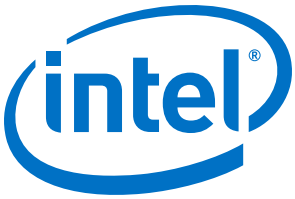


Multi Channel DMA for PCI Express IP User Guide

Updated for Intel® Quartus® Prime Design Suite: **20.2**

IP Version: **20.0.0**

Subscribe
Send Feedback

UG-20297 | 2020.07.20
Latest document on the web: **PDF** | **HTML**

Contents

1. Terms and Acronyms 4

2. Introduction 5

2.1. Multi Channel DMA for PCI Express IP Features 6
2.2. Device Family Support 6
2.3. Recommended Speed Grades 7
2.4. Performance and Resource Utilization 7
2.5. Release Information 7

3. Functional Description 8

3.1. Multi Channel DMA 8
3.1.1. H2D Data Mover 8
3.1.2. D2H Data Mover 9
3.1.3. Descriptors 9
3.2. User Logic Interface 13
3.2.1. Avalon-MM PIO Master 14
3.2.2. Avalon-ST Source (H2D) and Sink (D2H) 14
3.2.3. Avalon-MM Write (H2D) and Read (D2H) Master 15
3.3. Control Registers 16

4. Registers 17

4.1. Queue Control (QCSR) 18
4.2. MSI-X Memory Space 22
4.3. Control Register (GCSR) 23

5. Interface Overview 25

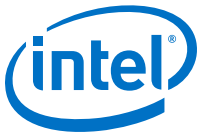
5.1. Port List 26
5.2. Clocks 27
5.3. Resets 27
5.4. Avalon-MM PIO Master 27
5.5. Avalon-MM Write Master (H2D) 28
5.6. Avalon-MM Read Master (D2H) 28
5.7. Avalon-ST Source (H2D) 29
5.8. Avalon-ST Sink (D2H) 29

6. Parameters 31

6.1. IP Settings 31
6.1.1. System Settings 31
6.1.2. MCDMA Settings 32
6.1.3. Device Identification Registers 32
6.1.4. PCI Express / PCI Capabilities Parameters 33
6.1.5. Configuration, Debug and Extension Options 37
6.1.6. PHY Characteristics 37
6.2. Example Designs 37

7. Designing with the IP Core 39

7.1. Generating the IP Core 39
7.2. Simulating the IP Core 40
7.3. IP Core Generation Output - Intel Quartus Prime Pro Edition 41



7.4. Systems Integration and Implementation....44
7.4.1. Clock Requirements....44
7.4.2. Reset Requirements....44
7.4.3. Required Supporting IP....45

8. Software Programming Model....46

8.1. Architecture....47
8.2. Software Flow....48
8.3. API Flow....50
8.3.1. Single Descriptor Load and Submit....50
8.3.2. Multiple Descriptor Load and Submit....50
8.3.3. `libmqdma` Library API List....51

9. Revision History....57

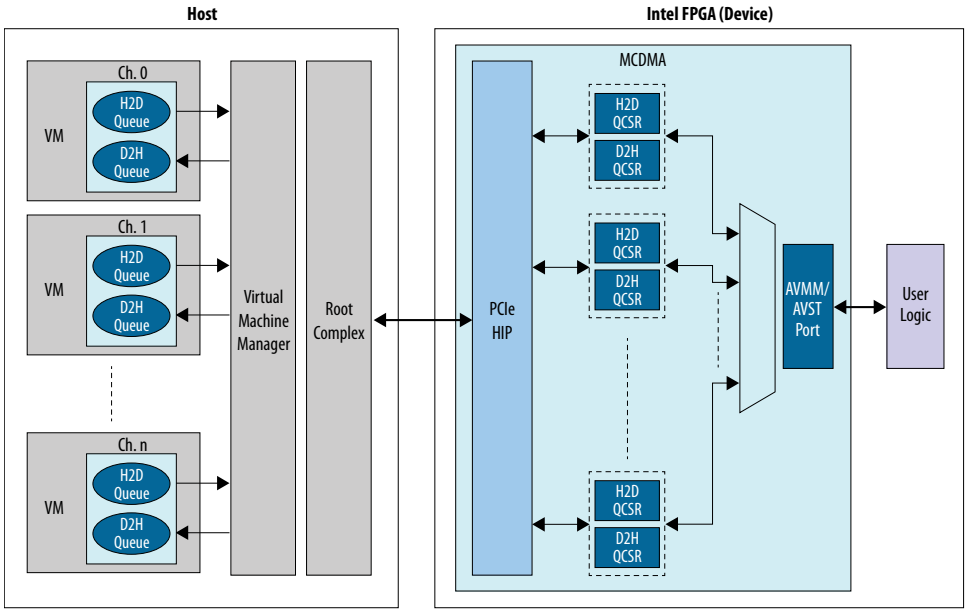
1. Terms and Acronyms

Table 1. Acronyms

Term	Definition
PCIe*	Peripheral Component Interconnect Express (PCI Express*)
DMA	Direct Memory Access
MCDMA	Multi Channel Direct Memory Access
PIO	Programmed Input/Output
H2D	Host-to-Device
D2H	Device-to-Host
H2DDM	Host-to-Device Data Mover
D2HDM	Device-to-Host Data Mover
QCSR	Queue Control and Status register
GCSR	General Control and Status Register
IP	Intellectual Property
HIP	Hard IP
PD	Packet Descriptor
QID	Queue Identification
TIDX	Queue Tail Index (pointer)
HIDX	Queue Head Index (pointer)
TLP	Transaction Layer Packet
IMMWR	Immediate Write Operation
MRRS	Maximum Read Request Size
CvP	Configuration via Protocol
PBA	Pending Bit Array
API	Application Programming Interface
Avalon®-MM (or AVMM)	Avalon Memory-Mapped Interface
Avalon-ST (or AVST)	Avalon Streaming Interface
SOF	Start of a File (or packet) for streaming
EOF	End of a File (or packet) for streaming
File (or Packet)	A group of descriptors defined by SOF and EOF bits of the descriptor for the streaming. At Avalon-ST user interface, a file (or packet) is marked by means of sof/eof.

2. Introduction

Figure 1. Multi Channel DMA for PCI Express IP Usage in Server Hardware Infrastructure



The Multi Channel DMA for PCI Express IP enables you to efficiently transfer data between the host and device. The Multi Channel DMA for PCI Express IP supports multiple DMA channels between the host and device over the underlying PCIe link. A DMA channel consists of H2D (host to device) and D2H (device to host) queue pair.

As shown in the figure above, the Multi Channel DMA for PCI Express IP can be used in a server's hardware infrastructure to allow communication between various VM-clients and their FPGA-device based counterparts. The Multi Channel DMA for PCI Express Ioperates on descriptor-based queues set up by driver software to transfer data between local FPGA and host. Multi Channel DMA for PCI Express IP's control logic reads the queue descriptors and executes them. Separate queues are used for D2H and H2D operations for each channel.

The Multi Channel DMA for PCI Express IP integrates the Intel® PCIe Hard IP and interfaces with the host Root Complex via the PCIe serial lanes. On the user logic interface, Avalon-MM/Avalon-ST interfaces allow the designer for easy integration of the Multi Channel DMA for PCI Express IP with other Platform Designer components.

2.1. Multi Channel DMA for PCI Express IP Features

- Supports Intel Stratix® 10 H-Tile PCIe Gen3 x16
- DMA user interface bandwidth 512 bit @ 250 MHz
- Allows user to select Avalon-MM or Avalon-ST DMA interface
- Avalon-ST DMA interface supports 4 ports and one DMA channel per port
- Avalon-MM DMA interface supports 1 port and up to 8 DMA channels
- Integrated MSI-X
- Support for Max Payload Size value of 512 bytes
- Support for Completion Re-ordering
- Support for preventing Head-of-Line Blocking
- Avalon-ST Interface uses big-endian.

Note: MSI is not supported. Support for user MSI-X generation may be supported in future release.

2.2. Device Family Support

The following terms define Multi Channel DMA for PCI Express IP core support levels for Intel FPGA IP cores in Intel Stratix 10 devices:

- **Advanced support**: the IP core is available for simulation and compilation for this device family. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP core for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).
- **Preliminary support**: the IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.
- **Final support**: the IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 2. Device Family Support Table

Device Family	Support Level
Intel Stratix 10	Preliminary
Other device families	No support

Related Information

Timing and Power Models
Reports the default device support levels in the current version of the Intel Quartus Prime Pro Edition software.

2.3. Recommended Speed Grades

Table 3. Recommended Speed Grades

Lane Rate	Link Width	Interface Width (bits)	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen3	x16	512	250	-1,-2

Related Information

Quartus Standard to Timing Closure and Optimization
Use this link for the Quartus Prime Pro Edition Software.

2.4. Performance and Resource Utilization

Table 4. Multi Channel DMA for PCI Express IP resource utilization based on default parameter setting

IP Mode	User Interface	# PF, # Port, # Ch./ Port	ALMs needed	Logic Registers	M20Ks
Gen3 x16 512 bit	Avalon-ST	1,4,1	80,718	204,461	781
Gen3 x16 512 bit	Avalon-MM	1,1,4	48,519	115,158	556

2.5. Release Information

IP versions are the same as the Intel Quartus® Prime Design Suite software versions up to v19.1. From Intel Quartus Prime Design Suite software version 19.2 or later, IP cores have a new IP versioning scheme. If an IP core version is not listed, the user guide for the previous IP core version applies. The IP versioning scheme (X.Y.Z) number changes from one software version to another.

A change in:

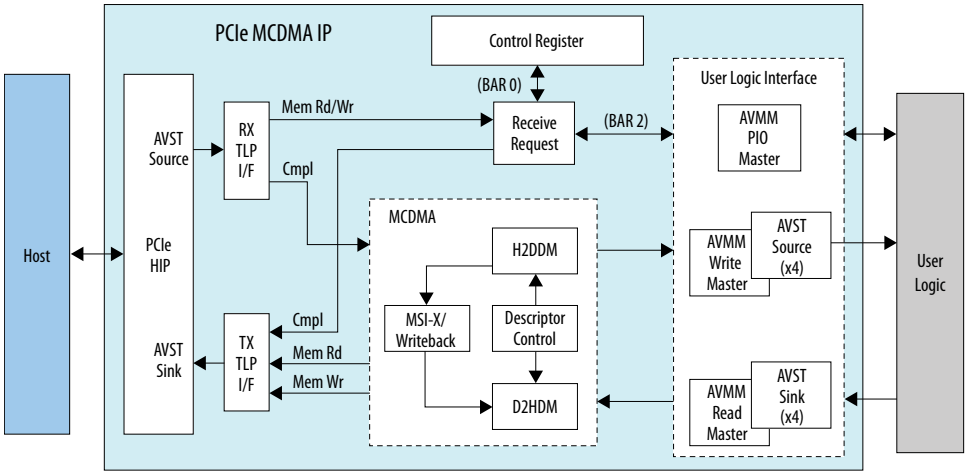
- X indicates a major revision of the IP. If you update your Intel Quartus Prime software, you must regenerate the IP.
- Y indicates the IP includes new features. Regenerate your IP to include these new features.
- Z indicates the IP includes minor changes. Regenerate your IP to include these changes.

Table 5. Release information for the Multi Channel DMA for PCI Express IP Core

Item	Description
IP Version	20.0.0
Intel Quartus Prime Version	Intel Quartus Prime Pro Edition 20.2 Software Release
Release Date	2020.07.20
Ordering Part Number (OPN)	IP-PCIEMCDMA
Product ID (PID)	017C

3. Functional Description

Figure 2. Multi Channel DMA for PCI Express IP Block Diagram



3.1. Multi Channel DMA

Multi Channel DMA for PCI Express IP consists primarily of H2DDM & D2HDM blocks. It also offers a DMA-bypass capability to the Host for doing PIO Read/Writes to device memory.

3.1.1. H2D Data Mover

The Host-to-Device Data Mover (H2DDM) module transfers data from the host memory to local memory through the PCIe Hard IP and the Avalon-ST Source interface.

There are two modes of usage for the H2DDM: queue descriptors fetching and H2D data payload transfer.

When used for descriptor fetching, the destination of the completion data is internal descriptor FIFOs where descriptors are stored before being dispatched to the H2DDM or D2HDM for actual data transfer.

When used for data payload transfer, it generates Mem Rd TLPs based on descriptor information such as PCIe address (source), data size, and MRRS value and forwards the received data to the user logic through the Avalon-MM Write Master / Avalon-ST Source interface. The received completions are re-ordered to ensure the read data is delivered to user logic in order.

When a descriptor is completed, that is, all read data has been received and forwarded to the Avalon-MM Write Master / Avalon-ST Source interface, The H2DDM performs the housekeeping tasks that include:

- Schedule MSI-X for a completed queue, if enabled
- Schedule Writeback Consumed Head Pointer for a completed queue, if enabled
- Update Consume Head Pointer for software polling

Based on the updated status, software can proceed with releasing the transmit buffer and reuse the descriptor ring entries.

3.1.2. D2H Data Mover

The D2H Data Mover (D2HDM) transfers data from device memory to host memory. It receives the data from the user logic through the Avalon-MM Read Master / Avalon-ST Sink interface and generates Mem Wr TLPs to move the data to the host based on descriptor information such as PCIe address (destination), data size, and MPS value.

