setting the Status register to *1b1*. The controller is now ready to transmit data from the software.

3.3 Stop sequence

If a stop of a channel is requested, the *0b0* value is written to its Control register. The controller completes the dispatch of a currently processed packet (if there is any for this channel) and all previously received. This is indicated by an update of HDP and HHP registers to newer values which, in the end, have equal value as the SDP and SHP registers. This signifies the successful stop sequence and the controller indicates this by setting the Status register of the stopped channel to *0b0*. The stopped channel does not forward any incoming data and drops them (however, every other enabled channel can still receive the data from the software).

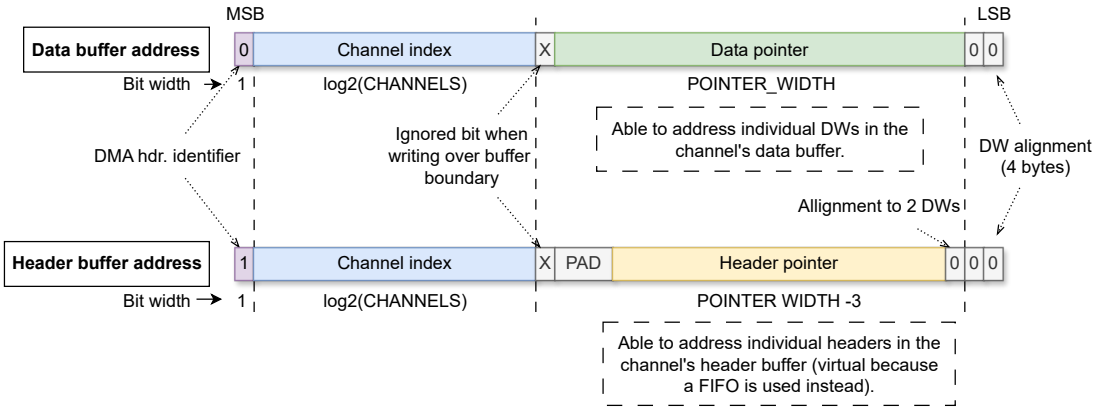
Note

Although many channels can be deactivated at once from the software, the controller deactivates them sequentially. This also applies to the execution of the start sequence.

3.4 Data transmission

The data transmission occurs over one or multiple channels using the shared MFB bus. Every incoming packet is received as a set of PCIe transactions. The PCIe address within each transaction's header contains information about the channel on

which this transaction gets transferred, the pointer to the data/header buffer, and the information on whether the transaction contains packet data or a DMA header. The address format is shown in the following figure:



PCIe Address Layout

A structure of the PCIe address as processed by the METADATA_EXTRACTOR component. The X bit gets ignored but this address extension minimizes alignment overhead in the software when writing a packet that crosses the buffer boundary.

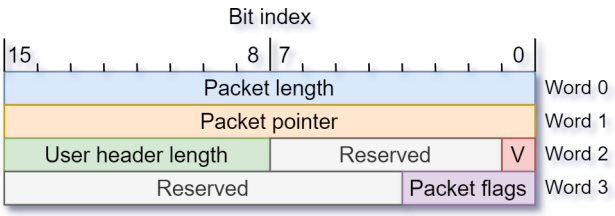
The PCIe header is cut off from a PCIe transaction when passed through METADATA_EXTRACTOR and metadata are further transported with the beginning of a transaction. The second component START_STOP_CTLR handles transaction forwarding based on the channel's activity. This component drops transactions when the channel is inactive and forwards them when active. It is also a direct respondent to the SW_MANAGER when the Start/Stop sequence takes place. Accepted transactions are forwarded either to PCIE_TRANS_BUFFER or HDR_FIFO (instantiated as dma_hdr_fifo_i in the top-level entity) based on their content.

