DYNANIC

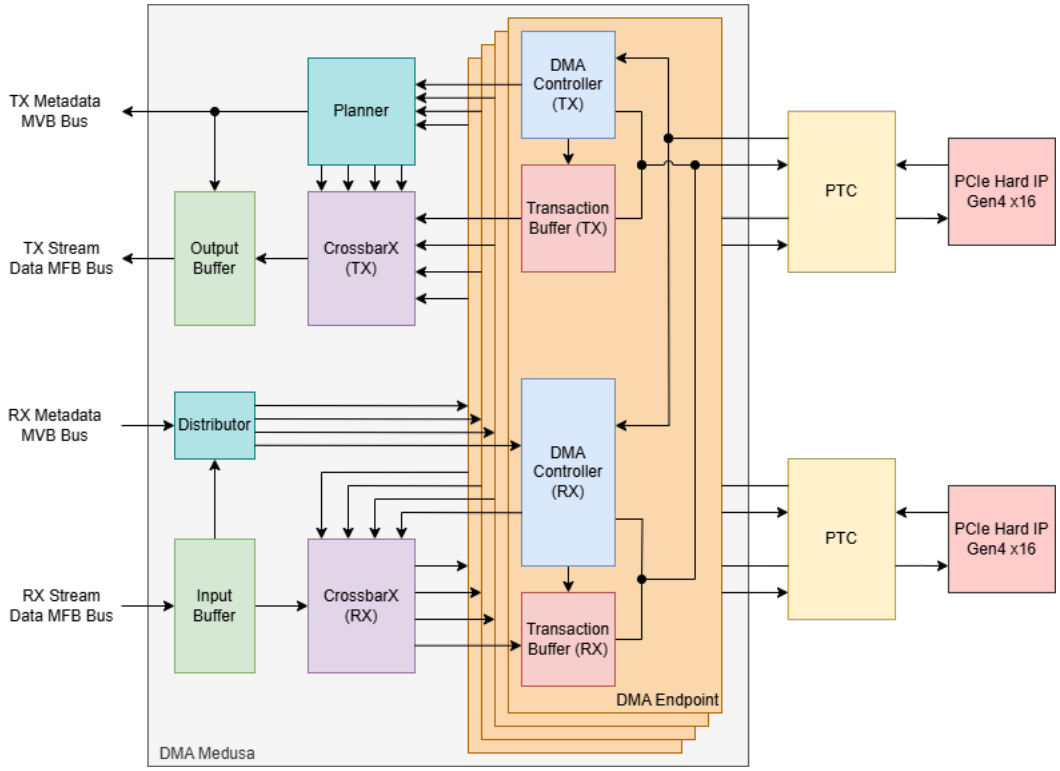
DMA Medusa IP Core Datasheet

1 Features

- **High-speed data transfer:** Supports transfer speeds of up to 400 Gbps
- **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

DMA Medusa is configurable the DMA controller, which is designed for high-speed transmission of large volumes of packet data through one or more PCIe interface to and from the host memory. DMA Medusa is FPGA vendor-independent and it is compatible with various PCIe IP cores.



Block Diagram of DMA Medusa

This design has (by default) a 2048b user data interface (typically for Ethernet) running at 200 MHz, meaning that up to 4 shortest Ethernet frames can be in a word. To achieve a transfer speed of 400 Gbps, on the other hand, different configurations

and number of PCIe interfaces can be used (1x PCIe Gen5 x16, 2x PCIe Gen4 x16, 4x PCIe Gen4 x8,...). It is not always necessary to have such a high transfer rate, therefore the DMA Medusa controller allows you to change the bus data width and PCIe interface configuration according to specific requirements. However, for further description, we will consider the default data width and PCIe interface in the 2x PCIe Gen4 x16 configuration.

Our solution consists of several main parts: input/output buffer, CrossbarX, DMA Endpoint and PTC block. We will explain the functioning of these parts in a simplified way using a specific circuit of the whole architecture for the direction of data transfer from the FPGA to the server (C2H - Card to Host). The packet arrives over the data bus and is stored in the input buffer. Along with the packet, the corresponding metadata arrives in the input buffer, which carries additional information such as the packet length and the number of the assigned DMA channel. This is because the DMA controller allows the packets being transmitted to be divided into a number of queues (DMA channels). Since DMA channels are statically mapped to DMA endpoints, the upper bits of the channel number also determine which DMA endpoint will be selected. A DMA Endpoint consists of two main parts, the DMA Controller and the Transaction Buffer. The DMA Endpoint can only process one data packet in each clock cycle. The former provides control and operation of the individual DMA channels and keeps track of the available memory in RAM. Packet metadata first arrives at the input of the DMA Controller. This information is used to determine how many PCIe transactions the packet will be divided into and what RAM address the packet will be written to. The instructions for CrossbarX are then generated. This block allows different sized chunks of data to be transferred between the Input Buffer and all Transaction Buffers (each in a single DMA Endpoint). The data in the Transaction Buffer is in the form of individual PCIe requests (as opposed to the Input Buffer, where it is individual packets). When packet data is successfully written as PCIe transactions to the Transaction Buffer, it is read and sent to the PTC block. This block allows two DMA Endpoints to be connected to a single PCIe Hard IP, while managing the allocation of PCIe tags to individual transactions.

