IP CORE MANUAL

AXI4–Stream to PCI Express Direct Memory Access (DMA) 512–Bit IP

px_dma_ppkt2pcie_512

PENTEK

Pentek, Inc.
One Park Way
Upper Saddle River, NJ 07458
(201) 818–5900
http://www.pentek.com/

Manual Part Number: 807.48438 Rev: 1.0 – October 7, 2019

Manual Revision History

Date	Version	Comments
10/7/19	1.0	Initial Release

Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Pentek products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Pentek hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON–INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Pentek shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in conjunction with, the Materials (including your use of Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage and loss was reasonably foreseeable or Pentek had been advised of the possibility of the same. Pentek assumes no obligation to correct any error contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the materials without prior written consent. Certain products are subject to the terms and conditions of Pentek's limited warranty, please refer to Pentek's Ordering and Warranty information which can be viewed at http://www.pentek.com/contact/customerinfo.cfm; IP cores may be subject to warranty and support terms contained in a license issued to you by Pentek. Pentek products are not designed or intended to be fail–safe or for use in any application requiring fail–safe performance; you assume sole risk and liability for the use of Pentek products in such critical applications.

Copyright

Trademarks

Pentek, Jade, and Navigator are trademarks or registered trademarks of Pentek, Inc.

ARM and AMBA are registered trademarks of ARM Limited. PCI, PCI Express, PCIe, and PCI–SIG are trademarks or registered trademarks of PCI–SIG. Xilinx, Kintex UltraScale, Vivado, and Platform Cable USB are registered trademarks of Xilinx Inc., of San Jose, CA.

Printed in the United States of America.

Table of Contents

Page

Table of Contents

Table of Contents

Page

Chapter 4: Register Space

Table of Contents

IP Facts

Description

Pentek's Navigator™ **AXI4–Stream to PCI Express® (PCIe®) Direct Memory Access (DMA)** 512–Bit IP Core is a DMA engine that facilitates high–bandwidth, flexible movement of data from an AXI4–Stream source to the Xilinx® Gen3 Integrated Block for PCI Express IP Core. This core operates using a linked–list methodology to write the incoming data into the memory of the PCIe host.

This core complies with the ARM® AMBA® *AXI4 Specification* and also provides a control/status register interface. This manual defines the hardware interface, software interface, and parameterization options for the AXI4–Stream to PCIe DMA 512–Bit Core.

Features

- Fully AXI4–compliant interfaces
- Linked list DMA operation with up to 1024 link descriptors
- Supports transfer of data and metadata to the Xilinx PCIe Core
- Supports DMA transfer length control from metadata information
- Includes a unique Loop Increment mode which allows for the reduction of required descriptors and interrupt frequency
- Supports early DMA termination (optional) when an end of packet is reached in the input AXI4–Stream

Table 1–1: IP Facts Table

Core Specifics	
Supported Design Family[a]	Ultrascale+
Supported User Interfaces	AXI4–Lite and AXI4–Stream
Resources	See Table 2–1
Provided with the Core	
Design Files	VHDL
Example Design	Not Provided
Test Bench	VHDL
Constraints File	Not Provided[b]
Simulation Model	VHDL
Supported S/W Driver	HAL Software Support
Tested Design Flows	
Design Entry	Vivado® Design Suite 2019.1 or later
Simulation	Vivado VSim
Synthesis	Vivado Synthesis
Support	
Provided by Pentek fpgasupport@pentek.com	

a.For a complete list of supported devices, see the *Vivado Design Suite Release Notes*.
b.Clock constraints can be applied at the top level module of the user design.

This page is intentionally blank

Chapter 1: Overview

1.1 Functional Description

The AXI4–Stream to PCIe DMA 512–Bit Core is a high–bandwidth, reliable DMA engine used to move data from a framed AXI4–Stream source such as an Analog to Digital Converter, to the Xilinx PCIe Core. The DMA core operates using a linked–list methodology which provides the flexibility and ability to create complex data move–ment scenarios.

This DMA core also transmits sideband metadata which includes information about the data being moved such as timestamp, transfer length, and start and end of data acqui–sition markers. This core has the ability to auto–increment and loop through a number of data frames with one DMA link descriptor, using AXI4–Stream end–of–packet markers to govern the DMA transfer lengths.