When a descriptor is completed, that is, all DMA data has been sent to the host, the D2HDM performs housekeeping tasks that include:

- Schedule MSI-X for a completed queue, if enabled
- Schedule Writeback Consumed Head Pointer for a completed queue, if enabled
- Update Consume Head Pointer for software polling

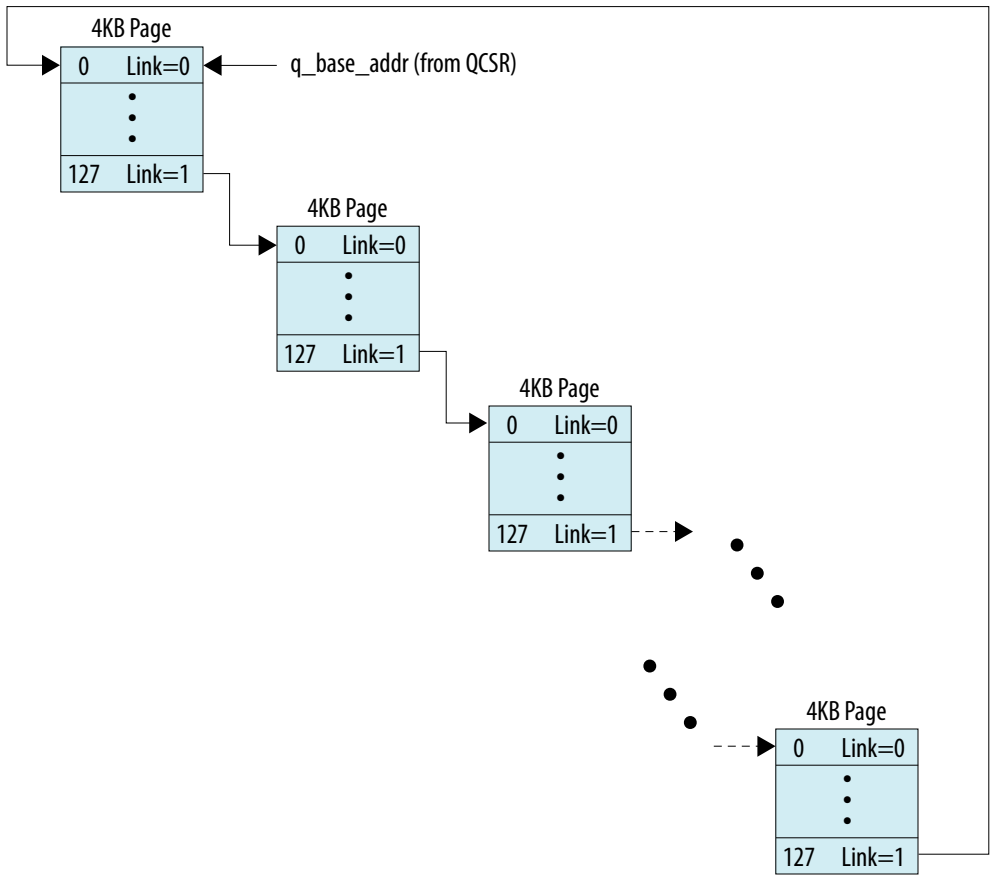
Based on the updated status, software can proceed with releasing the receive buffer and reuse the descriptor ring entries.

3.1.3. Descriptors

A DMA channel to support Multi Channel DMA data movement consists of a pair of the descriptor queues: one H2D descriptor queue and one D2H descriptor queue. Descriptors are arranged contiguously within a 4 KB page.

Each descriptor is 32 bytes in size. The descriptors are kept in host memory in a linked-list of 4 KB pages. For a 32 byte descriptor and a 4 KB page, each page contains upto 128 descriptors. The last descriptor in a 4 KB page must be a “link descriptor” – a descriptor containing a link to the next 4 KB page with the link bit set to 1. The last entry in the linked list must be a link pointing to the base address programmed in the QCSR, in order to achieve a circular buffer containing a linked-list of 4 KB pages. The figure below shows the descriptor linked list.

Figure 3. Descriptor Linked-List



Software and hardware communicate and manage the descriptors using tail index pointer (Q_TAIL_POINTER) and head index pointer (Q_HEAD_POINTER) QCSR registers as shown in the following figure. The DMA starts when software writes the last valid descriptor index to the Q_TAIL_POINTER register.

Figure 4. Descriptor Ring Buffer

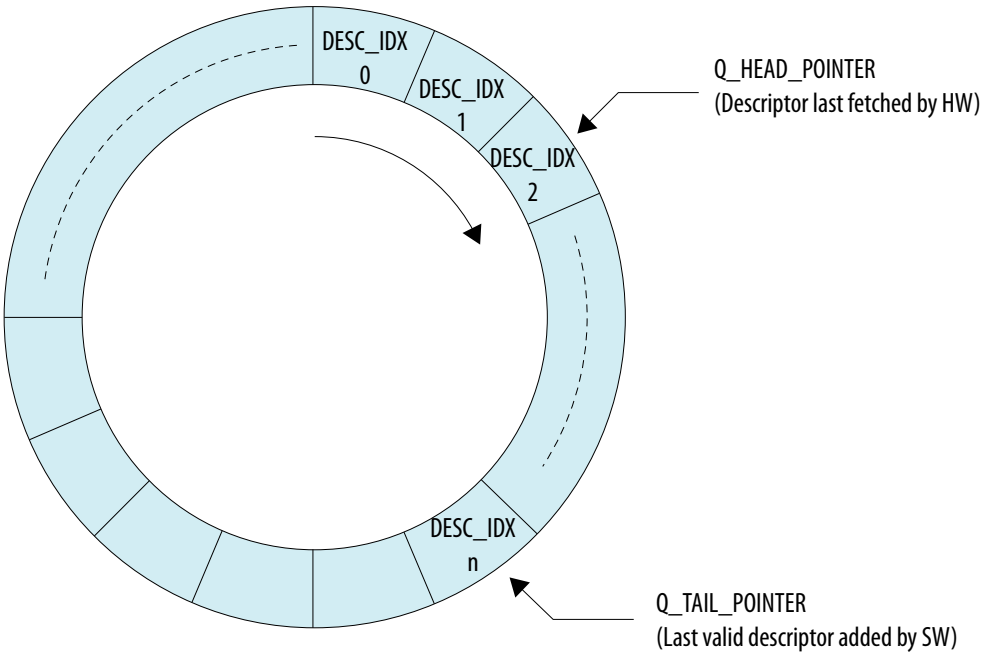


Table 6. Software Descriptor Format

Name	Width	Description
SRC_ADDR [63:0]	64	If Link bit =0, then this field contains the source address. Starting system address of allocated transmit buffer read by DMA that can be any byte alignment. If the queue is H2D, then this field contains the address in Host Memory. If the queue is D2H, then this is the AVMM address in device memory. If the link bit is set, then this contains the address of the next 4 KB page in host memory containing the descriptors.
DEST_ADDR [127:64]	64	Provided link=0, this field means: Starting local AVMM address written by DMA that can be any byte alignement. If the queue is D2H, then this field contains the address in Host Memory. If the queue is H2D, then this is the AVMM address in device memory.
PYLD_CNT [147:128]	20	Provided link=0, this field means: DMA payload size in bytes. Max 1 MB, with 20'h0 indicating 1 MB.
RSRVD [159:148]	12	Reserved

continued...

Name	Width	Description
DESC_IDX [175:160]	16	Unique Identifier for each descriptor, assigned by the software driver. This value is written to Q_COMPLETED_POINTER register when a descriptor data transfer is complete.
MSIX_EN [176]	1	Enable MSI-X per descriptor
WB_EN [177]	1	Enable Write Back per descriptor
RSRVD [191:178]	14	Reserved
RX_PYLD_CNT [211:192]	20	Received actual payload for D2H data movement (upstream)
RSRVD [221:212]	10	Reserved
SOF [222]	1	SOF indicator for Avalon-ST streaming. In the H2D streaming, this bit causes the Avalon-ST Source interface to assert h2d_st_sof_o, indicating start of a file/packet. In the D2H streaming, this bit is set in the Descriptor itself, by a Writeback (if Writeback is enabled) when the user logic asserts d2h_st_sof_i, indicating start of a file/packet. *Note:* In the H2D streaming, both SOF and EOF can be set in the same descriptor (file size = payload count) or it can span multiple descriptor pages. *Note:* In the D2H streaming, if user logic prematurely ends the data transfer by asserting d2h_st_eof_i in the middle of a descriptor data move then and starts a next file/packet, the SOF bit in the next descriptor is set by a Writeback. *Note:* SOF bit is an optional feature for DMAs involving file data transfers using Avalon-ST interface.
EOF [223]	1	EOF indicator for Avalon-ST streaming. In the H2D streaming, this bit causes the Avalon-ST Source interface to assert h2d_st_eof_o, indicating end of a file/packet. In the D2H streaming, this bit is set within the descriptor itself by a Writeback (if Writeback is enabled) when the user logic asserts d2h_st_eof_i, indicating end of a packet. Along with the EOF bit, Writeback also updates the actual received payload count (RX_PYLD_CNT) field of the last descriptor. *Note:* EOF bit is an optional feature for DMAs involving file data transfers using Avalon-ST interface.

continued...

Name	Width	Description
RSRVD [253:224]	30	Reserved
DESC_INVALID [254]	1	Indicates if current descriptor content is valid or stale
LINK [255]	1	Link =0 Descriptor contains the source address, destination address and length. Link = 1 Descriptor contains the address of the next 4 KB page in host memory containing the descriptors.

3.1.3.1. MSI-X/Write Back

MSI-X and Writeback block updates the host with the current processed queue's head pointer and interrupt. Apart from a global MSI-X Enable and Writeback Enable, there is a provision to selectively enable or disable the MSI-X and Writeback on a per-descriptor basis. This feature can be used by applications to throttle the MSI-X/ Writeback.

The table below shows the relation between global and per-descriptor MSI-X/ Writeback Enable.

Table 7. Multi Channel DMA Per-descriptor Enable vs. Global MSI-X/Writeback Enable

Global Enable	Per-descriptor Enable	MSI-X/Writeback Generation
1	1	On
1	0	On only for SOF/EOF and error conditions
0	1	Off
0	0	Off

If enabled, a writeback is sent to the host to update the status (completed descriptor ID) stored in Q_CONSUMED_HEAD_ADDR location. In addition, for D2H streaming DMA, an additional writeback is issued to the D2H descriptor itself when the IP's Avalon-ST sink interface has received an sof/eof from the user logic. It updates the D2H descriptor packet information fields such as start of a file/packet(SOF), end of a file/ packet(EOF), and received payload count (RX_PYLD_CNT).

3.2. User Logic Interface

Multi Channel DMA for PCI Express IP provides the following Avalon-MM / Avalon-ST interfaces to support user logic register access and H2D/D2H DMA data transfers:

- Avalon-MM PIO Master
- Avalon-ST Source
- Avalon-ST Sink
- Avalon-MM Write Master
- Avalon-MM Read Master

Note: For information about Avalon-MM and Avalon-ST interfaces, refer to the Avalon Interface specification.

Related Information

Avalon Interface Specifications

3.2.1. Avalon-MM PIO Master

Avalon-MM PIO Master is used to write/read control registers implemented in the user logic. PCIe BAR2 is mapped to the Avalon-MM PIO Master. Any TLP targeting BAR2 is forwarded to the user logic. TLP address targeting the PIO interface should be 8 bytes aligned. The PIO interface supports non-bursting 64-bit write and read transfers.

The Avalon-MM PIO Master is always present irrespective of the Interface type (Avalon-ST/Avalon-MM) that you select.

3.2.2. Avalon-ST Source (H2D) and Sink (D2H)

Avalon-ST Interface is used to send DMA data between the host and device through the streaming interface. You can enable the streaming interface by selecting Avalon-ST Interface type in the IP Parameter Editor. The Multi Channel DMA for PCI Express IP supports four Avalon-ST Source and Sink ports. Each port and DMA channel have 1:1 mapping.

In streaming the DMA data, the packet boundary is indicated by the SOF and EOF bit of the descriptor and corresponding sof and eof signals of the Avalon-ST interface.

Table 8. Packet boundary in streaming

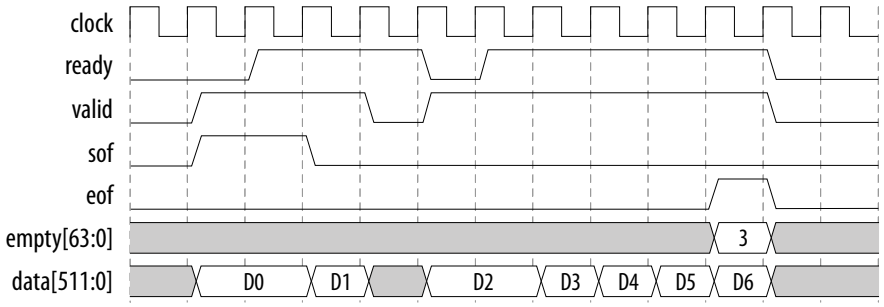
Packet boundary	Descriptor field	Avalon-ST Source (H2D) Interface Output	Avalon-ST Sink (D2H) Interface Input
Start of a packet	SOF bit	`h2d_st_sof_o`	`d2h_st_sof_i`
End of a packet	EOF bit	`h2d_st_eof_o`	`d2h_st_eof_i`

Head-of-Line Blocking Prevention

The H2D and D2H Data Movers service each channel independently based on a round-robin arbitration scheme. To prevent Head of Line blocking (HOL) in one Avalon-ST port from impacting the performance of other ports, Multi Channel DMA for PCI Express IP provides up to eight parallel Host-to-device descriptor fetch streams (4 for H2D descriptor fetch & 4 for D2H) and up to four parallel Host-to-device data streams. These data/descriptor fetch streams are independent of each other. Any persisting backpressure from an Avalon-ST Source port might stall one of the four H2D streams. However, the concurrent architecture along with round robin arbitration allows other streams to be mutually exclusive and operate effectively without any impact.