In the case of the opposite direction of data transfer, from the server to the FPGA (H2C - Host to Card). The operation of the DMA controller is analogous. The DMA controller must first get information where the packets are stored in RAM and then start reading them. The read data is written to the Transaction Buffer, however, it can come in a random order. The DMA Controller again generates instructions, the Planner schedules these instructions from all DMA Endpoints appropriately and sends them to CrossbarX. It then starts transferring data from the Transaction Buffer to the Output Buffer so that the packets from all DMA Endpoints are ready in the Transaction Buffer.

3 Transmitted data

- At the lowest level, DMA transmits individual frames (these are typically network packets) and their associated frame metadata.
- Frame metadata mainly contain the length of the entire frame (16b) and user metadata (12b).
- In RX direction framework metadata is transmitted to the header space (RX DMA header), in TX direction it is transmitted directly in the descriptor.
- Users (incl. network device driver) typically do not have 12b for metadata. Therefore, there is a convention to use the User Header.
- The User Header is preceded by the packet and becomes part of the frame. Thus, the header length (with any padding) is added to the packet length.

User metadata bit vector	11	10	9	8	7	6	5	4	3	2	1	0
User Header	nHDR=0	nPKT	CRC	X	User Header length in bytes							
App specific metadata	nHDR=1	nPKT	CRC	X	U	U	U	U	U	U	U	U

Table-1. *Description of items in user metadata*

Item	Name	Description	Note
nHDR	User	Value 0: User header is present and its size is specified in metadata at bits 7:0; Value 1: meaning of bits 7:0 is application-specific	
nPKT	Ethernet packet	Value 0: the frame contains an Ethernet packet; Value 1: it is an application-specific frame (e.g. UH, etc.)	Introduced for direct processing e.g. by the network driver (after trimming any user header)
CRC	CRC of the packet	Specifies whether the Ethernet packet includes a CRC (only the proposal is not used now)	Valid only when nPKT = 0
U	User data	Application specific use	
X	Reserved	Reserved for future use	

Table-2. *RX DMA header*

Field	Bit index	Description
Length	15:0	Frame length in bytes
Metadata	27:16	User metadata vector
Descriptors	29:28	Number of released descriptors: 0 = zero descriptors, 1 = 1 descriptor, 2 = 2 descriptors, 3 = 3 or more descriptors
Reserved	31:30	Reserved for future use

4 Register map - one DMA channel

Offset	Register	Access	Description	Note
0x00	Control Register	(R/W)	Bit 0: start the controller	
0x04	Status Register	(R/O)	Bit 0: the controller is running	
0x08	Error Register	(R/O)	Reserved for future use	
0x0C	Feature Register	(R/W)	Reserved for future use	
0x10	Software descriptor pointer (SDP)	(R/W)	Specifies the number of descriptors ready for download (up to 32b)	
0x14	Software header pointer (SHP)	(R/W)	Specifies the number of packets released from the header buffer (up to 32b)	Not in TX
0x18	Hardware descriptor pointer (HDP)	(R/O)	Specifies the number of descriptors filled with packets (up to 32b)	
0x1C	Hardware header pointer (HHP)	(R/O)	Specifies the number of packets written in the header buffer (up to 32b)	Not in TX
0x20	Update timeout	(R/W)	Bit 27:0: Lowest number of clock cycles between two pointer updates to RAM	
0x24	Interrupt	(R/W)	TBD: Interrupt number, interrupt enable	