This core accepts input AXI4–Streams in the **Packetized Sample Data/ Timestamp/ Information (PPKT) AXI4–Stream Format** (see Section 3.2 for more details) and gen–erates PCIe write requests which are transferred through the output **PCIe Requester Request Interface**. This core stores link descriptors in an internal RAM. Up to 1024 link descriptors can be configured in the RAM. This **Link Descriptor RAM** can be accessed and set up by an AXI Master in the user design through an **AXI4–Lite Interface**.

An **AXI4–Lite Control/Status Register (CSR) Bus** accesses the control/ status regis–ters within the **Register Space** of the core as shown in Figure 1–1. This core also includes a **Linked List State Machine** which includes the required logic to generate the buffer write requests based on the linked list descriptors defined by the user.

1.1 Functional Description (continued)

Figure 1–1: AXI4–Stream to PCIe DMA 512–Bit Core Block Diagram

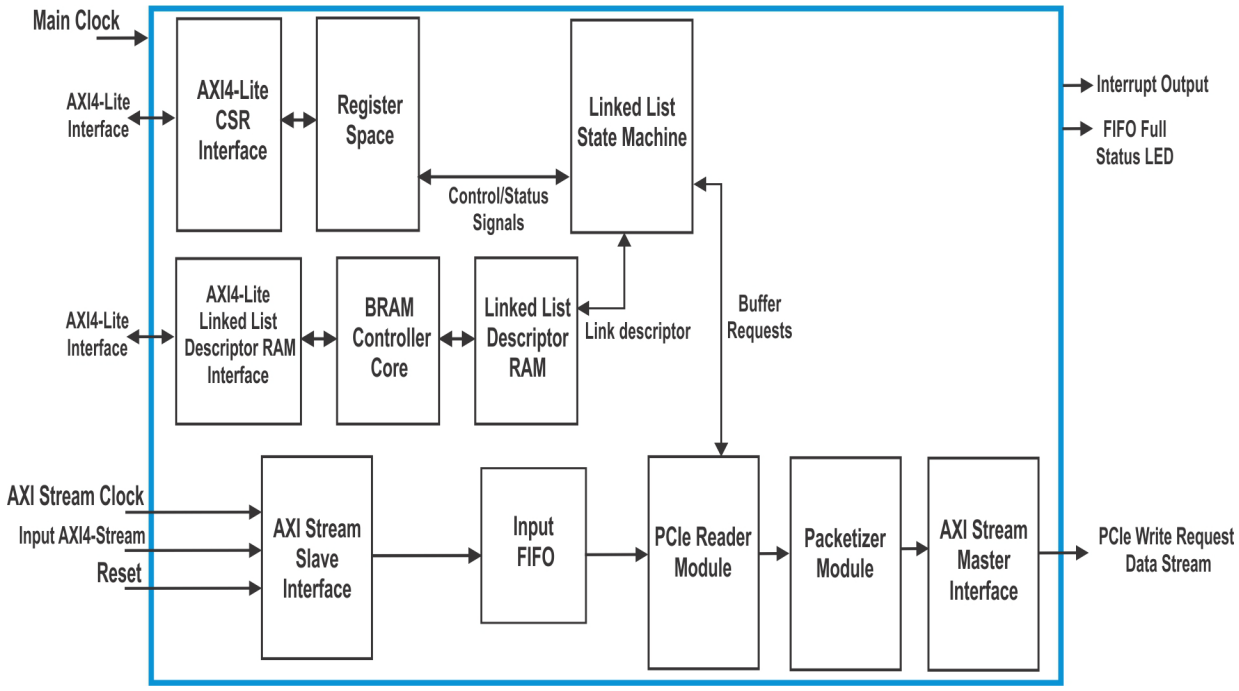


Figure 1–1 is a top–level block diagram of the AXI4–Stream to PCIe DMA 512–Bit Core. The modules within the block diagram are described below.

- **AXI4–Lite Control/Status Register (CSR) Interface:** This module implements a 32–bit **AXI4–Lite CSR Slave Interface** to access the control/status registers within the **Register Space** of this core. For additional details about the AXI4–Lite CSR Interface, refer to Section 3.1, AXI4–Lite Core Interfaces.