The following is the Avalon-ST interface timing for both H2D and D2H directions. A data transfer happens when both valid and ready signals become '1'. Both valid and ready signals can go to '0' within a packet boundary.

Figure 5. Avalon-ST Interface Timing Diagram



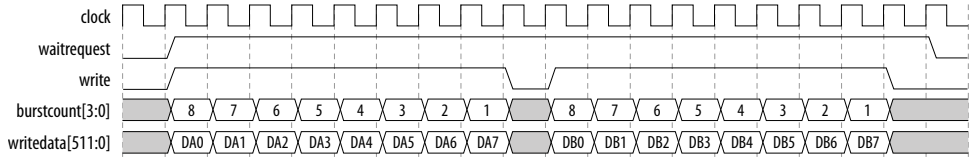
3.2.3. Avalon-MM Write (H2D) and Read (D2H) Master

Avalon-MM Interface is used to transfer data between the host and device through the memory-mapped interface. You can enable the Memory-Mapped interface by selecting AVMM Interface type in the IP Parameter Editor. The Multi Channel DMA for PCI Express IP supports 1 write master port and 1 read master port. You can map up to 8 DMA channels.

Avalon-MM Write Master

Avalon-MM Write Master is used to write H2D DMA data to the Avalon-MM slave in the user logic through the memory-mapped interface. The Write Master data width is 512 bits and can write up to 512 bytes of data per a write burst (burst count of 8). The `waitrequestAllowance` of this port is enabled and set to 16, allowing the master to transfer up to 16 additional write command cycles after the `waitrequest` signal has been asserted.

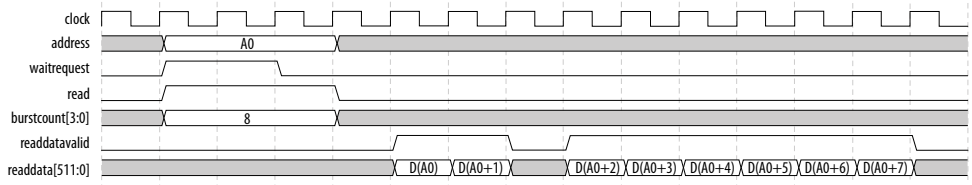
Figure 6. Avalon-MM Write with waitrequestAllowance 16



Avalon-MM Read Master

Avalon-MM Read Master is used to read D2H DMA data from the Avalon-MM slave in the user logic through the memory-mapped interface. The Read Master data width is 512 bits and can read up to 512 bytes of data per a read burst (burst count of 8).

Figure 7. Avalon-MM Read Master Timing Diagram



3.3. Control Registers

Multi Channel DMA for PCI Express IP provides 4 MB of control register space that is internally mapped to PCIe BAR0. The control register block contains the all the required registers to support the DMA operations. This includes QCSR space for individual queue control, MSI-X for interrupt generations, and GCSR for general global information.

The following table shows 4MB space mapped for each function in PCIe config space through BAR0.

Table 9. Control Registers

Address Space	Range	Size	Description
QCSR (D2H, H2D)	22'h00_0000 - 22'h0F_FFFF	1 MB	Individual queue control and status registers, up to 2048 D2H and 2048 H2D queues
MSI-X (Table and PBA)	22'h10_0000 - 22'h1F_FFFF	1 MB	MSI-X Table and PBA space
GCSR	22'h20_0000 - 22'h2F_FFFF	1 MB	General DMA control and status registers. Only for PF0.
Reserved	22'h30_0000 – 22'h3F_FFFF	1MB	Reserved

Note: For more information on Control registers, refer to Control Register (GCSR) on page 23

4. Registers

The Multi Channel DMA for PCI Express IP provides configuration, control and status registers to support the DMA operations including:

- D2H and H2D Queue control and status (QCSR)
- MSI-X Table and PBA for interrupt generation
- General/global DMA control (GCSR)

These Multi Channel DMA registers are mapped to BAR0 of a function.

Note: GCSR is only for PF0.

Following table shows 4 MB aperture space mapped for PF0 in PCIe config space through BAR0.

Table 10. Multi Channel DMA CSR Address Space

Address Space Name	Range	Size	Description
QCSR (D2H, H2D)	22'h00_0000 - 22'h0F_FFFF	1MB	Individual queue control registers. Up to 2048 D2H and 2048 H2D queues.
MSI-X (Table and PBA)	22'h10_0000 - 22'h1F_FFFF	1MB	MSI-X Table and PBA space
GCSR	22'h20_0000 - 22'h2F_FFFF	1MB	General DMA control and status registers.
Reserved	22'h30_0000 – 22'h3F_FFFF	1MB	Reserved

Following table shows how QCSR registers for each DMA channel are mapped with 1 MB space of QCSR.

Table 11. QCSR Address Space

Address Space Name	Size	DMA Channel	Size	Description
QCSR (D2H)	512 KB	DMA Channel 0	256 B	QCSR for DMA channel 0
		DMA Channel 1	256 B	QCSR for DMA channel 1
		….	….	….
		DMA Channel N	256 B	QCSR for DMA channel N
QCSR (H2D)	512 KB	DMA Channel 0	256 B	QCSR for DMA channel 0

continued...

Address Space Name	Size	DMA Channel	Size	Description
		DMA Channel 1	256 B	QCSR for DMA channel 1
		….	….	….
		DMA Channel N	256 B	QCSR for DMA channel 2

4.1. Queue Control (QCSR)

QCSR space contains queue control and status information. This register space of 1 MB can support up to 2048 H2D and 2048 D2H queues, where each queue is allocated 256 bytes of register space. The memory space allocated to each function is enough for each function to have allocated all the DMA Channels. However, the actual number will depend on the parameters input at IP generation time.

Address [7:0] : Registers for the queues

Address [18:8]: Queue number

Address [19]: 0 = D2H, 1=H2D

The following registers are defined for H2D/D2H queues. The base address for H2D and D2H are different, but registers (H2D and D2H) has the same address offsets.

Table 12. Queue Control Registers

Register Name	Address Offset	Access Type	Description
Q_CTRL	8'h00	R/W	Control Register
RESERVED	8'h04		RESERVED
Q_START_ADDR_L	8'h08	R/W	Lower 32-bit of queue base address in system memory. This is the beginning of the linked-list of 4KB pages containing the descriptors.
Q_START_ADDR_H	8'h0C	R/W	Upper 32-bit of queue base address in system memory. This is the beginning of the linked-list of 4KB pages containing the descriptors.
Q_SIZE	8'h10	R/W	Number of max entries in a queue. Powers of 2 only.
Q_TAIL_POINTER	8'h14	R/W	Current pointer to the last valid descriptor queue entry in the host memory.
Q_HEAD_POINTER	8'h18	RO	Current pointer to the last descriptor that was fetched. Updated by Descriptor Fetch Engine.
Q_COMPLETED_POINTER	8'h1C	RO	Last completed pointer after DMA is done. Software can poll this for status if Writeback is disabled.

continued...

Register Name	Address Offset	Access Type	Description
Q_CONSUMED_HEAD_ADDR_L	8'h20	R/W	Lower 32-bit of the sytem address where the ring consumed pointer is stored. This address is used for comsumed pointer writeback.
Q_CONSUMED_HEAD_ADDR_H	8'h24	R/W	Upper 32-bit of the sytem address where the ring consumed pointer is stored. This address is used for comsumed pointer writeback.
Q_BATCH_DELAY	8'h28	R/W	Delay the descriptor fetch until the time elapsed from a prior fetch exceeds the delayvalue in this register to maximize fetching efficiency.
RESERVED	8'h2C		RESERVED
RESERVED	18'h30		RESERVED
RESERVED	8'h34		RESERVED
Q_DEBUG_STATUS_1	8'h38	RO	RESERVED
Q_DEBUG_STATUS_2	8'h3C	RO	RESERVED
Q_DEBUG_STATUS_3	8'h40	RO	RESERVED
Q_DEBUG_STATUS_4	8'h44	RO	RESERVED
Q_RESET	8'h48	R/W	Queue reset requested

The following registers are defined for each implemented H2D and D2H queue. The total QCSR address space for each H2D/D2H is 256B and requires 8-bit of address.

Table 13. Q_CTRL (Offset 8'h0)

Bit [31:0]	Name	R/W	Default	Description
[31:10]	rsvd			Reserved
[9]	q_intr_en	R/W	0	If set, upon completion generate a MSI-X interrupt.
[8]	q_wb_en	R/W	0	If set, upon completion, do a write back.
[7:1]	rsvd			Reserved
[0]	q_en	R/W	0	Enable. Once it is enabled, the DMA starts fetching pending descriptors and executing them.

Table 14. Q_START_ADDR_L (Offset 8'h8)

Bit [31:0]	Name	R/W	Default	Description
[31:0]	q_strt_addr_l	R/W	0	After software allocate the descriptor ring buffer, it writes the lower 32-bit allocated address to this register. The descriptor fetch engine use this address and the pending head/tail pointer to fetch the descriptors.

Table 15. Q_START_ADDR_H (Offset 8'hC)

Bit [31:0]	Name	R/W	Default	Description
[31:0]	q_strt_addr_h	R/W	0	After software allocate the descriptor ring buffer, it writes the upper 32-bit allocated address to this register. The descriptor fetch engine use this address and the pending head/tail pointer to fetch the descriptors.

Table 16. Q_SIZE (Offset 8'h10)

Bit [31:0]	Name	R/W	Default	Description
[31:5]	rsvd			Reserved
[4:0]	q_size	R/W	1	Size of the descriptor ring in power of 2 and max value of 16. The unit is number of descriptors. Hardware will default to using a value of 1 if an illegal value is written. A value of 1 means queue size of 2 (2^1). A value is 16 (0x10) means queue size of 64K (2^16).

Table 17. Q_TAIL_POINTER (Offset 8'h14)

Bit [31:0]	Name	R/W	Default	Description
[31:16]	rsvd			Reserved
[15:0]	q_tl_ptr	R/W	0	After software sets up a last valid descriptor in the descriptor buffer, it programs this register with the position of the last (tail) valid descriptor

continued...

Bit [31:0]	Name	R/W	Default	Description
				that is ready to be executed. The DMA Descriptor Engine fetches descriptors from the buffer upto this position of the buffer

Table 18. Q_HEAD_POINTER (Offset 8'h18)

Bit [31:0]	Name	R/W	Default	Description
[31:16]	rsvd			Reserved
[15:0]	q_hd_ptr	R/W	0	After DMA Descriptor Fetch Engine fetches the descriptors from the descriptor buffer, upto the tail pointer, it updates this register with that last fetched descriptor position. The fetch engine only fetches descriptors if the head and tail pointer is not equal.

Table 19. Q_COMPLETED_POINTER (Offset 8'h1C)

Bit [31:0]	Name	R/W	Default	Description
[31:16]	rsvd			Reserved
[15:0]	q_cmpl_ptr	R/W	0	This register is updated by hardware to store the last descriptor position (pointer) that DMA has completed, that is all data for that descriptor and previous descriptors have arrived at the intended destinations. Software can poll this register to find out the status of the DMA for a specific queue.

Table 20. Q_CONSUMED_HEAD_ADDR_L (Offset 8'h20)

Bit [31:0]	Name	R/W	Default	Description
[31:0]	q_cnsm_hd_addr_l	R/W	0	Software programs this register with the lower 32-bit address location where the writeback will target after DMA is completed for a set of descriptors.

Table 21. Q_CONSUMED_HEAD_ADDR_H (Offset 8'h24)

Bit [31:0]	Name	R/W	Default	Description
[31:0]	q_cnsm_hd_addr_h	R/W	0	Software programs this register with the upper 32-bit address location where the writeback will target after DMA is completed for a set of descriptors.

Table 22. Q_BATCH_DELAY (Offset 8'h28)

Bit [31:0]	Name	R/W	Default	Description
[31:20]	rsvd			Reserved
[19:0]	q_batch_dscr_delay	R/W	0	Software programs this register with the the amount of time between fetches for descriptors. Each unit is 2ns.

Table 23. Q_RESET (Offset 8'h48)

Bit [31:0]	Name	R/W	Default	Description
[31:1]	rsvd			Reserved
[0]	q_reset	R/W	0	Request reset for the queue by writing 1'b1 to this register, and poll for value of 1'b0 when reset has been completed by hardware. Hardware clears this bit after completing the reset of a queue.

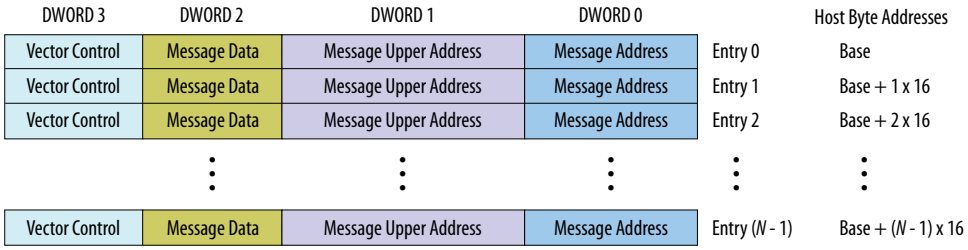
4.2. MSI-X Memory Space

The MSI-X Table and PBA memory is mapped to the second MB space of the Register address space. Allocated memory space can support up to 2048 MSI-X interrupts for a function. Actual amount of memory depends on the Multi Channel DMA for PCI Express IP configuration.

MSI-X Table

Each entry (vector) is 16 bytes (4 DWORDs) and is divided into Message Address, Data, and Mask (Vector Control) fields as shown in the figure below. To support 2048 interrupts, MSI-X Table requires 32 KB of space per function. But it is mapped to a 512 KB of space.