The different types of buffers have been chosen based on the ordering of PCIe transactions. To ensure sufficient throughput, the software driver incorporates write-combining [1] of packet data. However, this introduces weak ordering of transactions that are written to specific addresses in the BRAM array of the PCIE_TRANS_BUFFER. After each packet gets written from the host software, memory fencing is triggered and the DMA header is written after that. The DMA header is shown in the following figure and follows the same format as in the case of RX_DMA_CALYPTE.

After fencing is issued, it is ensured that DMA header always comes in order with packet data. Enforcing such ordering ensures that DMA header arrives after all of

[1]Write-combining (WC) is a mapping of a memory region (in case of the H2F controller, the mapping PCI BAR in the driver) that improves processor write performance by combining multiple write transactions to a bigger burst. However, WC employs a weakly ordered model a delayed dispatch of transactions. Therefore, the fencing operation needs to take place to ensure data coherency. A call to fence ensures the completion of all previously triggered load and store instructions.

packet data have been received. The header is simply stored in a FIFO storage that is common to all channels.



DMA Header format

A DMA header format is the same as used in the RX DMA controller with the only exception that the V bit is ignored. The User header length and Packet flags fields get copied and transported with the beginning of a packet.

Note

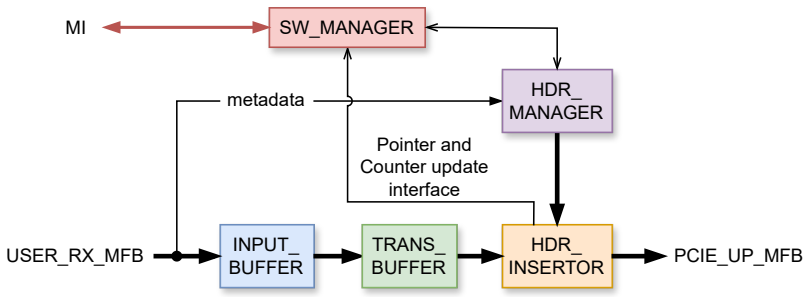
Since every channel is controlled by a separate process in the software, the data can be written in parallel. These, however, arrive on a common MFB bus and are thus read sequentially based on the order in which DMA headers arrived.

The PKT_DISPATCHER reads composed packets from the transaction buffer based on the received DMA headers in the header FIFO. The application metadata, such as User header length and Packet flags, are copied from the DMA header of each packet [2]. When the end of a packet is reached, the component updates the read pointers (located in HDP and HHP registers) and also the counter of sent packets/bytes. The register array and packet counters for every channel are instantiated in the SW_MANAGER component, which also triggers the start/stop sequence on a channel.

[2] The metadata follow the general format of the MVB_HDR_META signal as specified in the Header metadata format

4 RX DMA Calypte

The RX DMA Calypte controller is a standalone component of the DMA Calypte module, designed for transferring packet data in the F2H (FPGA-to-Host) direction. The block diagram below illustrates its internal architecture:



Schematic view of the internal blocks within the RX DMA Calypte controller

The controller accepts packets on the USER_RX_MFB bus and generates PCIe transactions on its output, the PCIE_UP_MFB bus. Each packet is segmented into 128-byte transactions that are dispatched as individual PCIe transactions. This 128-byte segment size has been selected as a compromise between high throughput and low latency. While future modifications to the segment size are possible, they will not affect the entity's port configuration.

4.1 Control/Status Registers

To enable software control, the controller utilizes an address space with configuration/status (C/S) registers. Currently, each channel has its own register space, each having a size of 128 bytes. The first channel's registers are located at address 0x00, the second at 0x80, the third at 0x100, and so on. These registers are connected to the MI Bus. The C/S register set for one channel is shown in the following table:

Address	Name	Access Permission (FPGA/Host)	Description
0x00	Control	R/W	Bit 0: Set to 1 to request the enable of a channel; set to 0 to request a stop.
0x04	Status	W/R	Bit 0: Set to 1 if a channel is enabled; 0 if disabled.
0x08	Reserved	N/A	-
0x0C	Reserved	N/A	-
0x10	Software data pointer (SDP)	R/W	Write pointer for data (up to 16 bits)