- **Register Space:** This module contains control and status registers including Interrupt Enable, Interrupt Flag, and Interrupt Status registers. Registers are accessed through the AXI4–Lite Interface.

1.1 Functional Description (continued)

- **AXI4−Lite Linked List Descriptor RAM Interface:** This interface is used to connect to the AXI4−Lite linked list descriptor bus which is used to access the **Linked List Descriptor RAM** within the core. For additional details about the AXI4−Lite Interface, refer to Section 3.1 AXI4−Lite Core Interfaces.

- **AXI4−Lite BRAM Controller Core:** This is an AXI4−Lite **BRAM Controller Core** which is connected to the **AXI4−Lite Linked List Descriptor RAM Interface** to communicate with the Linked−List RAM within the AXI4−Stream to PCIe DMA 512−Bit Core.

- **Linked List Descriptor RAM:** This is a Xilinx Dual Port Block RAM generated to store up to 1024 link descriptors.

- **Linked List State Machine:** The **Linked List State Machine** is used to generate buffer write requests to the **PCIe Reader** module based on the link descriptors defined in the **Linked List Descriptor RAM**.

- **AXI4−Stream Interfaces:** The AXI4−Stream to PCIe DMA 512−Bit Core has two AXI4−Stream Interfaces to receive packetized sample data/ timestamp/ data information AXI4−Streams, and to transfer **PCIe Write Request Data Streams**. For more details about the AXI4−Stream Interfaces please refer to **Section 3.2 AXI4−Stream Core Interfaces**.

- **Input FIFO:** This is an AXI4−Stream **Input FIFO** generated to store the input packed AXI data streams. The user can define the size of input FIFO to be generated based on the size of AXI4−Stream input data to be written to the PCIe core. The FIFO size is fixed at 32KB/

- **PCIe Reader Module:** The **PCIe Reader** module reads the input FIFO for the AXI4−Stream data and generates the required PCIe write request payload and header data from the buffer write requests received from the **Linked List State Machine**.

- **Packetizer Module:** This module packetizes the generated payload and header data into PCIe Requester Request AXI4−Streams which are compatible with the Requester Request Interface of the Xilinx PCIe Core. The generated PCIe Requester Request AXI4−Streams follow the standard AXI4−Stream format and must be connected to the Xilinx PCIe Core through a Pentek PCIe Requester Interface Gasket Core in order to convert the format of the **`tready`** and **`tkeep`** AXI4−Stream signals into a format compatible with the Requester Request Interface Bus of the Xilinx PCIe Core.

1.2 Applications

The AXI4−Stream to PCIe DMA 512−Bit Core can be incorporated into an Ultrascale+ FPGA to perform DMA transfers of data from an AXI4−Stream source to the Xilinx PCIe Core.

1.3 System Requirements

For a list of system requirements, see the Vivado Design Suite Release Notes.

1.4 Licensing and Ordering Information

This core is included with all Pentek Navigator FPGA Design Kits for Pentek Jade series board products. Contact Pentek for licensing and ordering information (www.pentek.com).

1.5 Contacting Technical Support

Technical Support for Pentek's Navigator FPGA Design Kits is available via e−mail (fpgasupport@pentek.com) or by phone (201−818−5900 ext. 238, 9 am to 5 pm EST).

1.6 Documentation

This user manual is the main document for this IP core. The following documents provide supplemental material:

1) *Vivado Design Suite User Guide: Designing with IP*

2) *Vivado Design Suite User Guide: Programming and Debugging*

3) *ARM AMBA AXI4 Protocol Version 2.0 Specification*
http://www.arm.com/products/system−ip/amba−specifications.php

4) *Xilinx Gen3 Integrated Block for PCI Express Product Guide*

Chapter 2: General Product Specifications

2.1 Standards

The AXI4−Stream to PCIe DMA 512−Bit Core has bus interfaces that comply with the A*RM AMBA AXI4−Lite Protocol Specification* and the *AMBA AXI4−Stream Protocol Specification*.

2.2 Performance

The performance of the AXI4−Stream to PCIe DMA 512−Bit Core is limited by the FPGA logic speed. The values presented in this section should be used as an estimation guideline. Actual performance can vary.