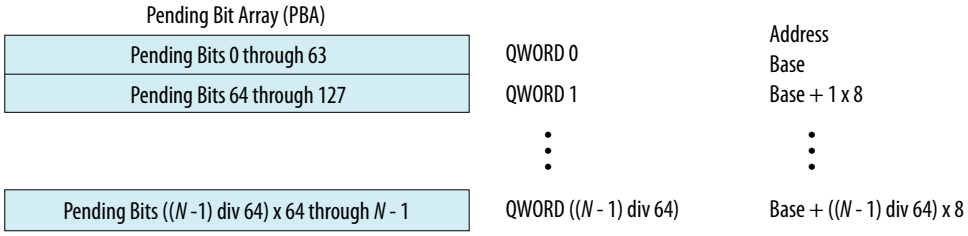
Figure 8. MSI-X Table Structure



MSI-X PBA

MSI-X PBA (Pending Bit Array) memory space is mapped to a 512 KB region. Actual amount of memory depends on the IP configuration. The Pending Bit Array contains the Pending bits, one per MSI-X Table entry, in array of QWORDs (64 bits). The PBA format is shown below.

Figure 9. MSI-X PBA Structure



Each DMA Channel is allocated 4 MSI-X vectors:

- 2'b00: H2D DMA Vector
- 2'b01: H2D Event Interrupt
- 2'b10: D2H DMA Vector
- 2'b11: D2H Event Interrupt

4.3. Control Register (GCSR)

This space contains global control/status registers that control the DMA operation. Access to this register set is restricted to PF0 only.

Table 24. Control Register

Register Name	Address Offset	Access Type	Description
CTRL	8'h00	R/W	Reserved
RESERVED	8'h04		Reserved
WB_INTR_DELAY	8'h08	R/W	Delay the writeback and/or the MSI-X interrupt until the time elapsed from a prior

continued...

Register Name	Address Offset	Access Type	Description
			writeback/interrupt exceeds the delay value in this register.
RESERVED	8'h0C – 8'h6F		Reserved
VER_NUM	8'h70	RO	Multi Channel DMA for PCI Express IP version number

Table 25. CTRL (Offset 8'h0)

Bit [31:0]	Name	R/W	Default	Deacription
[31:0]	rsvd			Reserved

Table 26. WB_INTR_DELAY (Offset 8'h08)

Bit [31:0]	Name	R/W	Default	Deacription
[31:20]	rsvd			Reserved
[19:0]	wb_intr_delay	R/W	0	Delay the writeback and/or the MSI-X interrupt until the time elapsed from a prior writeback/ interrupt exceeds the delay value in this register. Each unit is 2ns.

Table 27. VER_NUM (Offset 8'h70)

Bit [31:0]	Name	R/W	Default	Deacription
[31:16]	rsvd			RESERVED
[15:8]	MAJ_VER	RO	1	Major version number of Multi Channel DMA for PCI Express IP
[7:0]	MIN_VER	RO	0	Minor version number of Multi Channel DMA for PCI Express IP

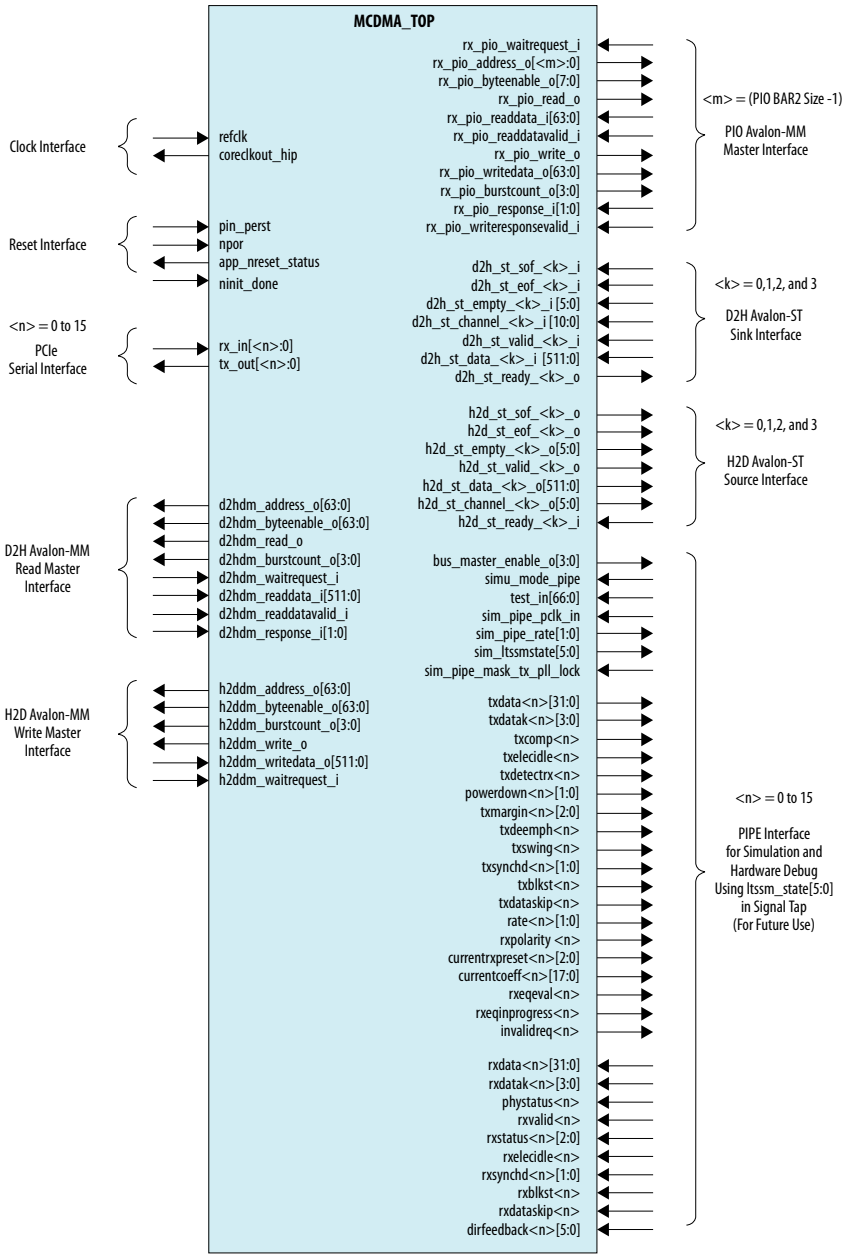
5. Interface Overview

Interfaces for the Multi Channel DMA for PCI Express IP are:

- Avalon-MM Write Master interface – for H2D DMA
- Avalon-ST Source Interface- for H2D DMA
- Avalon-MM Read Master interface – for D2H DMA
- Avalon-ST Sink Interface- for D2H DMA
- Avalon-MM PIO Master – for DMA-Bypass interface

5.1. Port List

Figure 10. Multi Channel DMA for PCI Express IP Port List



5.2. Clocks

Table 28. Multi Channel DMA for PCI Express IP Clocks

Signal Name	I/O Type	Description
refclk	Input	PCIe reference clock defined by the PCIe specification. 100 MHz ± 300 ppm. This input reference clock must be stable and free-running at device power-up for a successful device configuration.
coreclkout_hip	Output	This is an output clock provided to user logic. 250 MHz for Gen3 x16. Avalon-MM / Avalon-ST user interfaces are synchronous to this clock.

5.3. Resets

Table 29. Multi Channel DMA for PCI Express IP Resets

Signal Name	I/O Type	Description
pin_perst	Input	This is an active-low input to the PCIe Hard IP, and implements the PERST# function defined by the PCIe specification.
npor	Input	Application drives this active-low reset input to the PCIe Hard IP. This resets entire PCIe Hard IP. If not used, you must tie this input to 1.
app_nreset_status	Output	This is an active low reset status. This is deasserted after the PCIe Hard IP has come out of reset.
ninit_done	Input	This is an active low input signal. A "1" indicates that the FPGA device is not yet fully configured. A "0" indicates the device has been configured and is in normal operating mode. To use the ninit_done input, instantiate the Reset Release Intel FPGA IP in your design and use its ninit_done output. The Reset Release IP is required in Intel Stratix 10 design. It holds the Multi Channel DMA for PCI Express IP in reset until the FPGA is fully configured and has entered user mode.

5.4. Avalon-MM PIO Master

The Avalon-MM PIO Master interface is used to write to /read from external registers implemented in the user logic.

Table 30. Avalon-MM PIO Master

Signal Name	I/O Type	Description
rx_pio_address_o[21:0]	Output	PIO Write Address.
rx_pio_writedata_o[63:0]	Output	PIO Write Data Payload.
rx_pio_byteenable_o[7:0]	Output	PIO Write Data Byte Enable.
rx_pio_write_o	Output	PIO Write.
rx_pio_read_o	Output	PIO Read
rx_pio_burstcount_o[3:0]	Output	PIO Write Burst Count.
rx_pio_waitrequest_i	Input	PIO Write WaitRequest.
rx_pio_writeresponsevalid_i	Input	PIO response valid to a write request
rx_pio_readdata_i[63:0]	Input	PIO Read Data.
rx_pio_readdatavalid_i	Input	PIO Read data valid
rx_pio_response_i[1:0]	Input	PIO response. Reserved for future release. Tie to 0.

5.5. Avalon-MM Write Master (H2D)

The H2D Avalon-MM Write Master interface is used to write H2D DMA data to the external Avalon-MM slave. This master is 512-bit write master that is capable of writing maximum 512 Byte of data per Avalon-MM transaction. The `WaitRequestAllowance` of this port is enabled and set to 16 allowing the master to transfer continuously 16 data phases after the `WaitRequest` signal has been asserted.

Table 31. Avalon-MM Write Master (H2D)

Signal Name	I/O Type	Description
h2ddm_waitrequest_i	Input	H2D Wait Request
h2ddm_write_o	Output	H2D Write
h2ddm_address_o[63:0]	Output	H2D Write Address
h2ddm_burstcount_o[3:0]	Output	H2D Write Burst Count
h2ddm_writedata_o[511:0]	Output	H2D Write Data Payload
h2ddm_byteenable_o[63:0]	Output	H2D Byte Enable

5.6. Avalon-MM Read Master (D2H)

The D2H Avalon-MM Read Master interface is use to read D2H DMA data from the external AVMM slave. This port is 512-bit master that is capable of reading maximum 512 bits of data per Avalon-MM transaction.

Table 32. Avalon-MM Read Master (D2H)

Signal Name	I/O Type	Description
d2hdm_read_o	Output	D2H Read.
d2hdm_address_o[63:0]	Output	D2H Read Write Address.
d2hdm_byteenable_o[63:0]	Output	D2H Byte Enable
d2hdm_burstcount_o[3:0]	Output	D2H Burst Count.
d2hdm_waitrequest_i	Input	D2H Write WaitRequest.
d2hdm_readdatavalid_i	Input	D2H Read Data Valid.
d2hdm_readdata_i[511:0]	Input	D2H Read Data.
d2hdm_response_i[1:0]	Input	Tied to 0

5.7. Avalon-ST Source (H2D)

The H2D Avalon-ST source interface is used to send H2D DMA data to the external Avalon-ST sink logic.

Table 33. Avalon-ST Source (H2D)

<n>=0,1,2,3

Signal Name	I/O Type	Description
h2d_st_data_<n>_o[511:0]	Output	H2D Streaming data from host to device
h2d_st_valid_<n>_o	Output	Valid for all outgoing signals. A '1' represents the readiness of data to be sent.
h2d_st_ready_<n>_i	Input	Backpressure from device. A '1' represents, device readiness for receiving data.
h2d_st_sof_<n>_o	Output	Start of file (or packet) as instructed in host descriptor.
h2d_st_eof_<n>_o	Output	End of file (or packet) as instructed in host descriptor.
h2d_st_empty_<n>_o[5:0]	Output	Represents the number of empty bytes in h2d_st_data_<n>_o, and valid only when both h2d_st_valid_<n>_o and h2d_st_eop_<n>_o is '1'.
h2d_st_channel_<n>_o[10:0]	Output	To support multi-Channel per port. For future purpose.

5.8. Avalon-ST Sink (D2H)

The D2H Avalon-ST Sink interface is used to read D2H DMA data from the external Avalon-ST source logic.

Table 34. Avalon-ST Sink (D2H)

<n>=0,1,2,3

Signal Name	I/O Type	Description
d2h_st_valid_<n>_i	Input	Valid for all incoming signals. A '1' represents the device readiness for data to be sent.
d2h_st_data_<n>_i[511:0]	Input	D2H Streaming data from device to host.
d2h_st_ready_<n>_o	Output	Backpressure from Multi Channel DMA for PCI Express IP. A '1' represents, IP readiness for receiving data.
d2h_st_empty_<n>_i[5:0]	Input	Represents the number of empty bytes in d2h_st_data_<n>_i, and valid only when both d2h_st_valid__<n>_i and d2h_st_eop__<n>_i is '1'.
d2h_st_sof_<n>_i	Input	Start of file (or packet) as instructed by the user logic.
d2h_st_eof_<n>_i	Input	End of file (or packet) as instructed by the user logic.
d2h_st_channel_<n>_i[10:0]	input	To support multi-Channel per port. For future purpose.

6. Parameters

This chapter provides a reference for all the parameters of the Multi Channel DMA for PCI Express IP.

Table 35. Design Environment Parameter

Starting in Intel Quartus Prime 18.0, there is a new parameter **Design Environment** in the parameters editor window.