0x14	Software header pointer (SHP)	R/W	Write pointer for headers (up to 16 bits)
0x18	Hardware data pointer (HDP)	R/W	Read pointer for data (up to 16 bits)
0x1C	Hardware header pointer (HHP)	R/W	Read pointer for headers (up to 16 bits)
0x20	Reserved	N/A	-
0x24	Reserved	N/A	-
0x28	Reserved	N/A	-
0x2C	Reserved	N/A	-
0x30	Reserved	N/A	-
0x34	Reserved	N/A	-
0x38	Reserved	N/A	-
0x3C	Reserved	N/A	-
0x40	Data baseL	R/W	Base addres of the data buffer in a host memory (lower 32 bits).
0x44	Data baseH	R/W	Base addres of the data buffer in a host memory (upper 32 bits).
0x48	Header baseL	R/W	Base addres of the header buffer in a host memory (lower 32 bits).
0x4C	Header baseH	R/W	Base addres of the header buffer in a host memory (upper 32 bits).
0x50	Reserved	N/A	-
0x54	Reserved	N/A	-
0x58	Data pointer mask (DPM)	W/R	Determines data buffer size
0x5C	Header pointer mask (HPM)	W/R	Determines header buffer size
0x60	Received packetsL	W/RW (Strobe)	Counter of received packets (lower part)
0x64	Received packetsH	W/RW (Strobe)	Counter of received packets (upper part)
0x68	Received bytesL	W/RW (Strobe)	Counter of received bytes (lower part)
0x6C	Received bytesH	W/RW (Strobe)	Counter of received bytes (upper part)
0x70	Discarded packetsL	W/RW (Strobe)	Counter of discarded packets (lower part)
0x74	Discarded packetsH	W/RW (Strobe)	Counter of discarded packets (upper part)
0x78	Discarded bytesL	W/RW (Strobe)	Counter of discarded bytes (lower part)
0x7C	Discarded bytesH	W/RW (Strobe)	Counter of discarded bytes (upper part)

Note

Counter registers have a strobe functionality that requires specific writes to manipulate a counter's register from the host:

- **0x0:** Resets a counter and its register
- **0x1:** Samples a value of a counter to its register
- **0x2:** Combination of the two previous, e.g. a value of a counter is sampled to its register and the counter is put to reset.

Write to one counter register affects all counters as well as their registers. This ensures value coherency between the counters.

4.2 Start sequence

The *Control* and *Status* registers are the most important ones in terms of ensuring the channel's activity. When a start of a channel is requested, several registers need to be initialized from the MI bus:

- The *DataBaseL*, *DataBaseH*, *HeaderBaseL* and *HeaderBaseH* registers need to contain valid addresses that were previously reserved in the host memory. This memory needs to be initialized as DMA-able.
- The SDP and SHP pointer registers need to be initialized to 0.
- Finally, a write of value 0b1 to the Control register is issued. This immediately starts the required channel that responds by setting the Status register to 1b1. The controller is now ready to transmit application data.

4.3 Stop sequence

If a stop of a channel is requested, the 0b0 value is written to its Control register. The controller completes the dispatch of a currently processed packet (if there is any for this channel) and waits for the software to process all of the packets on this channel. This is indicated by an update of SDP and SHP registers to newer values which are now equal to the HDP and HHP registers. This signifies the successful stop sequence and the controller indicates this by setting the Status register of the stopped channel to 0b0. The stopped channel does not forward any incoming data and drops them (however, every other enabled channel can still send its data to the host memory).

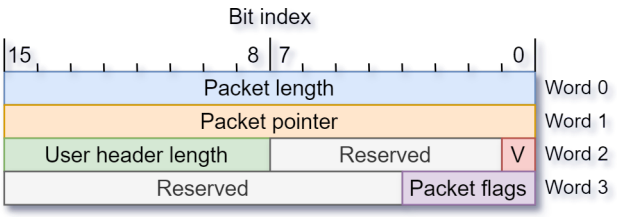
Note

Although many channels can be deactivated at once from the software, the controller deactivates them sequentially. This also applies to the execution of the start sequence.