2.2.1 Maximum Frequencies

The AXI4−Stream to PCIe DMA 512−Bit Core has two incoming clock signals. The input Main clock (**`aclk`**) and the input AXI4−Stream Clock (**`s_axis_ppkt_aclk`**) have maximum frequencies of 250 MHz, on an Ultrascale+ −1 speed grade FPGA. 250 MHz is typically the PCIe AXI bus clock frequency.

2.2.2 Throughput

The throughput of the core can vary widely depending on many factors including the operating mode, DMA length, and AXI4−Stream input data bus width.

The number of DMAs in the user application will also significantly affect the maximum throughput of each DMA since an AXI4−Stream Switch Core must be used to arbitrate which DMA has access to the PCIe core. The length of DMA access and the mode options of the DMA engine can affect throughput. The longer the DMA access, the less significant is the time the DMA core takes to access the link descriptor and start operation.

2.3 Resource Utilization

The resource utilization of the AXI4–Stream to PCIe DMA 512–Bit Core is shown in Table 2–1. Resources have been estimated for the Ultrascale+ –1 speed grade device. These values were generated using the Vivado Design Suite.

Table 2–1: Resource Usage and Availability

Resource	# Used
LUTs	3,531
Flip–Flops	7,487
BRAMs	34.5

NOTE: Actual utilization may vary based on the user design in which the AXI4–Stream to PCIe DMA 512–Bit Core is incorporated.

2.4 Limitations and Unsupported Features

- This core does not support DMA transfer length less than a dword (4 bytes).
- This core does not have 4K boundary cross checking.

2.5 Generic Parameters

The generic parameters of the AXI4–Stream to PCIe DMA 512–Bit Core are described in Table 2–2. These parameters can be set as required by the user application while customizing the core.

Table 2-2: Generic Parameters

Port/Signal Name	Type	Description
has_fifo_full_led	Boolean	**Has Input FIFO Full LED:** When set to True, this parameter indicates that the AXI4–Stream to PCIe DMA 512–Bit Core has a FIFO status LED output indicating the Input FIFO Full Status.

Chapter 3: Port Descriptions

This chapter provides details about the port descriptions for the following interface types:

- AXI4–Lite Core Interfaces
- AXI4–Stream Core Interfaces

3.1 AXI4–Lite Core Interfaces

The AXI4–Stream to PCIe DMA 512–Bit Core has the following two AXI4–Lite core interfaces to control, and receive status from, the core.

- **Control/Status Register (CSR) Interface:** The interface through which all the control and status registers are accessed.
- **Linked List Descriptor RAM (DESCR) Interface:** The interface through which the link descriptors can be programmed and read back.

3.1.1 Control/Status Register (CSR) Interface

The CSR interface is an AXI4–Lite Slave Interface that can be used to access the control and status registers in the AXI4–Stream to PCIe DMA 512–Bit Core. Table 3–1 defines the ports in the CSR Interface. See Chapter 4 for a Control/Status Register memory map and bit definitions. See the *AMBA AXI4–Lite Specification* for more details on operation of the AXI4–Lite interfaces.

Table 3-1: Control/Status Register (CSR) Interface Port Descriptions

Port	Direction	Width	Description
aclk	Input	1	**Clock** (250 MHz)
s_axi_csr_aresetn	Input	1	**Reset:** Active low. This signal will reset all control registers to their initial states.
s_axi_csr_awaddr	Input	6	**Write Address:** Address used for write operations. It must be valid when **s_axi_csr_awvalid** is asserted and must be held until **s_axi_csr_awready** is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core. Note that the Register Space registers occupy an address range of [Base Address + (0x00 to 0x3C)].