0x28	Virt Register	(R/W)	Bit 7:0: Virtual Function ID (VFID); Bit 27:8: PASID; Bit 31: Generate TLP header with PASID	
0x2C	Reserved	(N/A)	Reserved for future use	
0x30	Reserved	(N/A)	Reserved for future use	
0x34	Reserved	(N/A)	Reserved for future use	
0x38	Reserved	(N/A)	Reserved for future use	
0x3C	Reserved	(N/A)	Reserved for future use	
0x40	Descriptor baseL	(R/W)	Base RAM address for descriptors (lower 32b)	
0x44	Descriptor baseH	(R/W)	Base RAM address for descriptors (upper 32b)	
0x48	Header baseL	(R/W)	Base address of the RAM space for writing headers (lower 32b)	Not in TX
0x4C	Header baseH	(R/W)	Base address of the RAM space for writing headers (upper 32b)	Not in TX
0x50	Update baseL	(R/W)	Base address of RAM space for HDP+HHP update (lower 32b)	
0x54	Update baseH	(R/W)	Base address of RAM space for HDP+HHP update (upper 32b)	
0x58	Descriptor pointer mask (DPM)	(R/W)	Descriptor pointer size restriction mask (up to 32b)	
0x5C	Header pointer mask (HPM)	(R/W)	Header pointer size restriction mask (up to 32b)	Not in TX
0x60	Received packetsL	(R/R)	Number of received packets (lower 32b)	In TX as Sent bytes; Note: 1, 2, 3
0x64	Received packetsH	(R/R)	Number of received packets (upper 16b)	In TX as Sent bytes; Note: 1, 2, 3
0x68	Received bytesL	(R/R)	Number of received bytes (lower 32b)	In TX as Sent bytes; Note: 1, 2, 3
0x6C	Received bytesH	(R/R)	Number of received bytes (upper 16b)	In TX as Sent bytes; Note: 1, 2, 3
0x70	Discarded packetsL	(R/R)	Number of dropped packets (lower 32b)	Not in TX; Note: 1, 2, 3

0x74	Discarded packetsH	(R/R)	Number of dropped packets (upper 16b)	Not in TX; Note: 1, 2, 3
0x78	Discarded bytesL	(R/R)	Number of dropped bytes (lower 32b)	Not in TX; Note: 1, 2, 3
0x7C	Discarded bytesH	(R/R)	Number of dropped bytes (upper 16b)	Not in TX; Note: 1, 2, 3

Register map notes

- **Write 0 to one counter:** Resets all counters on that channel simultaneously (R/R = Read/Reset).
- **Write 1 to one counter:** Captures (strobes) all counter values on that channel simultaneously for subsequent individual reading.
- **Write 2 to one counter:** Simultaneously captures all counter values on that channel before resetting them all (R/R = Read/Reset).

5 Interface

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

5.1 Generic map description

Name	Type	Default Value	Description
DEVICE	string	N/A	Name of target device, the supported are: - "ULTRASCALE" - "STRATIX10" - "AGILEX"
USR_MVB_ITEMS	natural	1	Number of per-packet Items on USR MVB Interfaces (Maximum number of packets starting in one word)
USR_MFB_REGIONS	natural	1	USR MFB data bus configuration. Defines the total width of the User data stream. At rate 200MHz is configuration MFB#(1,8,8,8) able to transfer 100 Gbps. For the 400G variant, set this parameter to 4.
USR_MFB_REGION_SIZE	natural	8	Number of Blocks within the Region
USR_MFB_BLOCK_SIZE	natural	8	Number of Items in the Block
USR_MFB_ITEM_WIDTH	natural	8	Width of the Item [bits]
USR_RX_PKT_SIZE_MAX	natural	2^{12}	Maximum size of a RX User packet (in bytes). Defines width of Packet length signals.
USR_TX_PKT_SIZE_MAX	natural	2^{12}	Maximum size of a TX User packet (in bytes). Defines width of Packet length signals.
DMA_ENDPOINTS	natural	1	Number of used logical DMA (PCIe) Endpoints (each has its own MVB+MFB interface)
PCIE_MPS	natural	256	Maximum size of PCIe write request (in bytes)
PCIE_MRRS	natural	512	Maximum size of PCIe read request (in bytes)
DMA_TAG_WIDTH	natural	8	Width of PCIe Tag for one logical DMA Endpoint
PCIE_INTERRUPTS	natural	2048	Number of different PCIe Interrupt vectors
UP_MFB_REGIONS	natural	2	Upstream MFB interface. Must make up to 512b. (512 == REGIONS × REGION_SIZE ×
UP_MFB_REGION_SIZE	natural	1	
UP_MFB_BLOCK_SIZE	natural	8	