Parameter	Value	Description
Design Environment	**Standalone** **System**	Identifies the environment that the IP is in. • The **Standalone** environment refers to the IP being in a standalone state where all its interfaces are exported. • The **System** environment refers to the IP being instantiated in a Platform Designer system.

6.1. IP Settings

6.1.1. System Settings

Figure 11. Multi Channel DMA for PCI Express IP Parameter Editor

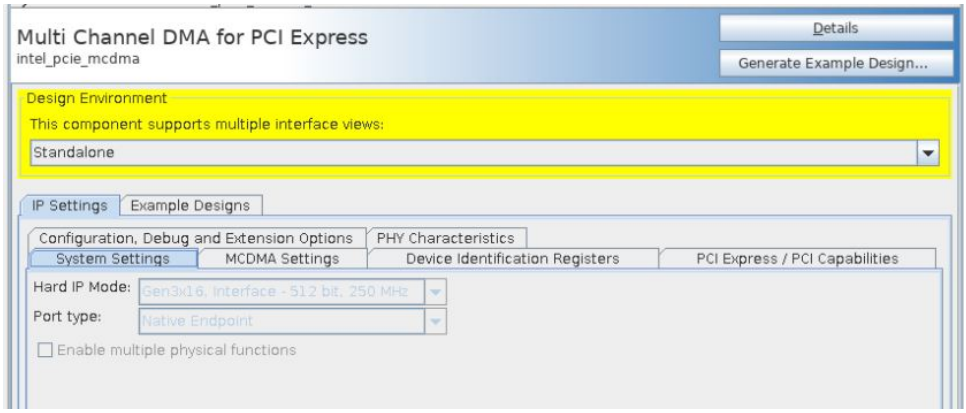


Table 36. System Settings

Parameter	Value	Description
Hard IP mode	**Gen3x16, 512-bit interface, 250 MHz**	Selects the following elements: • The lane data rate. Gen3 is supported • The Application Layer interface frequency

continued...

Parameter	Value	Description
		The width of the data interface between the hard IP Transaction Layer and the Application Layer implemented in the FPGA fabric.
Port type	**Native Endpoint**	Specifies the port type.

6.1.2. MCDMA Settings

Figure 12. MCDMA Settings Parameters

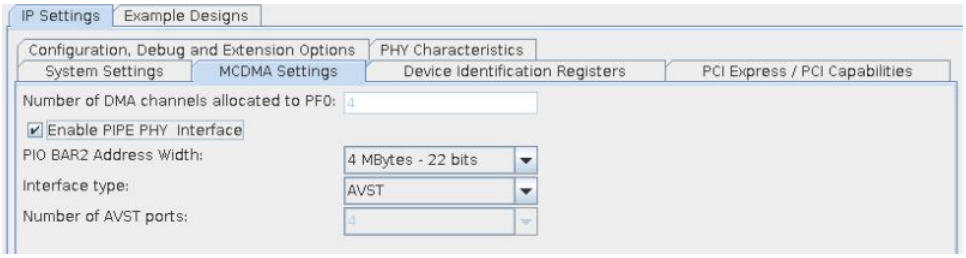


Table 37. MCDMA Settings Table

Parameter	Value	Description
Number of DMA Channels allocated to PF0	For Avalon-ST Interface type: 1 For Avalon-MM Interface type: 1 - 8	Number of DMA Channels between the host and device PF0 Avalon-ST / Avalon-MM ports. For Avalon-ST interface type, only 1 channel per port is supported. For Avalon-MM Interface type, up to 8 channels are supported.
Enable PIPE PHY Interface	On / Off	PIPE PHY Interface is for simulation only. This should be enabled for example design generation. Default: On
PIO BAR2 Address Width	7 - 63 bits	Address width for PIO AVMM port. Default address width is 22 bits
Interface Type	Avalon-ST / Avalon-MM	User logic interface type for D2HDM and H2DDM. Default: Avalon-ST Interface
Number of Avalon-ST/Avalon-MM Ports	For Avalon-ST Interface Type : 4 For Avalon-MM Interface Type: 1	In the Intel Quartus Prime 20.2 release, Number of Avalon-ST/Avalon-MM Ports is fixed.

6.1.3. Device Identification Registers

The following table lists the default values of the read-only registers in the PCI* Configuration Header Space. You can use the parameter editor to set the values of these registers.

You can specify Device ID registers for each Physical Function.

Note: In the Intel Quartus Prime Pro Edition 20.2 Release, only PF0 is supported.

Table 38. PCI Header Configuration Space Registers

Register Name	Default Value	Description
Vendor ID	0x00001172	Sets the read-only value of the `Vendor ID` register. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification*. Address offset: 0x000.
Device ID	0x00000000	Sets the read-only value of the `Device ID` register. Address offset: 0x000.
VF Device ID	0x00000000	Sets the read-only value of the `Device ID` register.
Revision ID	0x00000001	Sets the read-only value of the `Revision ID` register. Address offset: 0x008.
Class code	0x00000000	Sets the read-only value of the `Class Code` register. You must set this register to a non-zero value to ensure correct operation. Address offset: 0x008.
Subsystem Vendor ID	0x00000000	Sets the read-only value of `Subsystem Vendor ID` register in the PCI Type 0 Configuration Space. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification*. This value is assigned by PCI-SIG to the device manufacturer. This value is only used in Root Port variants. Address offset: 0x02C.
Subsystem Device ID	0x00000000	Sets the read-only value of the `Subsystem Device ID` register in the PCI Type 0 Configuration Space. This value is only used in Root Port variants. Address offset: 0x02C

6.1.4. PCI Express / PCI Capabilities Parameters

This group of parameters defines various capability properties of the IP core. Some of these parameters are stored in the PCI Configuration Space - PCI Compatible Configuration Space. The byte offset indicates the parameter address.

6.1.4.1. Device Capabilities

Table 39. Device Registers

Parameter	Possible Values	Default Value	Address	Description
Maximum payload sizes supported	**512 bytes** *Note:* Value is fixed at 512 bytes	512 bytes	0x074	Specifies the maximum payload size supported. This parameter sets the read-only value of the max payload size supported field of the Device Capabilities register.
Function level reset	TBD	TBD	N/A	TBD

6.1.4.2. Link Capabilities

Table 40. Link Capabilities

Parameter	Value	Description
Link port number (Root Port only)	**0x01**	Sets the read-only value of the port number field in the `Link Capabilities` register. This parameter is for Root Ports only. It should not be changed.
Slot clock configuration	**On/Off**	When you turn this option **On**, indicates that the Endpoint uses the same physical reference clock that the system provides on the connector. When **Off**, the IP core uses an independent clock regardless of the presence of a reference clock on the connector. This parameter sets the Slot Clock Configuration bit (bit 12) in the `PCI Express Link Status` register.

6.1.4.3. MSI-X Capabilities

Table 41. MSI-X Capabilities

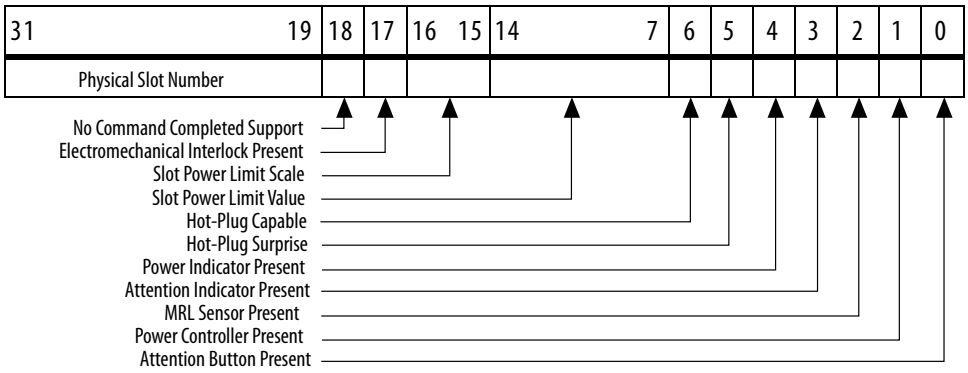
Parameter	Value	Address	Description
MSI-X Capabilities			
Enable MSI-X	**Always On**		When **On**, adds the MSI-X capability structure, with the parameters shown below. *Note:* Note: In Intel Quartus Prime Pro Edition 20.2 Release, Multi Channel DMA for PCI Express IP supports only internal MSI-X generation and doesn't support user MSI-X event. Hence the parameters shown below are grayed out
	Bit Range		
Table size	[10:0]	0x068[26:16]	System software reads this field to determine the MSI-X Table size <*n*>, which is encoded as <*n–1*>. For example, a returned value of 2047 indicates a table size of 2048. This field is read-only in the MSI-X Capability Structure. Legal range is 0–2047 (2^{11}). VF's share a common Table Size. VF Table BIR/Offset, and PBA BIR/Offset are fixed at compile time. BAR4 accesses these tables. The table Offset field = 0x600. The PBA Offset field =0x400 for SRIOV. You must implement an MSI-X table. If you do not intend to use MSI-X, you may program the table size to 1.
Table offset	[31:0]		Points to the base of the MSI-X Table. The lower 3 bits of the table BAR indicator (BIR) are set to zero by software to form a 64-bit qword-aligned offset. This field is read-only.
Table BAR indicator	[2:0]		Specifies which one of a function's BARs, located beginning at 0x10 in Configuration Space, is used to map the MSI-X table into memory space. This field is read-only. Legal range is 0–5.
Pending bit array (PBA) offset	[31:0]		Used as an offset from the address contained in one of the function's Base Address registers to point to the base of the MSI-X PBA. The lower 3 bits of the PBA BIR are set to zero by software to form a 32-bit qword-aligned offset. This field is read-only in the MSI-X Capability Structure. (1)
Pending BAR indicator	[2:0]		Specifies the function Base Address registers, located beginning at 0x10 in Configuration Space, that maps the MSI-X PBA into memory space. This field is read-only in the MSI-X Capability Structure. Legal range is 0–5.

6.1.4.4. Slot Capabilities

Table 42. Slot Capabilities

Parameter	Value	Description
Use Slot register	**On/Off**	This parameter is only supported in Root Port mode. The slot capability is required for Root Ports if a slot is implemented on the port. Slot status is recorded in the `PCI Express Capabilities` register. Defines the characteristics of the slot. You turn on this option by selecting **Enable slot capability**. Refer to the figure below for bit definitions.
Slot power scale	`0–3`	Specifies the scale used for the **Slot power limit**. The following coefficients are defined: • 0 = 1.0x • 1 = 0.1x • 2 = 0.01x • 3 = 0.001x The default value prior to hardware and firmware initialization is b'00. Writes to this register also cause the port to send the `Set_Slot_Power_Limit` Message. Refer to Section 6.9 of the *PCI Express Base Specification Revision* for more information.
Slot power limit	`0–255`	In combination with the **Slot power scale value**, specifies the upper limit in watts on power supplied by the slot. Refer to Section 7.8.9 of the *PCI Express Base Specification* for more information.
Slot number	`0-8191`	Specifies the slot number.

Figure 13. Slot Capability



(1) Throughout this user guide, the terms word, DWORD and qword have the same meaning that they have in the *PCI Express Base Specification*. A word is 16 bits, a DWORD is 32 bits, and a qword is 64 bits.

6.1.4.5. Power Management

Table 43. Power Management Parameters

Parameter	Value	Description
Endpoint L0s acceptable latency	**Maximum of 64 ns** **Maximum of 128 ns** **Maximum of 256 ns** **Maximum of 512 ns** **Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **No limit**	This design parameter specifies the maximum acceptable latency that the device can tolerate to exit the L0s state for any links between the device and the root complex. It sets the read-only value of the Endpoint L0s acceptable latency field of the `Device Capabilities Register` (0x084). This Endpoint does not support the L0s or L1 states. However, in a switched system there may be links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 64 ns. This is a safe setting for most designs.
Endpoint L1 acceptable latency	**Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **Maximum of 8 us** **Maximum of 16 us** **Maximum of 32 us** **Maximum of 64 ns** **No limit**	This value indicates the acceptable latency that an Endpoint can withstand in the transition from the L1 to L0 state. It is an indirect measure of the Endpoint's internal buffering. It sets the read-only value of the Endpoint L1 acceptable latency field of the `Device Capabilities Register`. This Endpoint does not support the L0s or L1 states. However, a switched system may include links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 1 µs. This is a safe setting for most designs.

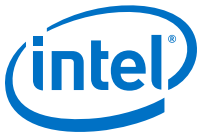
The Intel Stratix 10 Avalon-ST Hard IP for PCI Express and Intel Stratix 10 Avalon-MM Hard IP for PCI Express do not support the L1 or L2 low power states. If the link ever gets into these states, performing a reset (by asserting `pin_perst`, for example) allows the IP core to exit the low power state and the system to recover.

These IP cores also do not support the in-band beacon or sideband WAKE# signal, which are mechanisms to signal a wake-up event to the upstream device.

6.1.4.6. Vendor Specific Extended Capability (VSEC)

Table 44. VSEC

Parameter	Value	Description
User ID register from the Vendor Specific Extended Capability	Custom value	Sets the read-only value of the 16-bit User ID register from the Vendor Specific Extended Capability. This parameter is only valid for Endpoints.



6.1.5. Configuration, Debug and Extension Options

Table 45. Configuration, Debug and Extension Options

Parameter	Value	Description
Enable Hard IP dynamic reconfiguration of PCIe read-only registers	**Off**	Parameter is not supported.
Enable transceiver dynamic reconfiguration	**Off**	Parameter is not supported.
Enable Native PHY, LCPLL, and fPLL ADME for Toolkit	**Off**	Parameter is not supported.
Enable PCIe Link Inspector	**Off**	Parameter is not supported.