Table 3-1: Control/Status Register (CSR) Interface Port Descriptions (Continued)

Port	Direction	Width	Description
s_axi_csr_awprot	Input	3	**Protection:** The AXI4–Stream to PCIe DMA 512–Bit Core ignores these bits.
s_axi_csr_awvalid	Input	1	**Write Address Valid:** This input must be asserted to indicate that a valid write address is available on **s_axi_csr_awaddr**. The AXI4–Stream to PCIe DMA 512–Bit Core asserts **s_axi_csr_awready** when it is ready to accept the address. The **s_axi_csr_awvalid** must remain asserted until the rising clock edge after the assertion of **s_axi_csr_awready**.
s_axi_csr_awready	Output	1	**Write Address Ready:** This output is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when it is ready to accept the write address.The address is latched when **s_axi_csr_awvalid** and **s_axi_csr_awready** are high on the same cycle.
s_axi_csr_wdata	Input	32	**Write Data:** This data will be written to the address specified by **s_axi_csr_awaddr** when **s_axi_csr_wvalid** and **s_axi_csr _wready** are both asserted. The value must be valid when **s_axi_csr_wvalid** is asserted and held until **s_axi_csr_wready** is also asserted.
s_axi_csr_wstrb	Input	4	**Write Strobes:** This signal, when asserted, indicates the number of bytes of valid data on the **s_axi_csr_wdata** signal. Each of these bits, when asserted, indicate that the corresponding byte of **s_axi_csr_wdata** contains valid data. Bit 0 corresponds to the least significant byte, and bit 3 to the most significant.
s_axi_csr_wvalid	Input	1	**Write Valid:** This signal must be asserted to indicate that the write data is valid for a write operation. The value on **s_axi_csr _wdata** is written into the register at address **s_axi_csr_awaddr** when **s_axi_csr_wready** and **s_axi_csr_wvalid** are high on the same cycle.
s_axi_csr_wready	Output	1	**Write Ready:** This signal is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when it is ready to accept data. The value on **s_axi_csr_wdata** is written into the register at address **s_axi_csr_awaddr** when **s_axi_csr_wready** and **s_axi_csr_wvalid** are high on the same cycle, assuming that the address has already or simultaneously been submitted.

Table 3-1: Control/Status Register (CSR) Interface Port Descriptions (Continued)			
Port	**Direction**	**Width**	**Description**
s_axi_csr_bresp	Output	2	**Write Response:** The AXI4–Stream to PCIe DMA 512–Bit Core indicates success or failure of a write transaction through this signal, which is valid when **s_axi_csr_bvalid** is asserted; 00 = Success of normal access 01 = Success of exclusive access 10 = Slave Error 11 = Decode Error Note: For more details about this signal refer to the *AMBA AXI Specification*.
s_axi_csr_bready	Input	1	**Write Response Ready:** This signal must be asserted by the user logic when it is ready to accept the Write Response.
s_axi_csr_bvalid	Output	1	**Write Response Valid:** This signal is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when the write operation is complete and the Write Response is valid. It is held until **s_axi_csr_bready** is asserted by the user logic.
s_axi_csr_araddr	Input	6	**Read Address:** Address used for read operations. It must be valid when **s_axi_csr_arvalid** is asserted and must be held until **s_axi_csr_arready** is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core.
s_axi_csr_arprot	Input	3	**Protection:** These bits are ignored by the AXI4–Stream to PCIe DMA 512–Bit Core.
s_axi_csr_arvalid	Input	1	**Read Address Valid:** This input must be asserted to indicate that a valid read address is available on **s_axi_csr_araddr**. The core asserts **s_axi_csr_arready** when it ready to accept the Read Address. This input must remain asserted until the rising clock edge after the assertion of **s_axi_csr_arready**.
s_axi_csr_arready	Output	1	**Read Address Ready:** This output is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when it is ready to accept the read address. The address is latched when **s_axi_csr_arvalid** and **s_axi_csr_arready** are high on the same cycle.
s_axi_csr_rdata	Output	32	**Read Data:** This value is the data read from the address specified by the **s_axi_csr_araddr** when **s_axi_csr_arvalid** and **s_axi_csr_arready** are high on the same cycle.

Table 3-1: Control/Status Register (CSR) Interface Port Descriptions (Continued)			
Port	**Direction**	**Width**	**Description**
s_axi_csr_rresp	Output	2	**Read Response:** The AXI4–Stream to PCIe DMA 512–Bit Core indicates success or failure of a read transaction through this signal, which is valid when **s_axi_csr_rvalid** is asserted; 00 = Success of normal access 01 = Success of exclusive access 10 = Slave Error 11 = Decode Error Note: For more details about this signal refer to the *AMBA AXI Specification*.
s_axi_csr_rvalid	Output	1	**Read Data Valid:** This signal is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when the read is complete and the read data is available on **s_axi_csr_rdata**. It is held until **s_axi_csr_rready** is asserted by the user logic.
s_axi_csr_rready	Input	1	**Read Data Ready:** This signal is asserted by the user logic when it is ready to accept the Read Data.
irq	Output	1	**Interrupt:** This is an active high, edge–type interrupt output.