4.4 Data transmission

The data transmission occurs over one or multiple channels using the shared MFB bus. Every incoming packet is accompanied by an index of a channel processing this packet, the length of application metadata which allows to distinguish a user header that precedes the packet data, and application-specific flags [3]. These fields get separated from the packet data when entering the F2H controller.

In the INPUT_BUFFER, every packet is aligned to the beginning of a bus word. This simplifies the buffering in the TRANS_BUFFER that gathers bus words in order to form the 128 B segments [4]. This results in 4 words or 2 words in total for 256-bit or 512-bit bus, respectively (see Supported PCIe Configurations). The buffered segment is prepended with a PCIe header in the HDR_INSERTOR component and dispatched to the PCIe domain with the address of the channel's Data buffer. The component dispatches every other segment in the same way while updating the HDP register value. After the packet has ended and the last segment has been sent, the HDR_INSERTOR sends the last PCIe transaction containing the DMA Header that indicates the delivery of a packet in the host memory. The DMA Headers are stored in a separate Header Buffer within each channel. When the controller sends the DMA Header transaction, the value of the HHP register is updated. The software driver periodically polls the position of a current head pointer (the value of SHP) in the header buffer until it captures valid data. The format of the DMA header can be seen in the following figure:



DMA Header format

A DMA header format is the same as used in the RX DMA controller with the only exception that the V bit is ignored. The User header length and Packet flags fields get copied and transported with the beginning of a packet.

[3] The metadata follow the general format of the MVB_HDR_META signal as specified in the Header metadata format

[4] The reason for buffering the segments is that the PCIe IP does not support any gaps in the data between a transaction's start and end. Because of that, the data needs to be sent continuously before ending the transaction. For more information refer to the *Requester Request Interface* description in the *UltraScale+ Devices Integrated Block for PCI Express Product Guide (PG213)*.

Note

Since every channel is controlled by a separate process in the software, the data can be written in parallel. These, however, arrive on a common MFB bus and are thus read sequentially based on the order in which DMA headers arrived.

The PKT_DISPATCHER reads composed packets from the transaction buffer based on the received DMA headers in the header FIFO. The application metadata, such as User header length and Packet flags, are copied from the DMA header of each packet. When the end of a packet is reached, the component updates the read pointers (located in HDP and HHP registers) and also the counter of sent packets/bytes. The register array and packet counters for every channel are instantiated in the SW_MANAGER component, which also triggers the start/stop sequence on a channel.

5 Interface

This section will provide a information about generic configuration of the DMA Calypte and Interface overview.

5.1 Generic map description

Generic	Type	Default	Description
Settings affecting both RX and TX or the top level entity itself			
DEVICE	string	"ULTRASCALE"	Name of target device, the supported are: • "ULTRASCALE" • "STRATIX10" • "AGILEX"
USR_MFB_REGIONS	natural	1	USER MFB interface configuration that is used for user data stream. The allowed configurations are: • (1,4,8,8) • (1,8,8,8)
USR_MFB_REGION_SIZE	natural	8	
USR_MFB_BLOCK_SIZE	natural	8	
USR_MFB_ITEM_WIDTH	natural	8	
HDR_META_WIDTH	natural	24	Width of User Header Metadata information • on RX: added to the DMA header • on TX: extracted from a DMA header
Requester Request (RQ) MFB interface settings. The allowed configurations are:			• (1,1,8,32) • (2,1,8,32)
PCIE_RQ_MFB_REGIONS	natural	2	
PCIE_RQ_MFB_REGION_SIZE	natural	1	
PCIE_RQ_MFB_BLOCK_SIZE	natural	8	
PCIE_RQ_MFB_ITEM_WIDTH	natural	32	
Completer Request (CQ) MFB interface settings. The allowed configurations are:			• (1,1,8,32) • (2,1,8,32)
PCIE_CQ_MFB_REGIONS	natural	2	
PCIE_CQ_MFB_REGION_SIZE	natural	1	
PCIE_CQ_MFB_BLOCK_SIZE	natural	8	
PCIE_CQ_MFB_ITEM_WIDTH	natural	32	
RX DMA controller settings			
RX_CHANNELS	natural	8	Total number of RX DMA Channels (powers of 2, starting at 2)
RX_PTR_WIDTH	natural	16	Width of Software and Hardware Header/DataPointer. • Affects logic complexity (MI C/S registers especially) • Maximum value: 16
USR_RX_PKT_SIZE_MAX	natural	2^{12}	Maximum size of a User packet in bytes (in interval between 60 and 2^{12}, inclusively)