6.1.6. PHY Characteristics

Table 46. PHY Characteristics

Parameter	Value	Description
Gen2 TX de-emphasis	**3.5dB** **6dB**	Specifies the transmit de-emphasis for Gen2. Intel recommends the following settings: • 3.5 dB: Short PCB traces • 6.0 dB: Long PCB traces.
VCCR/VCCT supply voltage for the transceiver	**1_1V** **1_0V**	Allows you to report the voltage supplied by the board for the transceivers.

6.2. Example Designs

Table 47. Example Designs

Parameter	Value	Description
Currently Selected Example Design	**PIO using MQDMA Bypass mode** (default) (For AVST Interface type only) • **Device-side Packet loopback** • **Packet Generate/ Check** (For AVMM Interface type only) **AVMM DMA**	Select an example design available from the pulldown list. Avalon-ST/Avalon-MM Interface type setting determines available example designs
Simulation	**On/Off**	When **On**, the generated output includes a simulation model.

continued...

Parameter	Value	Description
Select simulation Root Complex BFM	**Third-party BFMIntel FPGA BFM**	Choose the appropriate BFM for simulation. **Intel FPGA BFM:** Default. This bus functional model (BFM) supports x16 configurations by downtraining to x8. **Third-party BFM:** Select this If you want to simulate all 16 lanes using a third-party BFM.
Synthesis	**On/Off**	When **On**, the generated output includes a synthesis model.
Generated HDL format	**Verilog/VHDL**	Only Verilog HDL is available in the current release.
Target Development Kit	**None** **Intel Stratix 10 GX H-Tile Development Kit** **Intel Stratix 10 MX H-Tile Development Kit**	Select the appropriate development board. If you select one of the development boards, system generation overwrites the device you selected with the device on that development board. *Note:* If you select **None**, system generation does not make any pin assignments. You must make the assignments in the `.qsf` file.

Note: For more information about example designs, refer to the PCIe Multi-Channel Direct Memory Access IP for H-Tile Design Example User Guide.

Related Information

Multi Channel DMA for PCI Express IP Design Example User Guide

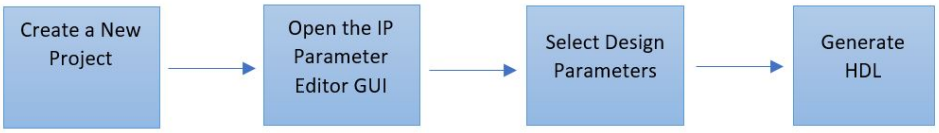
7. Designing with the IP Core

7.1. Generating the IP Core

You can use the Intel Quartus Prime Pro Edition IP Catalog or Platform Designer to define and generate an Intel Stratix 10 H-Tile Multi Channel DMA for PCI Express IP custom component.

Follow the steps shown in the figure below to generate a custom Multi Channel DMA for PCI Express IP component.

Figure 14. IP Generation Flowchart



You can select Multi Channel DMA for PCI Express IP in the Intel Quartus Prime Pro Edition IP Catalog or Platform Designer as shown below:

Figure 15. Intel Quartus Prime Pro Edition IP Catalog (with filter applied)

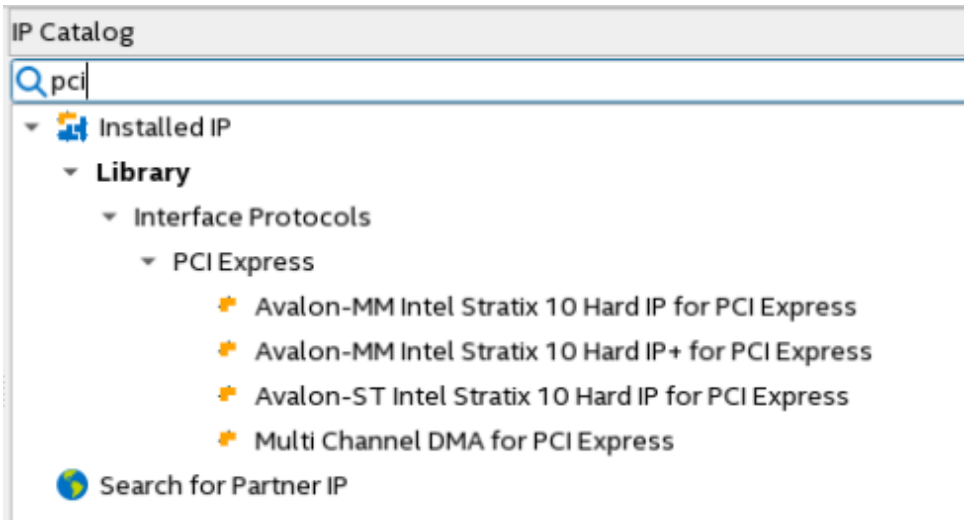
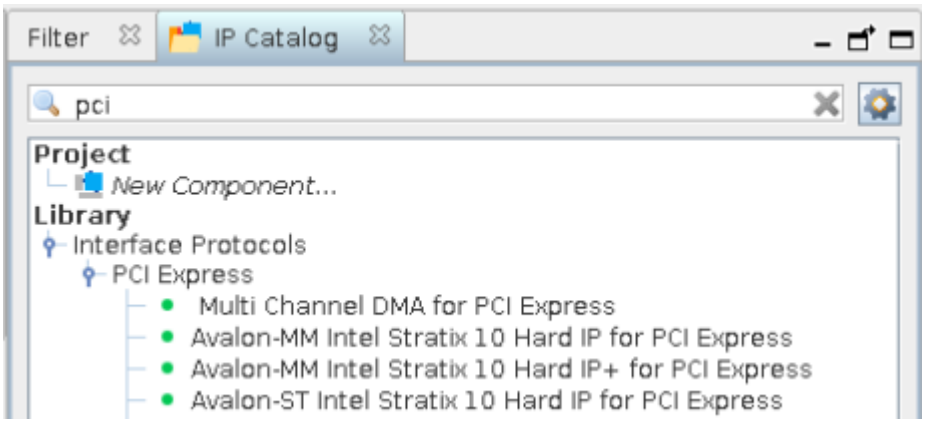


Figure 16. Platform Designer IP Catalog (with filter applied)

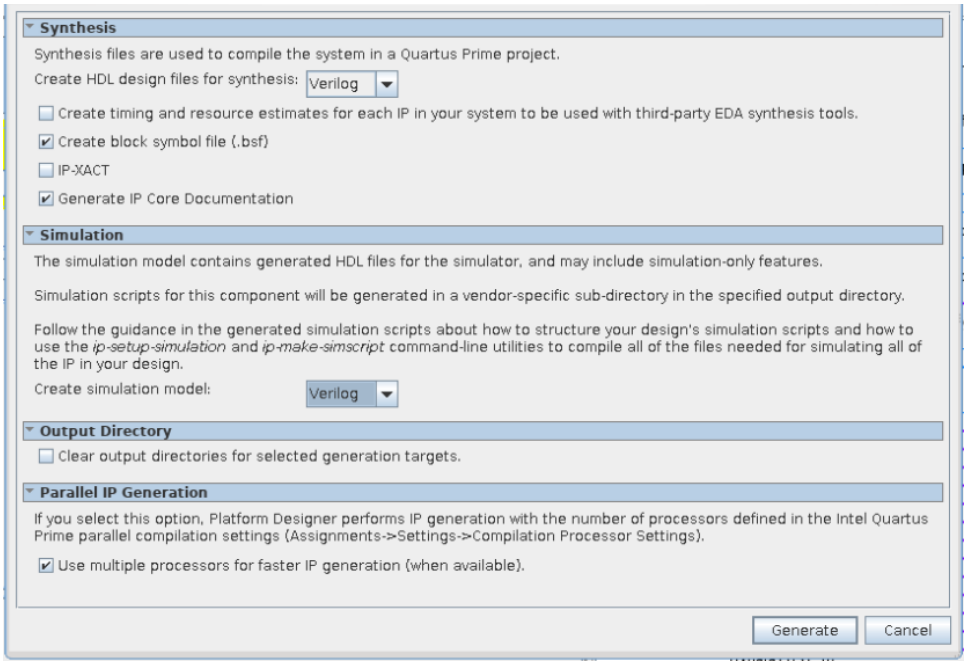


7.2. Simulating the IP Core

The Intel Quartus Prime Pro Edition software optionally generates a functional simulation model, a testbench or design example, and vendor-specific simulator setup scripts when you generate your parameterized Multi Channel DMA for PCI Express IP core. For Endpoints, the generation creates a Root Port BFM. There is no support for Root Ports in this release of the Intel Quartus Prime Pro Edition.

To enable IP simulation model generation, set **Create simulation model** to **Verilog** or **VHDL** when you generate HDL:

Figure 17. Multi Channel DMA for PCI Express IP Simulation in Intel Quartus Prime Pro Edition



The Intel Quartus Prime Pro Edition supports the following simulators.

Table 48. Supported Simulators

Vendor	Simulator	Version	Platform
Aldec	Active-HDL	10.3	Windows
Aldec	Riviera-PRO	2016.10	Windows, Linux
Cadence	Incisive Enterprise (NCSim)	15.20	Linux
Cadence	Xcelium Parallel Simulator	17.04.014	Linux
Mentor Graphics	ModelSim PE	10.5c	Windows
Mentor Graphics	ModelSim SE	10.5c	Windows, Linux
Mentor Graphics	QuestaSim	10.5c	Windows, Linux
Synopsys	VCS/VCS MX	2016, 06-SP-1	Linux

Note: The Intel testbench and Root Port BFM provide a simple method to do basic testing of the Application Layer logic that interfaces to the PCIe IP variation. This BFM allows you to create and run simple task stimuli with configurable parameters to exercise basic functionality of the example design. The testbench and Root Port BFM are not intended to be a substitute for a full verification environment. Corner cases and certain traffic profile stimuli are not covered. To ensure the best verification coverage possible, Intel strongly recommends that you obtain commercially available PCIe verification IP and tools, or do your own extensive hardware testing, or both.

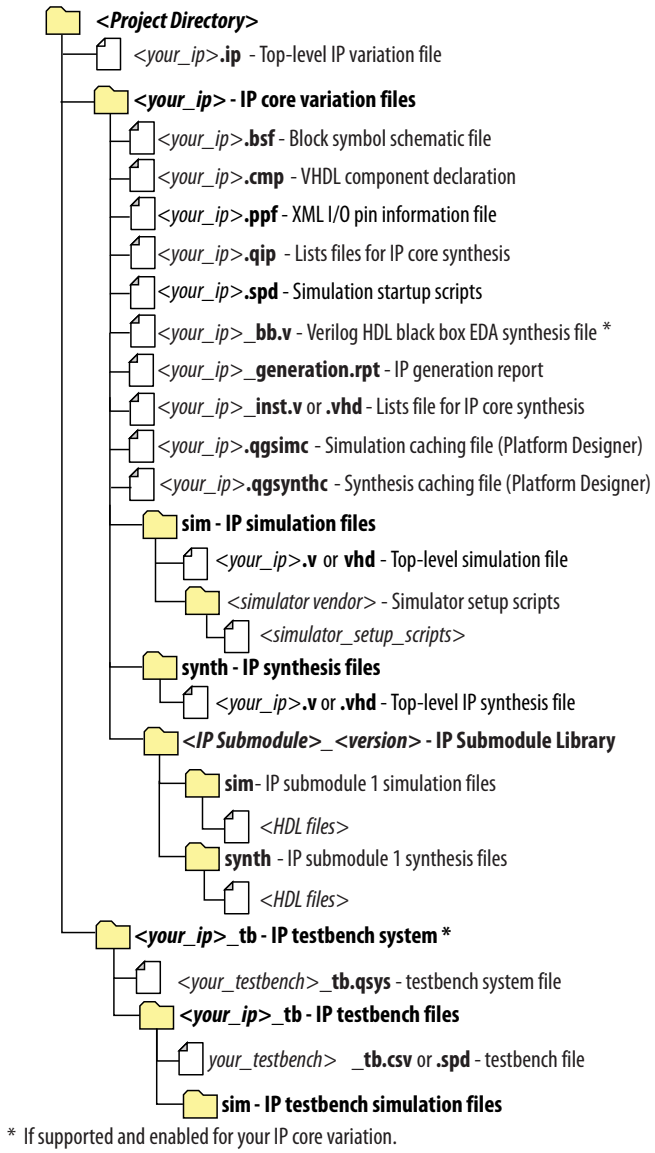
Related Information

- Introduction to Intel FPGA IP Cores
- Simulating Intel FPGA IP Cores
- Simulation Quick-Start
- Multi Channel DMA for PCI Express Design Example User Guide

7.3. IP Core Generation Output - Intel Quartus Prime Pro Edition

The Intel Quartus Prime Pro Edition software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.

Figure 18. Individual IP Core Generation Output (Intel Quartus Prime Pro Edition)



* If supported and enabled for your IP core variation.

Table 49. Output Files of Intel FPGA IP Generation

File Name	Description
<your_ip>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a .qsys file.
<your_ip>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<your_ip>_generation.rpt	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.

continued...