UP_MFB_ITEM_WIDTH	natural	32	BLOCK_SIZE × ITEM_WIDTH)
DOWN_MFB_REGIONS	natural	2	Downstream MFB interface. Must make up to 512b. (512 == REGIONS × REGION_SIZE × BLOCK_SIZE × ITEM_WIDTH)
DOWN_MFB_REGION_SIZE	natural	1	
DOWN_MFB_BLOCK_SIZE	natural	8	
DOWN_MFB_ITEM_WIDTH	natural	32	
HDR_META_WIDTH	natural	12	Width of User Header Metadata information
RX_CHANNELS	natural	8	Total number of RX DMA Channels. Minimum: 4
RX_DP_WIDTH	natural	16	Width of Software and Hardware Descriptor Pointer. Defines width of signals used for these values in DMA Module. Affects logic complexity. Maximum value: 32
RX_HP_WIDTH	natural	16	Width of Software and Hardware Header Pointer. Defines width of signals used for these values in DMA Module. Affects logic complexity. Maximum value: 32
RX_BLOCKING_MODE	boolean	FALSE	RX Blocking mode: if enabled, packets are not dropped due to lack of descriptors but must wait for new descriptors to be downloaded.
TX_CHANNELS	natural	8	Total number of TX DMA Channels Minimum value: TX_SEL_CHANNELS × DMA_ENDPOINTS
TX_SEL_CHANNELS	natural	8	Maximum number of active TX DMA Channels in one dma Endpoint Affects logic complexity and memory usage. Minimum: 4
TX_DP_WIDTH	natural	16	Width of Software and Hardware Descriptor Pointer. Defines width of signals used for these values in DMA Module. Affects logic complexity. Maximum value: 32
TX_USR_MFB_GAP_SIZE	natural	12+4+8	TX_USR_MFB Inter-frame Gap Size (in number of USR MFB Items)
DSP_CNT_WIDTH	natural	48	Debug generic: Width of DSP packet and byte statistics counters.
RX_GEN_EN	boolean	TRUE	Debug generic: Enable generation of RX/TX side of DMA Module
TX_GEN_EN	boolean	TRUE	Debug generic: Enable generation of RX/TX side of DMA Module
USR_EQ_DMA	boolean	FALSE	Debug generic: USR_CLK is the same as DMA_CLK

CROX_EQ_DMA	boolean	FALSE	Debug generic: CROX_CLK is the same as DMA_CLK
CROX_DOUBLE_DMA	boolean	TRUE	Debug generic: CROX_CLK is exactly double frequency of DMA_CLK from the same source (CROX_RESET is the same as DMA_RESET)
SPEED_METER_EN	boolean	TRUE	Debug generic: Enable MI-accessible MFB Speed meters in each DMA Endpoint
DBG_CNTR_EN	boolean	FALSE	Debug generic: Enable additional MI-accessible debug counters
MI_WIDTH	natural	32	Do not change! Width of MI bus
MI_META_WIDTH	natural	2	Do not change!

5.2 Port map description

Name	Dir	Type	Description
DMA_CLK	in	std_logic	Main Clock for the DMA Module
DMA_RESET	in	std_logic	
CROX_CLK	in	std_logic	Clock and Reset for CrossbarX data transfers
CROX_RESET	in	std_logic	
USR_CLK	in	std_logic	Clock and Reset for the User-side interface
USR_RESET	in	std_logic	
RX_USR_MVB_LEN	in	std_logic_vector (USR_MVB_ITEMS $\times$ $\log_2$ (USR_RX_PKT_SIZE_MAX $+ 1) -$ 1 downto 0)	Rx User MVB Data
RX_USR_MVB_HDR_META	in	std_logic_vector (USR_MVB_ITEMS$\times$ HDR_META_WIDTH $- 1$ downto 0)	
RX_USR_MVB_CHANNEL	in	std_logic_vector (USR_MVB_ITEMS $\times$ $\log_2$ (RX_CHANNELS) $- 1$ downto 0)	
RX_USR_MVB_DISCARD	in	std_logic_vector (USR_MVB_ITEMS $- 1$ downto 0)	
RX_USR_MVB_VLD	in	std_logic_vector (USR_MVB_ITEMS $- 1$ downto 0)	Rx User MVB Protocol signals
RX_USR_MVB_SRC_RDY	in	std_logic	
RX_USR_MVB_DST_RDY	out	std_logic	