TRBUF_REG_EN	boolean	false	Enables an additional register of the transaction buffer that improves throughput (see Transaction Buffer)
PERF_CNTR_EN	boolean	false	Enables performance counters allowing metrics generation.
TX DMA controller settings			
TX_CHANNELS	natural	8	Total number of TX DMA Channels (powers of 2, starting at 2)
TX_PTR_WIDTH	natural	13	Width of the Hardware Descriptor Pointer • Significantly affects the complexity of the controller (the C/S registers as well as buffers to store packets within each channel). • Maximum value: 13 (restricted as a compromise between the size of a controller and maximum intact size of a packet that the software can dispatch)
USR_TX_PKT_SIZE_MAX	natural	2^{12}	Maximum size of a User packet in bytes (in an interval between 60 and 2^{12}, inclusively)
Optional settings			
DSP_CNT_WIDTH	natural	64	Width of statistical counters within each channel
RX_GEN_EN	boolean	TRUE	Allows to disable one of the controllers in the DMA module
TX_GEN_EN	boolean	TRUE	
ST_SP_DBG_SIGNAL_W	natural	4	Width of the debug signal, do not use unless you know what you are doing
MI_WIDTH	natural	32	Width of MI bus

5.2 Port map description

Port	Dir	Type	Width
Clocks and Resets			
CLK	in	std_logic	
RESET	in	std_logic	
RX DMA User-side MFB			
USR_RX_MFB_META_HDR_META	in	std_logic_vector	log2(RX_CHANNELS)
USR_RX_MFB_META_CHAN	in	std_logic_vector	HDR_META_WIDTH
USR_RX_MFB_DATA	in	std_logic_vector	USR_MFB_REGIONS* USR_MFB_REGION_SIZE* USR_MFB_BLOCK_SIZE* USR_MFB_ITEM_WIDTH
USR_RX_MFB_SOF	in	std_logic_vector	USR_MFB_REGIONS
USR_RX_MFB_EOF	in	std_logic_vector	USR_MFB_REGIONS
USR_RX_MFB_SOF_POS	in	std_logic_vector	USR_MFB_REGIONS* max(1, log2(USR_MFB_REGION_SIZE))
USR_RX_MFB_EOF_POS	in	std_logic_vector	USR_MFB_REGIONS* max(1, log2(USR_MFB_REGION_SIZE* USR_MFB_BLOCK_SIZE))