File Name	Description
<*your_ip*>.qgsimc (Platform Designer systems only)	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qgsynth (Platform Designer systems only)	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qip	Contains all information to integrate and compile the IP component.
<*your_ip*>.csv	Contains information about the upgrade status of the IP component.
<*your_ip*>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (.bdf).
<*your_ip*>.spd	Input file that ip-make-simscript requires to generate simulation scripts. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.
<*your_ip*>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.
<*your_ip*>_bb.v	Use the Verilog blackbox (_bb.v) file as an empty module declaration for use as a blackbox.
<*your_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_ip*>.regmap	If the IP contains register information, the Intel Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Intel Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system slaves, Platform Designer accesses the registers by name.
<*your_ip*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
/mentor/	Contains a msim_setup.tcl script to set up and run a ModelSim simulation.
/aldec/	Contains a Riviera*-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs/ /synopsys/vcsmx/	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file to set up and run a VCS MX* simulation.
/cadence/	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules/	Contains HDL files for the IP core submodule.
/<*IP submodule*>/	Platform Designer generates /synth and /sim sub-directories for each IP submodule directory that Platform Designer generates.

7.4. Systems Integration and Implementation

7.4.1. Clock Requirements

The Multi Channel DMA for PCI Express IP Core has a single 100 MHz input clock and a single output clock. An additional clock is available for Intel internal simulations only.

refclk

Each instance of the Multi Channel DMA for PCI Express IP core has a dedicated **refclk** input signal. This input reference clock can be sourced from any reference clock in the transceiver tile. Refer to the *Intel Stratix 10 Device Family Pin Connection Guidelines* for additional information on termination and valid locations.

coreclkout_hip

This output clock is a fixed frequency clock that drives the Application Layer. The output clock frequency is fixed to 250 MHz based on the maximum link speed and width settings in the IP Parameter Editor.

sim_pipe_pclk_in

This input clock is for internal Intel simulation only.

7.4.2. Reset Requirements

The Multi Channel DMA for PCI Express IP Core has two, asynchronous, active low reset inputs, `npor` and `pin_perst`. Both reset the Transaction, Data Link and Physical Layers.

npor

The Application Layer drives the **npor** reset input to the PCIe IP core. If you choose to design an Application Layer does not drive **npor**, you must tie this output to `1'b1`. The **npor** signal resets all registers and state machines to their initial values.

pin_perst

This is the PCIe Fundamental Reset signal. Asserting this signal returns all registers and state machines to their initialization values. Each instance of the PCIe IP core has a dedicated **pin_perst** pin. You must connect the **pin_perst** of each hard IP instance to the corresponding nPERST pin of the device. These pins have the following location.

- `NPERSTL0` : Bottom Left PCIe IP core and Configuration via Protocol (CvP)
- `NPERSTL1`: Middle Left PCIe PCIe IP core (When available)
- `NPERSTL2`: Top Left PCIe IP core (When available)
- `NPERSTR0`: Bottom Right PCIe IP core (When available)
- `NPERSTR1`: Middle Right PCIe IP core (When available)
- `NPERSTR2`: Top Right PCIe IP core (When available)

For maximum compatibility, always use the bottom PCIe IP core on the left side of the device first. This is the only location that supports CvP using the PCIe link.

7.4.3. Required Supporting IP

Intel Stratix 10 devices use a parallel, sector-based architecture that distributes the core fabric logic across multiple sectors. Device configuration proceeds in parallel with each Local Sector Manager (LSM) configuring its own sector. Consequently, FPGA registers and core logic are not released from reset at exactly the same time, as has always been the case in previous families.

In order to keep application logic held in the reset state until the entire FPGA fabric is in user mode, Intel Stratix 10 devices require you to include the Intel Stratix 10 Reset Release IP.

Refer to the Multi Channel DMA for PCI Express IP design example to see how the Reset Release IP is connected with the Multi Channel DMA for PCI Express IP component.

Related Information

AN 891: Using the Reset Release Intel FPGA IP

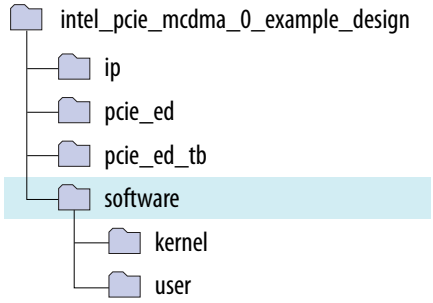
8. Software Programming Model

The Multi Channel DMA for PCI Express IP Linux software consists of the following components:

- Test Application
- Kernel Framework
- User Space Library and API

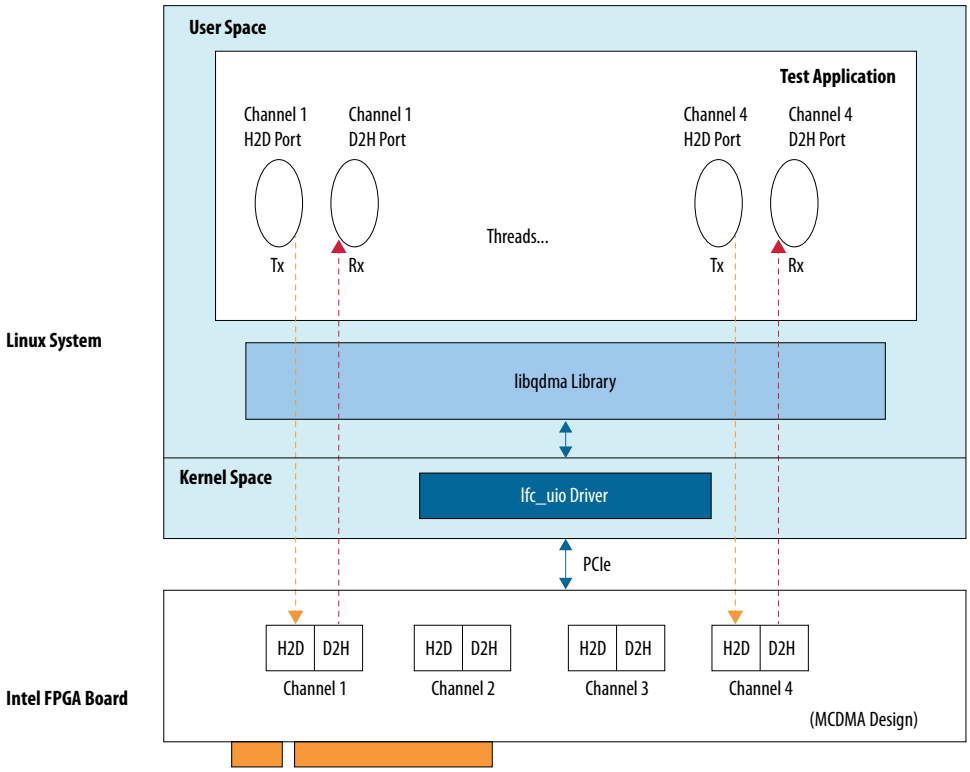
The software files are created in the Multi Channel DMA for PCI Express IP design example project folder when you generate an Multi Channel DMA for PCI Express IP design example from the IP Parameter Editor as shown below. The software configuration is specific to the example design generated by Intel Quartus Prime.

Figure 19. Software Folder Structure



8.1. Architecture

Figure 20. Block Level Software Architecture



The major blocks of Multi Channel DMA for PCI Express IP software architecture are

ifc_uio

- O/S framework running in the kernel space
- Probes and exports channel BARs to libmqdma
- Supports Interrupt notification/clearing

libmqdma

This is an user-space library used by the application to access the PCIe device.

- This library has the APIs to access the MCDMA design and users can develop their application using this API.
- It features calls for allocating and releasing the channel.

Sample application

This application uses the APIs from libmqdma and will take the following command line arguments as the input.

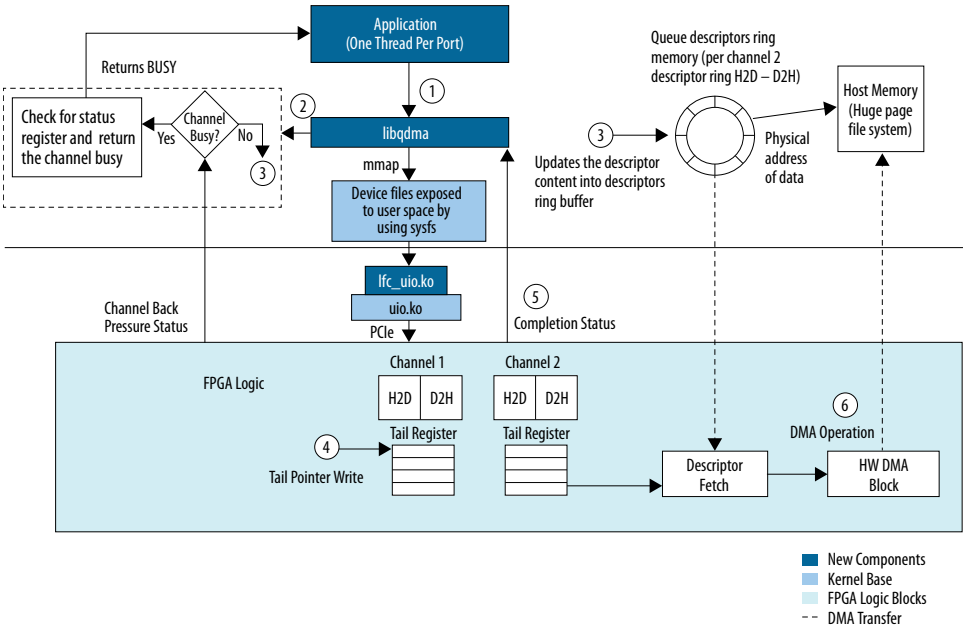
- Number of threads
- Total message sizes/ time duration
- Packet size per descriptor

- Write/Read
- Completion reporting method
- Number of channels

It runs multiple threads for accessing the DMA channel. It also has performance measure capability.

8.2. Software Flow

Figure 21. Multi Channel DMA for PCI Express IP Software Operation Flow



Step 1

- Application creates the thread based on the required port of a channel
- After spawning the thread, the thread will try to acquire the available channel's port. In case if all channels ports are busy thread will wait in poll mode
- In the context of thread, libqdma will allocate descriptors buffer memory in the host
- Libqdma will initialize following register which includes Starting address of descriptors, queue size, write back address for Consumed Head and then enables the channels

QCSR registers:

- Q_START_ADDR_L (Offset 8'h08)
- Q_START_ADDR_H (Offset 8'h0C)
- Q_SIZE (Offset 8'h10)
- Q_CONSUMED_HEAD_ADDR_L (Offset 8'h20)

- Q_CONSUMED_HEAD_ADDR_H (Offset 8'h24)
- Q_BATCH_DELAY (Offset 8'h28)
- Q_CTRL (Offset 8'h00)

GCSR register:

- WB_INTR_DELAY (Offset 8'h08)

Step 2

- Threads will continuously try to send/receive the data and library keeps checking if channel is busy or descriptor ring is full
- If channel is not busy and descriptor ring is not full it goes to step 3. If channel is busy or descriptors ring is full thread will retry to initiate the transfer again

Descriptor ring full is identified by checking the Consumed Head and Tail pointer registers.
Channel busy is identified based on H2D_TPTR_AVL and D2H_TPTR_AVL register which has information on available space for number of tail pointer update in H2D and D2H ports.

Step 3

Thread requests for new descriptor to submit the request and updates the required field i.e. descriptor index, SOF, EOF, Payload, MSI-X enable and writeback enable.

Step 4

After initializing descriptor ring buffer, the libqdma will write number of descriptor updates into tail register of QCSR region. On every descriptor update the tail pointer is increased by 1.

QCSR tail pointer register: Q_TAIL_POINTER (Offset 8'h14)

Step 5

- Once the tail pointer write happens, Multi Channel DMA for PCI Express IP will fetch descriptors from host memory starting from the programmed Q_START_ADDR_L/H address
- Multi Channel DMA for PCI Express IP parses the descriptor content to find the sources, destination addresses and length of the data from descriptor and starts DMA operation

Step 6

Once descriptor processing is completed, IP will notify the completion status based on following methods, which can be enabled in each descriptor.

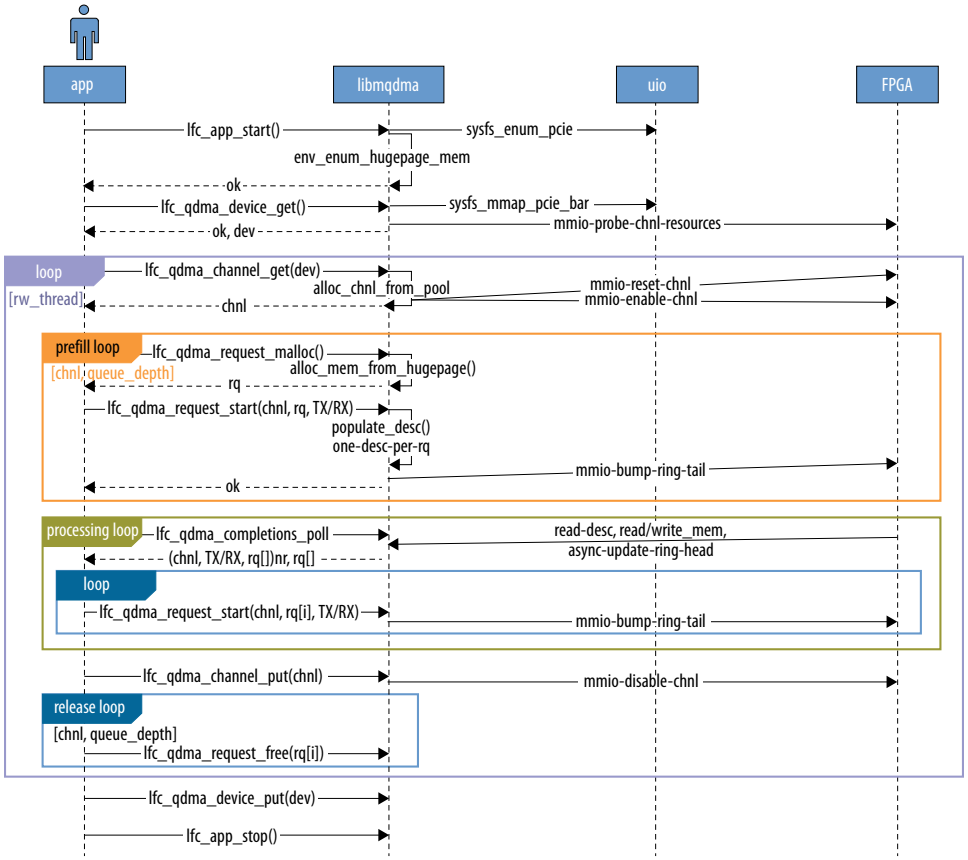
- Either based on MSI-X Interrupt : Multi Channel DMA for PCI Express IP sends MSI-X interrupt to host if enabled in Q_CTRL.
- Writeback: Multi Channel DMA for PCI Express IP updates Q_CONSUMED_HEAD_ADDR_L/H, if writeback is enabled in Q_CTRL.