3.1.2 Linked List Descriptor RAM (DESCR) Interface

Table 3–2 defines the ports in the DESCR Interface. This interface is an AXI4–Lite Slave Interface that can be used to access the Linked List Descriptor RAM in the AXI4–Stream to PCIe DMA 512–Bit Core. See Chapter 5 for the Descriptor RAM memory map and bit definitions. This interface is associated with **aclk**. See the *AMBA AXI4–Lite Specification* for more details on operation of the AXI4–Lite interfaces.

Table 3-2: Linked List Descriptor RAM (DESCR) Interface Port Descriptions			
Port	**Direction**	**Width**	**Description**
aclk	Input	1	**Clock** (250 MHz)
aresetn	Input	1	**Reset:** Active Low. This reset does not have any effect on the Descriptor RAM contents. To clear those contents, they must be written to individually.

Table 3-2: Linked List Descriptor RAM (DESCR) Interface Port Descriptions (Continued)

Port	Direction	Width	Description
s_axi_descr_awaddr	Input	16	**Write Address:** Address used for write operations. It must be valid when **s_axi_descr_awvalid** is asserted and must be held until **s_axi_descr_awready** is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core. Note that the DESCR interface RAM occupies an address range of [Base Address + (0x0000 to 0x1000)].
s_axi_descr_awprot	Input	3	**Protection:** The AXI4–Stream to PCIe DMA 512–Bit Core ignores these bits.
s_axi_descr_awvalid	Input	1	**Write Address Valid:** This input must be asserted to indicate that a valid write address is available on **s_axi_descr_awaddr**. The AXI4–Stream to PCIe DMA 512–Bit Core asserts **s_axi_descr_ awready** when it is ready to accept the address. The **s_axi_ descr_awvalid** must remain asserted until the rising clock edge after the assertion of **s_axi_descr_awready**.
s_axi_descr_awready	Output	1	**Write Address Ready:** This output is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when it is ready to accept the write address.The address is latched when **s_axi_descr_awvalid** and **s_axi_descr_awready** are high on the same cycle.
s_axi_descr_wdata	Input	32	**Write Data:** This data will be written to the address specified by **s_axi_descr_awaddr** when **s_axi_descr_wvalid** and **s_axi_descr_wready** are both asserted. The value must be valid when **s_axi_descr_wvalid** is asserted and held until **s_axi_descr_wready** is also asserted.
s_axi_descr_wstrb	Input	4	**Write Strobes:** This signal, when asserted, indicates the number of bytes of valid data on the **s_axi_descr_wdata** signal. Each of these bits, when asserted, indicate that the corresponding byte of **s_axi_descr_wdata** contains valid data. Bit 0 corresponds to the least significant byte, and bit 3 to the most significant.
s_axi_descr_wvalid	Input	1	**Write Valid:** This signal must be asserted to indicate that the write data is valid for a write operation. The value on **s_axi_descr_wdata** is written into the register at address **s_axi_descr_awaddr** when **s_axi_descr_wready** and **s_axi_descr_wvalid** are High on the same cycle.