USR_RX_MFB_SRC_RDY	in	std_logic	
USR_RX_MFB_DST_RDY	out	std_logic	
TX DMA User-side MFB			
USR_TX_MFB_META_CHAN	out	std_logic_vector	log2(TX_CHANNELS)
USR_TX_MFB_META_HDR_META	out	std_logic_vector	HDR_META_WIDTH
USR_TX_MFB_DATA	out	std_logic_vector	USR_MFB_REGIONS* USR_MFB_REGION_SIZE* USR_MFB_BLOCK_SIZE* USR_MFB_ITEM_WIDTH
USR_TX_MFB_SOF	out	std_logic_vector	USR_MFB_REGIONS
USR_TX_MFB_EOF	out	std_logic_vector	USR_MFB_REGIONS
USR_TX_MFB_SOF_POS	out	std_logic_vector	USR_MFB_REGIONS* max(1, log2(USR_MFB_REGION_SIZE))
USR_TX_MFB_EOF_POS	out	std_logic_vector	USR_MFB_REGIONS* max(1, log2(USR_MFB_REGION_SIZE* USR_MFB_BLOCK_SIZE))
USR_TX_MFB_SRC_RDY	out	std_logic	
USR_TX_MFB_DST_RDY	in	std_logic	
Debug signals (Should not be used by the user of the component)			
ST_SP_DBG_CHAN	out	std_logic_vector	log2(TX_CHANNELS)
ST_SP_DBG_META	out	std_logic_vector	ST_SP_DBG_SIGNAL_W
RQ PCIe Interface - Upstream MFB Interface (for sending data to the PCIe Endpoint)			
PCIE_RQ_MFB_DATA	out	std_logic_vector	PCIE_RQ_MFB_REGIONS* PCIE_RQ_MFB_REGION_SIZE* PCIE_RQ_MFB_BLOCK_SIZE* PCIE_RQ_MFB_ITEM_WIDTH
PCIE_RQ_MFB_META	out	std_logic_vector	PCIE_RQ_MFB_META_WIDTH
PCIE_RQ_MFB_SOF	out	std_logic_vector	PCIE_RQ_MFB_REGIONS
PCIE_RQ_MFB_EOF	out	std_logic_vector	PCIE_RQ_MFB_REGIONS
PCIE_RQ_MFB_SOF_POS	out	std_logic_vector	PCIE_RQ_MFB_REGIONS* max(1, log2(PCIE_RQ_MFB_REGION_SIZE))
PCIE_RQ_MFB_EOF_POS	out	std_logic_vector	PCIE_RQ_MFB_REGIONS* max(1, log2(PCIE_RQ_MFB_REGION_SIZE* PCIE_RQ_MFB_BLOCK_SIZE))
PCIE_RQ_MFB_SRC_RDY	out	std_logic	
PCIE_RQ_MFB_DST_RDY	in	std_logic	
CQ PCIe Interface - Downstream MFB Interface (for receiving data from the PCIe Endpoint)			
PCIE_CQ_MFB_DATA	in	std_logic_vector	PCIE_CQ_MFB_REGIONS* PCIE_CQ_MFB_REGION_SIZE* PCIE_CQ_MFB_BLOCK_SIZE* PCIE_CQ_MFB_ITEM_WIDTH
PCIE_CQ_MFB_META	in	std_logic_vector	PCIE_CQ_MFB_META_WIDTH
PCIE_CQ_MFB_SOF	in	std_logic_vector	PCIE_CQ_MFB_REGIONS
PCIE_CQ_MFB_EOF	in	std_logic_vector	PCIE_CQ_MFB_REGIONS
PCIE_CQ_MFB_SOF_POS	in	std_logic_vector	PCIE_CQ_MFB_REGIONS* max(1, log2(PCIE_CQ_MFB_REGION_SIZE))
PCIE_CQ_MFB_EOF_POS	in	std_logic_vector	PCIE_CQ_MFB_REGIONS* max(1, log2(PCIE_CQ_MFB_REGION_SIZE* PCIE_CQ_MFB_BLOCK_SIZE))
PCIE_CQ_MFB_SRC_RDY	in	std_logic	
PCIE_CQ_MFB_DST_RDY	out	std_logic	

MI Interface for S/W access			
MI_ADDR	in	std_logic_vector	MI_WIDTH
MI_DWR	in	std_logic_vector	MI_WIDTH
MI_BE	in	std_logic_vector	MI_WIDTH/8
MI_RD	in	std_logic	
MI_WR	in	std_logic	
MI_DRD	out	std_logic_vector	MI_WIDTH
MI_DRDY	out	std_logic	
MI_ARDY	in	std_logic	

6 Supported PCIe Configurations

The design can be configured for two major PCIe IP configurations. This corresponds to setting the input/output MFB bus interfaces when configuring the DMA_CALYPTE entity.