RX_USR_MFB_DATA	in	std_logic_vector (USR_MFB_REGIONS× USR_MFB_REGION_SIZE× USR_MFB_BLOCK_SIZE× USR_MFB_ITEM_WIDTH− 1 downto 0)	Rx User MFB Bus
RX_USR_MFB_SOF	in	std_logic_vector (USR_MFB_REGIONS− 1 downto 0)	
RX_USR_MFB_EOF	in	std_logic_vector (USR_MFB_REGIONS− 1 downto 0)	
RX_USR_MFB_SOF_POS	in	std_logic_vector (USR_MFB_REGIONS × max(1, $\log_2$(USR_MFB_REGION_SIZE))− 1 downto 0)	
RX_USR_MFB_EOF_POS	in	std_logic_vector (USR_MFB_REGIONS × max(1, $\log_2$(USR_MFB_REGION_SIZE× USR_MFB_BLOCK_SIZE))− 1 downto 0)	
RX_USR_MFB_SRC_RDY	in	std_logic	
RX_USR_MFB_DST_RDY	out	std_logic	
TX_USR_MVB_LEN	out	std_logic_vector (USR_MVB_ITEMS× $\log_2$(USR_TX_PKT_SIZE_MAX + 1) −1 downto 0)	Tx User MVB Data
TX_USR_MVB_HDR_META	out	std_logic_vector (USR_MVB_ITEMS× HDR_META_WIDTH− 1 downto 0)	
TX_USR_MVB_CHANNEL	out	std_logic_vector (USR_MVB_ITEMS× $\log_2$(TX_CHANNELS)− 1 downto 0)	
TX_USR_MVB_VLD	out	std_logic_vector (USR_MVB_ITEMS − 1 downto 0)	Tx User MVB Protocol signals
TX_USR_MVB_SRC_RDY	out	std_logic	
TX_USR_MVB_DST_RDY	in	std_logic	
TX_USR_MFB_DATA	out	std_logic_vector (USR_MFB_REGIONS× USR_MFB_REGION_SIZE× USR_MFB_BLOCK_SIZE× USR_MFB_ITEM_WIDTH− 1 downto 0)	Tx User MFB Bus
TX_USR_MFB_SOF	out	std_logic_vector (USR_MFB_REGIONS− 1 downto 0)	

TX_USR_MFB_EOF	out	std_logic_vector (USR_MFB_REGIONS− 1 downto 0)	
TX_USR_MFB_SOF_POS	out	std_logic_vector (USR_MFB_REGIONS × max(1, log$_2$(USR_MFB_REGION_SIZE))− 1 downto 0)	
TX_USR_MFB_EOF_POS	out	std_logic_vector (USR_MFB_REGIONS × max(1, log$_2$(USR_MFB_REGION_SIZE× USR_MFB_BLOCK_SIZE))− 1 downto 0)	
TX_USR_MFB_SRC_RDY	out	std_logic	
TX_USR_MFB_DST_RDY	in	std_logic	
TX_USR_CHOKE_CHANS	in	std_logic_vector (TX_CHANNELS − 1 downto 0)	Selectively pause (choke) a DMA channel(s). Can be used to avoid stopping the whole DMA Module when just a single channel is slacking behind.
UP_MVB_DATA	out	slv_array_t (DMA_ENDPOINTS − 1 downto 0) (UP_MFB_REGIONS× DMA_UPHDR_WIDTH− 1 downto 0)	PCIe-side interface. Upstream MVB interface (for sending request headers to PCIe Endpoints)
UP_MVB_VLD	out	slv_array_t (DMA_ENDPOINTS − 1 downto 0) (UP_MFB_REGIONS− 1 downto 0)	
UP_MVB_SRC_RDY	out	std_logic_vector (DMA_ENDPOINTS − 1 downto 0)	
UP_MVB_DST_RDY	in	std_logic_vector (DMA_ENDPOINTS − 1 downto 0)	
UP_MFB_DATA	out	slv_array_t(DMA_ENDPOINTS− 1 downto 0)(UP_MFB_REGIONS× UP_MFB_REGION_SIZE× UP_MFB_BLOCK_SIZE× UP_MFB_ITEM_WIDTH− 1 downto 0)	PCIe-side interface. Upstream MFB interface (for sending data to PCIe Endpoints)
UP_MFB_SOF	out	slv_array_t(DMA_ENDPOINTS− 1 downto 0)(UP_MFB_REGIONS− 1 downto 0)	
UP_MFB_EOF	out	slv_array_t(DMA_ENDPOINTS− 1 downto 0)(UP_MFB_REGIONS− 1 downto 0)	