8.3. API Flow

8.3.1. Single Descriptor Load and Submit

The API flow below shows loading one descriptor in the descriptor ring buffer and then submit DMA transfer by updating the tail pointer register by increment of 1.

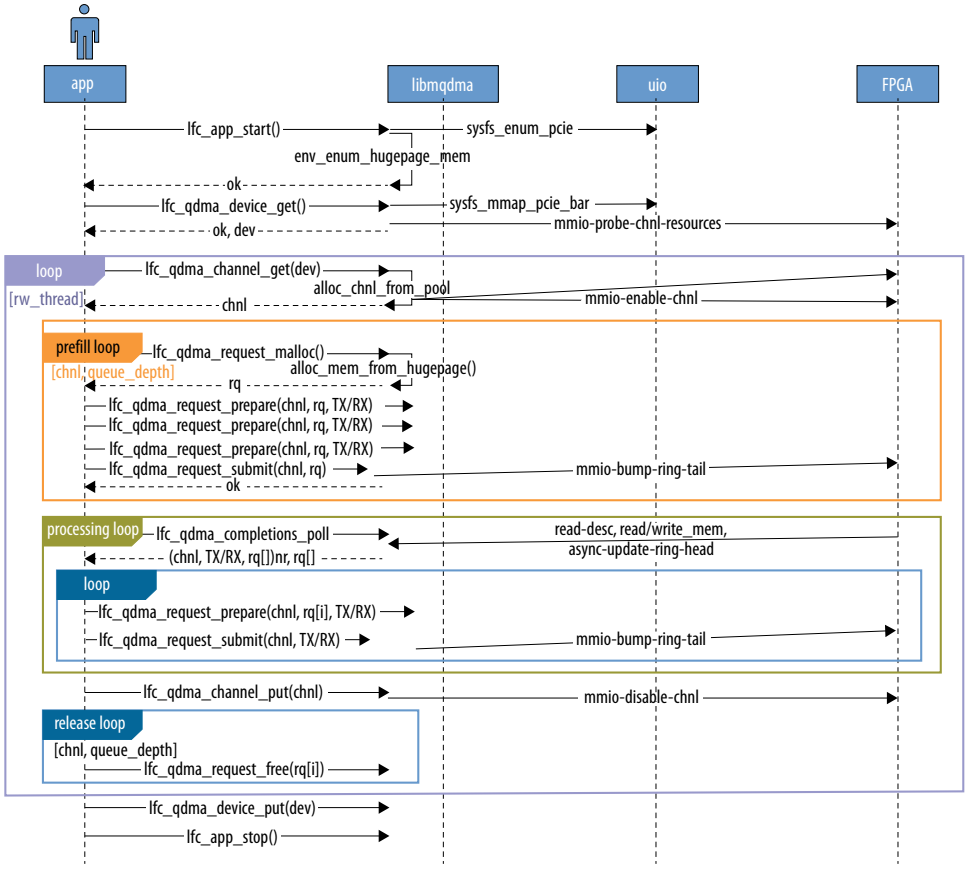
Figure 22. Single Descriptor Load and Submit



8.3.2. Multiple Descriptor Load and Submit

The API flow below shows loading the descriptors in bunch in the descriptor ring buffer and then submit for DMA transfer by updating the tail pointer register with total loaded descriptors.

Figure 23. Multiple Descriptor Load and Submit



8.3.3. `libmqdma` Library API List

This section describes the list of APIs which would be exposed to the application.

8.3.3.1. ifc_api_start

Table 50. ifc_api_start

API	API Description	Input Parameters	Return Values
`void ifc_app_start(void)`	This function is called at the time of application initialization. Probe and prepare the application for DMA transactions. • maps the enabled device memory to user space • memory allocation from huge page file system hugepagefs allows user space to get continuous and unswapable memory, which you can use for DMA operations.	void	0 on success negative otherwise

API	API Description	Input Parameters	Return Values
	Set default huge page size as 1 GB at boot time and allocate the required memory from huge pages.		

8.3.3.2. ifc_qdma_device_get

Table 51. ifc_qdma_device_get

API	API Description	Input Parameters	Return Values
`int ifc_qdma_device_get(const char *bdf, struct ifc_qdma_device **qdev)`	Based on the BDF, API returnS corresponding device context to the application. Application must maintain the device context and use it for further operations. When the application is done with I/O, it releases the context by using `ifc_qdma_device_put` API.	bdf - BDF qdev - Address of the pointer to device context	Updates device context and returns 0 on success negative otherwise

8.3.3.3. ifc_num_channels_get

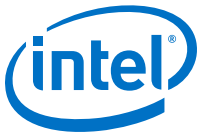
Table 52. ifc_num_channels_get

API	API Description	Input Parameters	Return Values
`int ifc_num_channels_get(struct ifc_qdma_device *qdev);`	This API returns the total number of channels supported by QDMA device	qdev - Pointer to device context	Number of channels supported

8.3.3.4. ifc_qdma_channel_get

Table 53. ifc_qdma_channel_get

API	API Description	Input Parameters	Return Values
`int ifc_qdma_channel_get(struct ifc_qdma_device *qdev, struct ifc_qdma_channel **chnl, int chno)`	Before submitting the DMA transactions, application is responsible to acquire the channel and pass the context on further interactions with framework. This API performs following: • Get the next available channel • Initialize the descriptors and data memory for both TX and RX queues • Enable the channel The last parameter in this API is the channel number. In case if you pass this parameter as -1, it will return available free channel. Otherwise, it allocates the available free channel.	qdev: QDMA device chnl: Pointer to update channel context chno: Channel no if user wants specific channel. -1 if no specific	0 : on success and populates channel context -1 : No channel is ready to be used. Channel context would be returned as NULL. -2 : Requested channel is already allocated. But valid channel context would be returned. Application may use this channel context.



8.3.3.5. ifc_qdma_device_put

Table 54. ifc_qdma_device_put

API	API Description	Input Parameters	Return Values
`void ifc_qdma_device_put(struct ifc_qdma_device *qdev)`	This API performs the unamapping of device memory and release the allocated resources	qdev: QDMA device	0 on success negative otherwise

8.3.3.6. ifc_qdma_channel_put

Table 55. ifc_qdma_channel_put

API	API Description	Input Parameters	Return Values
`void ifc_qdma_channel_put(struct ifc_qdma_channel *qchnl)`	Once the DMA transactions are completed, application must call this API to release the acquired channel so that, other process or other thread will acquire again. libmqdma disables this channel so that hardware will not look for DMA transactions.	qchan: channel context	0 on success negative otherwises

8.3.3.7. ifc_qdma_completion_poll

Table 56. ifc_qdma_completion_poll

API	API Description	Input Parameters	Return Values
`int ifc_qdma_completion_poll (struct ifc_qdma_channel *qchnl, int direction, void *pkt, int quota)`	Check if any previously queued and pending request got completed. If completed, the number of completed transactions would be returned to called so that the application submits those transactions.	qchnl: channel context dir: DMA direction, one of IFC_QDMA_DIRECTION_* pkts: address where completed requests to be copied quota: maximum number of requests to search	0 on success negative otherwise

8.3.3.8. ifc_qdma_request_start

Table 57. ifc_qdma_request_start

API	API Description	Input Parameters	Return Values
`int ifc_qdma_request_start(struct ifc_qdma_channel *qchnl, int dir, struct ifc_qdma_request *r);`	Depending on data direction, application selects TX/RX queue, populates the descriptors based on the passed request object and submits the DMA transactions. This is not blocking request. You may need to poll for the completion status.	qchnl: channel context received on ifc_qchannel_get() dir: DMA direction, one of IFC_QDMA_DIRECTION_* r: request struct that needs to be processed	0 on success negative otherwise

8.3.3.9. ifc_qdma_request_prepare

Table 58. ifc_qdma_request_prepare

API	API Description	Input Parameters	Return Values
`int ifc_qdma_request_prepare(struct ifc_qdma_channel *qchnl, int dir, struct ifc_qdma_request *r);`	Depending on the direction, application selects the queue and prepares the descriptor but not submit the transactions. Application must use ifc_qdma_request_submit API to submit the transactions to DMA engine.	qchnl: channel context received on ifc_qchannel_get() dir: DMA direction, one of IFC_QDMA_DIRECTION_* r: request struct that needs to be processed	Returns the number of transactions completed. negative otherwise

8.3.3.10. ifc_qdma_request_submit

Table 59. ifc_qdma_request_submit

API	API Description	Input Parameters	Return Values
`int ifc_qdma_request_submit(struct ifc_qdma_channel *qchnl, int dir);`	Submits all prepared and pending DMA transactions to MCDMA engine. Before calling this API, Application may need to call ifc_qdma_request_prepare to prepare the transactions. If you want to give priority to one channel and submit more number of transactions at a time from one channel, application may need to call multiple times and then submit APIs at a time to submit all the transactions.	qchnl: channel context received on ifc_qchannel_get() dir: DMA direction, one of IFC_QDMA_DIRECTION_*	0 on success negative otherwise

8.3.3.11. ifc_qdma_pio_read32

Table 60. ifc_qdma_pio_read32

API	API Description	Input Parameters	Return Values
`uint32_t ifc_qdma_pio_read32(struct ifc_qdma_device *qdev, uint64_t addr)`	Read the value from BAR2 address This API is used for PIO testing, dumping statistics, and pattern generation.	qdev: QDMA device addr: adderss to read	0 on success negative otherwise

8.3.3.12. ifc_qdma_pio_write32

Table 61. ifc_qdma_pio_write32

API	API Description	Input Parameters	Return Values
`void ifc_qdma_pio_write32(struct ifc_qdma_device *qdev, uint64_t addr, uint32_t val)`	Writes the value to BAR2 address	qdev: QDMA device addr: address to write val: value to write	0 on success and populates channel context negative otherwise

8.3.3.13. ifc_request_malloc

Table 62. ifc_request_malloc

API	API Description	Input Parameters	Return Values
`struct ifc_qdma_request *ifc_request_malloc(size_t len)`	libmqdma allocates the buffer for I/O request. The returned buffer would be DMA-able and allocated from huge pages	len - size of data buffer for I/O request	0 on success Negative otherwises

8.3.3.14. ifc_request_free

Table 63. ifc_request_free

API	API Description	Input Parameters	Return Values
`void ifc_request_free(void *req);`	Release the passed buffer and add in free pool	req - start address of allocation buffer	0 : on success and populates channel context -1 : channel not available -2 : Requested for specific channel. But already occupied

8.3.3.15. ifc_qdma_channel_set

Table 64. ifc_qdma_channel_set

API	API Description	Input Parameters	Return Values
`int ifc_qdma_channel_set(struct ifc_qdma_channel *chnl, int dir, enum ifc_qdma_queue_param parasssm_type, int val);`	set channel parameters	chnl: location to store channel address dir: direction (0 – RX, 1 – TX) type: parameter what we need to set val: value we need to set	0 : on success and populates channel context -1 : channel not available -2 : Requested for specific channel. But already occupied

8.3.3.16. ifc_app_stop

Table 65. ifc_app_stop

API	API Description	Input Parameters	Return Values
`void ifc_app_stop(void)`	ifc_app_stop unmaps the mapped resources and allocated memory	void	void

8.3.3.17. ifc_qdma_poll_init

Table 66. ifc_qdma_poll_init

API	API Description	Input Parameters	Return Values
`int ifc_qdma_poll_init(struct ifc_qdma_device *qdev)`	This will reset the poll eventfds Application, need to pass this fd_set to poll in case if MSI-X interrupts enabled.	qdev: QDMA device	0 on success Negative otherwise

8.3.3.18. ifc_qdma_poll_add

Table 67. ifc_qdma_poll_add

API	API Description	Input Parameters	Return Values
`int ifc_qdma_poll_add(struct ifc_qdma_device *qdev, ifc_qdma_channel *chnl, int dir)`	Append event fds to the poll list	qdev: QDMA device chnl: channel context dir: direction, which needs to poll	0 on success Negative otherwise

8.3.3.19. ifc_qdma_poll_wait

Table 68. ifc_qdma_poll_wait

API	API Description	Input Parameters	Return Values
`int ifc_qdma_poll_wait(struct ifc_qdma_device *qdev, ifc_qdma_channel **chnl, int *dir)`	Monitor for interrupts for all added queues. In case if any interrupt comes, it will return. Timeout: 1 msec	qdev: QDMA device qchan: address of channel context dir: address of direction parameters	0 on success and updates channel context and direction Negative otherwise

9. Revision History

Table 69. Revision History for Multi Channel DMA for PCI Express IP User Guide

Date	Intel Quartus Prime Version	IP Version	Changes
2020.07.20	20.2	20.0.0	Initial Release