	AMD UltraScale+ Architecture, Intel Avalon P-Tile	AMD UltraScale+ architecture, Intel Avalon P-Tile/R-Tile
PCI Express configuration	Gen3 x8	Gen3 x16 (AMD), Gen4 x16 (Intel), Gen5 x8 (Intel)
Internal bus width	256 bits	512 bits
Frequency	250 MHz	250 MHz (AMD), 400 MHz (Intel)
Input MFB configuration	1,4,8,8	1,8,8,8
Output MFB configuration	1,1,8,32	2,1,8,32

7 Resource Consumption

The following tables show the resource utilization on the AMD Virtex UltraScale+ chip (xcvu7p-flvb2104-2-i) for Gen3 x8 and Gen3 x16 PCIe configurations.

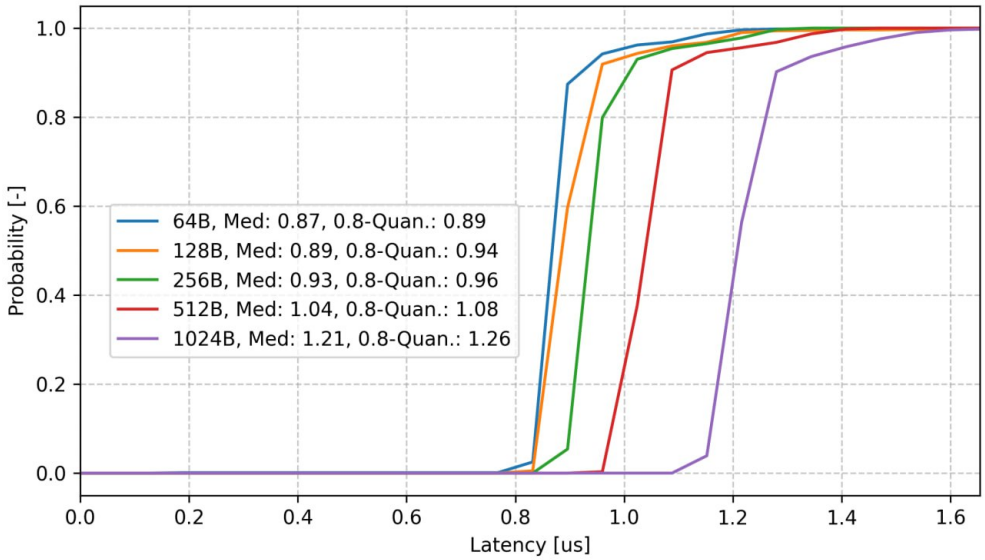
	PCIe Gen3 x8		PCIe Gen3 x16	
	16 channels	**64 channels**	**16 channels**	**64 channels**
LUT as Logic	7650 (0.97%)	10441 (1.32%)	16257 (2.06%)	18571 (2.36%)
LUT as Memory	1466 (0.37%)	2310 (0.59%)	1592 (0.40%)	2446 (0.62%)
Registers	10156 (0.64%)	11614 (0.74%)	16290 (1.03%)	17929 (1.14%)
CARRY logic	141 (0.14%)	238 (0.24%)	145 (0.15%)	243 (0.25%)
RAMB36 Tiles	32 (2.22%)	128 (8.89%)	0 (0.00%)	128 (8.89%)
RAMB18 Tiles	8 (0.28%)	8 (0.28%)	72 (2.50%)	8 (0.28%)
URAMs	8 (1.25%)	32 (5.00%)	8 (1.25%)	32 (5.00%)
DSPs	4 (0.09%)	4 (0.09%)	4 (0.09%)	4 (0.09%)