UP_MFB_SOF_POS	out	slv_array_t(DMA_ENDPOINTS−1 downto 0)(UP_MFB_REGIONS×max(1, log$_2$(UP_MFB_REGION_SIZE))−1 downto 0)	
UP_MFB_EOF_POS	out	slv_array_t(DMA_ENDPOINTS−1 downto 0)(UP_MFB_REGIONS×max(1, log$_2$(UP_MFB_REGION_SIZE×UP_MFB_BLOCK_SIZE))−1 downto 0)	
UP_MFB_SRC_RDY	out	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
UP_MFB_DST_RDY	in	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
DOWN_MVB_DATA	in	slv_array_t(DMA_ENDPOINTS−1 downto 0)(DOWN_MFB_REGIONS×DMA_DOWNHDR_WIDTH−1 downto 0)	PCIe-side interface. Downstream MVB interface (for receiving completion headers from PCIe Endpoints)
DOWN_MVB_VLD	in	slv_array_t(DMA_ENDPOINTS−1 downto 0)(DOWN_MFB_REGIONS−1 downto 0)	
DOWN_MVB_SRC_RDY	in	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
DOWN_MVB_DST_RDY	out	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
DOWN_MFB_DATA	in	slv_array_t(DMA_ENDPOINTS−1 downto 0)(DOWN_MFB_REGIONS×DOWN_MFB_REGION_SIZE×DOWN_MFB_BLOCK_SIZE×DOWN_MFB_ITEM_WIDTH−1 downto 0)	PCIe-side interface. Downstream MFB interface (for sending data from PCIe Endpoints)
DOWN_MFB_SOF	in	slv_array_t(DMA_ENDPOINTS−1 downto 0)(DOWN_MFB_REGIONS−1 downto 0)	
DOWN_MFB_EOF	in	slv_array_t(DMA_ENDPOINTS−1 downto 0)(DOWN_MFB_REGIONS−1 downto 0)	
DOWN_MFB_SOF_POS	in	slv_array_t(DMA_ENDPOINTS−1 downto 0)(DOWN_MFB_REGIONS × max(1, log$_2$(DOWN_MFB_REGION_SIZE))−1 downto 0)	

DOWN_MFB_EOF_POS	in	slv_array_t(DMA_ENDPOINTS−1 downto 0)(DOWN_MFB_REGIONS × max(1, log$_2$(DOWN_MFB_REGION_SIZE ×DOWN_MFB_BLOCK_SIZE))−1 downto 0)	
DOWN_MFB_SRC_RDY	in	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
DOWN_MFB_DST_RDY	out	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
PCIE_INTER_REQ	out	std_logic_vector((RX_CHANNELS+ TX_CHANNELS) − 1 downto 0)	PCIe interrupt requests
PCIE_INTER_VEC	out	slv_array_t((RX_CHANNELS+ TX_CHANNELS) − 1 downto 0)(log$_2$(PCIE_INTERRUPTS)− 1 downto 0)	
MI_ADDR	in	slv_array_t(DMA_ENDPOINTS− 1 downto 0)(MI_WIDTH− 1 downto 0)	MI interface for SW access (see register space part)
MI_MWR	in	slv_array_t(DMA_ENDPOINTS− 1 downto 0)(MI_META_WIDTH− 1 downto 0) := (others => '0')	
MI_DWR	in	slv_array_t(DMA_ENDPOINTS− 1 downto 0)(MI_WIDTH− 1 downto 0)	
MI_BE	in	slv_array_t(DMA_ENDPOINTS− 1 downto 0)(MI_WIDTH/8− 1 downto 0)	
MI_RD	in	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
MI_WR	in	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
MI_DRD	out	slv_array_t(DMA_ENDPOINTS − 1 downto 0)(MI_WIDTH − 1 downto 0)	
MI_ARDY	out	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	
MI_DRDY	out	std_logic_vector(DMA_ENDPOINTS − 1 downto 0)	

Bus Specification

The bus specification is available in the NDK-FPGA documentation.
MI Bus Specification
MVB Bus Specification
MFB Bus Specification

Packages

The DMA interface uses its own signal types and constants. Its definition can be found in the NDK-FPGA repository.
Note that VHDL 2008 standard is used and required.

6 Clocks and Resets

For the 100Gbps or 400Gbps version of the DMA Medusa, it is necessary to use 200MHz for the DMA_CLK. The CROX_CLK must be twice the DMA_CLK. The user clock can range from 100MHz to 400MHz. The reset should last at least 128 cycles of the given clock. For example, the 200MHz clock requires a reset of at least 640 ns.

7 Resource Utilization

This section summarizes the utilization of given FPGA families. It should be noted that the final logic resource depends on the unrelated logic.

7.1 400G Utilization

Table-6. *Utilization report for Altera Devices - 400G DMA*

Family	Part Number	ALUTs	Registers	Pins	Block Memory bits	Design Tool
AGILEX I	AGIB023R18A1E1V	281 801	340 448	0	21 915 992	Quartus 24.1
AGILEX I	AGIB027R29A1E2VR3	286 486	342 228	0	21 967 192	Quartus 24.1

7.2 200G Utilization

Table-7. *Utilization report for Altera Devices - 200G DMA*

Family	Part Number	ALUTs	Registers	Pins	Block Memory bits	Design Tool
AGILEX F	AGFB014R24A2E2V	105 542	167 996	0	10 419 372	Quartus 24.1
AGILEX F	AGFB014R24B2E2V	105 536	168 007	0	10 419 372	Quartus 24.1
STRATIX10	1SD280PT2F55E1VG	107 931	187 694	0	10 419 372	Quartus 24.1

Table-8. *Utilization report for AMD Devices - 200G DMA*

Family	Part Number	CLB LUTs	CLB Registers	Block RAM Tile	Design Tool
Virtex UltraScale+	xcu55c-fsvh2892-2L-e	109 117	149 322	349	Vivado 2022.2

7.3 100G Utilization

Table-9. *Utilization report for Altera Devices - 100G DMA*

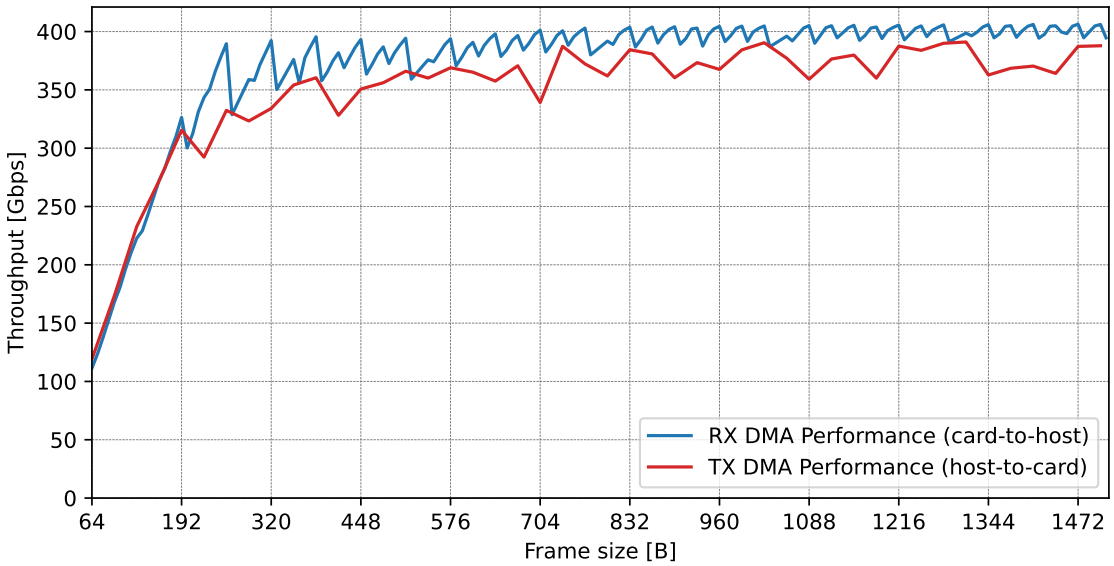
Family	Part Number	ALUTs	Registers	Pins	Block Memory bits	Design Tool
STRATIX10	1SM21BEU2F55E2VG	56 617	96 323	0	5 627 030	Quartus 24.1