8 Latency report

Since this module has been designed for low latency, this is our primary concern. Even though its RTL design reaches the minimum latency, the PCI Express protocol remains the biggest contributor to the overall latency. From our observations, the latency is also influenced by the vendor of the CPU where Intel devices perform slightly better than AMD devices. Some PCIe IPs for FPGAs provide a special low-latency mode such as the PCIE4 block used in AMD UltraScale+ architecture, which is enabled on all AMD cards whose measurements we provide. The latency is always measured as a Round-Trip-Time (RTT) latency either on the path: Host -> H2F Controller -> FPGA -> F2H Controller -> Host (HFH), or FPGA -> F2H Controller -> Host -> H2F Controller -> FPGA (FHF), which will be denoted for specific results. Every time the data are looped back, either in the Host for the FHF path or in the FPGA for the HFH path, the loopback is established with the shortest path possible (E.g., for the HFH to directly connect USR_TX_MFB to the USR_RX_MFB interface).

8.1 AMD FPGA

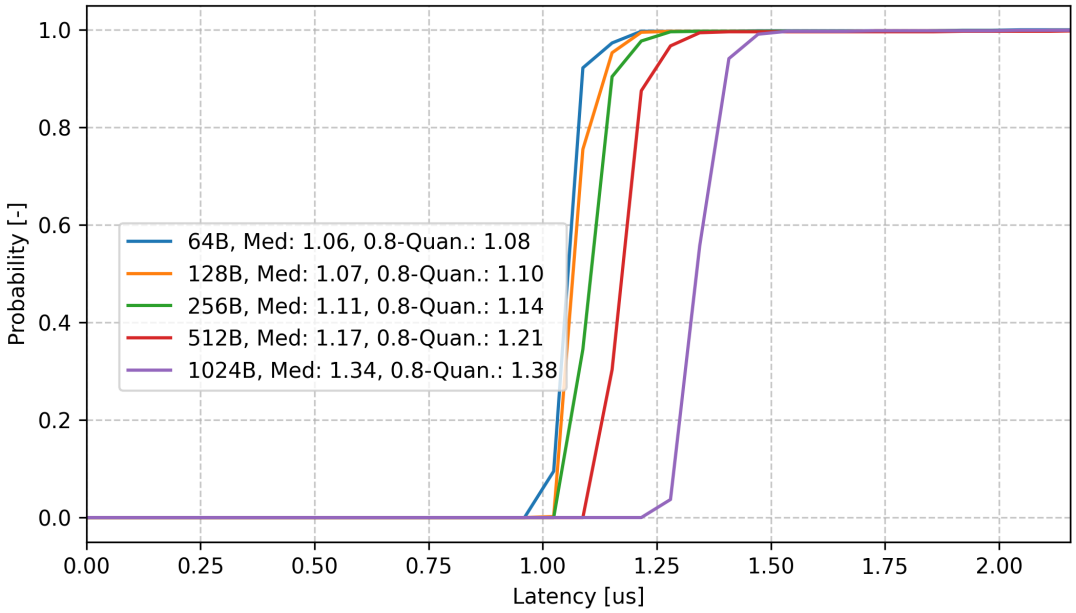
- **Card:** AMD Alveo X3522
- **CPU:** Intel(R) Xeon(R) E-2226G CPU @ 3.40GHz
- **RAM:** 64 GB (4 x 16GB)
- **PCIe configuration:** Gen3x8
- **HFH latency:** 64 byte packets, 1000000 repetitions
  - ~811 ns (median)
  - ~1.3 us (0.99-quantile)



FHF latency (1000 repetitions)

8.2 Intel FPGA

- **Card:** Silicom FPGA SmartNIC N6010
- **CPU:** Intel(R) Xeon(R) Gold 6348 CPU @ 2.60GHz
- **RAM:** 64 GB (4 x 16GB)
- **PCIe configuration:** Gen3x8
- **HFH latency:** 64 byte packets, 1000000 repetitions
  - ~1100 ns (median)
  - ~1.7 us (0.99-quantile)



FHF latency (1000 repetitions)

9 Revision History

Revision	Date	Description
1.0	April-14-2025	Initial release.