Table-10. *Utilization report for AMD Devices - 100G DMA*

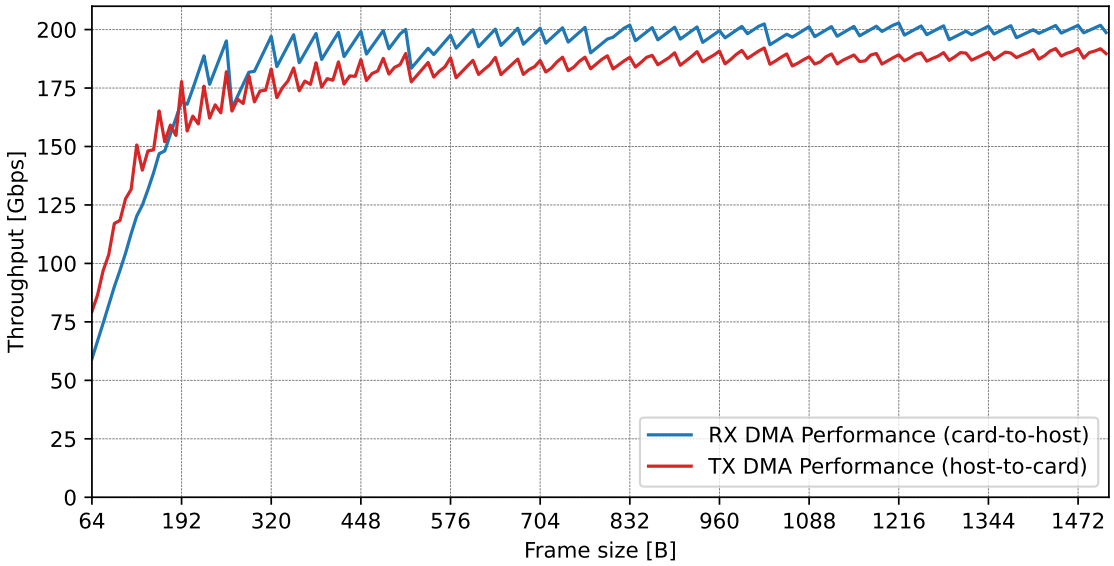
Family	Part Number	CLB LUTs	CLB Registers	Block RAM Tile	Design Tool
Zynq UltraScale+	xczu19eg-ffvc1760-2-i	56 060	75 368	186.5	Vivado 2022.2
Virtex UltraScale+	xcu55c-fsvh2892-2L-e	55 993	74 994	186.5	Vivado 2022.2
Kintex UltraScale+	xcku15p-ffve1760-2-e	56 111	76 164	186.5	Vivado 2022.2

8 Performance measurement

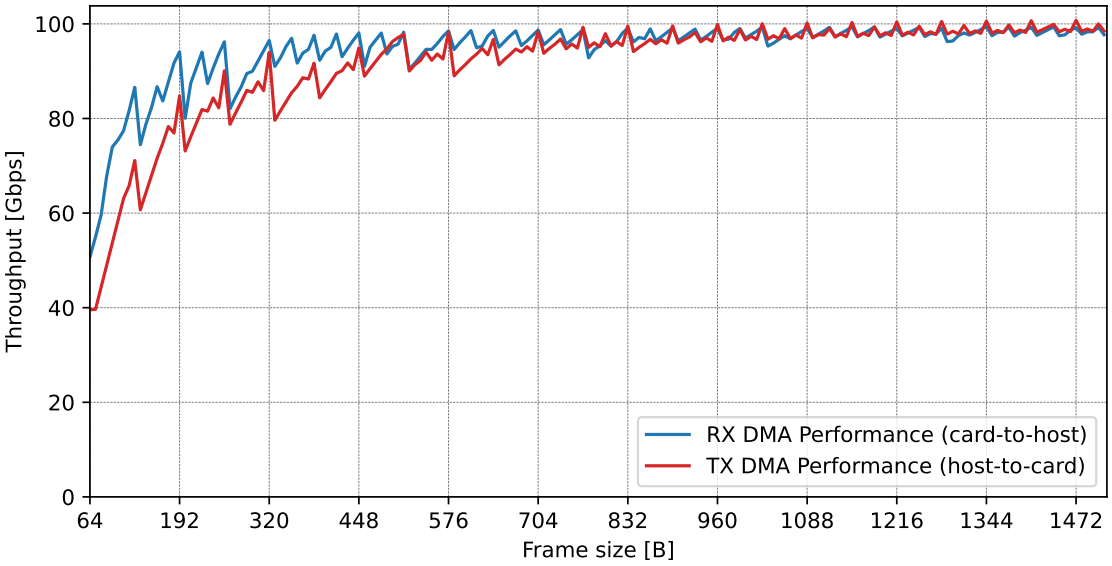
This section benchmarks the performance of the 100G, 200G, and 400G DMA solutions. Keep in mind that the final results are influenced by the host machine's capabilities and its specific PCIe generation.



400G DMA throughput measurement - ReflexCES XpressSX AGI-FH400G
PCIe 2xgen4x8x8, RAM 64GB (8x8GB), Intel(R) Xeon(R) Gold 6348 CPU@2.60GHz



200G DMA throughput measurement - Bittware IA-420F
PCIe 1xgen4x16, RAM 384GB (12x32GB), AMD EPYC 9554P 64-Core Processor



100G DMA throughput measurement - Silicom fb2CGhh@KU15P
PCIe 1xgen3x16, RAM 384GB (24x16GB), Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz

9 Revision History

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