Table 3-2: Linked List Descriptor RAM (DESCR) Interface Port Descriptions (Continued)			
Port	**Direction**	**Width**	**Description**
s_axi_descr_wready	Output	1	**Write Data Ready:** This signal is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when it is ready to accept data. The value on **s_axi_descr_wdata** is written into the register at address **s_axi_descr_awaddr** when **s_axi_descr_wready** and **s_axi_descr_wvalid** are high on the same cycle, assuming that the address has already or simultaneously been submitted.
s_axi_descr_bresp	Output	2	**Write Response:** The AXI4–Stream to PCIe DMA 512–Bit Core indicates success or failure of a write transaction through this signal, which is valid when **s_axi_descr_bvalid** is asserted; 00 = Success of normal access 01 = Success of exclusive access 10 = Slave Error 11 = Decode Error Note: For more details about this signal refer to the *AMBA AXI Specification*.
s_axi_descr_bready	Input	1	**Write Response Ready:** This signal must be asserted by the user logic when it is ready to accept the Write Response.
s_axi_descr_bvalid	Output	1	**Write Response Valid:** This signal is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when the write operation is complete and the Write Response is valid. It is held until **s_axi_descr_ bready** is asserted by the user logic.
s_axi_descr_araddr	Input	16	**Read Address:** Address used for read operations. It must be valid when **s_axi_descr_arvalid** is asserted and must be held until **s_axi_descr_arready** is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core.
s_axi_descr_arprot	Input	3	**Protection:** These bits are ignored by the AXI4–Stream to PCIe DMA 512–Bit Core.
s_axi_descr_arvalid	Input	1	**Read Address Valid:** This input must be asserted to indicate that a valid read address is available on **s_axi_descr_araddr**. The core asserts **s_axi_descr_arready** when it ready to accept the Read Address. This input must remain asserted until the rising clock edge after the assertion of **s_axi_descr_arready**.

Table 3-2: Linked List Descriptor RAM (DESCR) Interface Port Descriptions (Continued)			
Port	**Direction**	**Width**	**Description**
s_axi_descr_arready	Output	1	**Read Address Ready:** This output is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when it is ready to accept the read address. The address is latched when **s_axi_descr_arvalid** and **s_axi_descr_arready** are high on the same cycle.
s_axi_descr_rdata	Output	32	**Read Data:** This value is the data read from the address specified by the **s_axi_descr_araddr** when **s_axi_descr_ arvalid** and **s_axi_descr_arready** are high on the same cycle.
s_axi_descr_rresp	Output	2	**Read Response:** The AXI4–Stream to PCIe DMA 512–Bit Core indicates success or failure of a read transaction through this signal, which is valid when **s_axi_descr_rvalid** is asserted; 00 = Success of normal access 01 = Success of exclusive access 10 = Slave Error 11 = Decode Error Note: For more details about this signal refer to the *AMBA AXI Specification*.
s_axi_descr_rvalid	Output	1	**Read Data Valid:** This signal is asserted by the AXI4–Stream to PCIe DMA 512–Bit Core when the read is complete and the read data is available on **s_axi_descr_rdata**. It is held until **s_axi_descr_ rready** is asserted by the user logic.
s_axi_descr_rready	Input	1	**Read Data Ready:** This signal is asserted by the user logic when it is ready to accept the Read Data.

3.2 AXI4–Stream Core Interfaces

The AXI4–Stream to PCIe DMA 512–Bit Core has the following AXI4–Stream interfaces, used to receive and transfer data streams:

- **Packetized Sample Data/ Timestamp/ Information Stream (PPKT) Interface:** This is an AXI4–Stream Slave Interface of the core used to receive packed sample data/ timestamp/ data information AXI4–Streams.

- **PCIe Requester Request (PCIE_RQ) Interface:** This is the interface through which the AXI4–Stream to PCIe DMA 512–Bit Core transfers PCIe write requests. This interface is compatible with the Xilinx Gen3 Integrated Block for the PCIe core's Requester Request Interface in address–aligned mode.

- **PCIe Miscellaneous Control (CNTL) Interface:** This is the interface through which static control signals from the PCIe core are received.

3.2.1 Packetized Sample Data/ Timestamp/ Information Streams (PPKT) Interface

The Pentek Quartz and Jade series board products have packed AXI4–Streams that follow a Packetized Sample Data/ Timestamp/ Information Stream (PPKT) format. This type of data stream contains packed sample data streams as input to DMAs. The start of packet (SOP) and **`tlast`** signals are used to mark the start and end of gate acquisition data.

There is an AXI4–Stream Slave Interface across the input of the AXI4–Stream to PCIe DMA 512–Bit Core to receive AXI4–Streams in the PPKT format. Table 3–3 defines the ports in the AXI4–Stream Slave Packetized Sample Data/ Timestamp/ Information Stream Interface. See the *AMBA AXI4–Stream Specification* for more details on the operation of the AXI4–Stream Interface.

3.2 AXI4–Stream Core Interfaces (continued).

Table 3-3: Packetized Sample Data/ Timestamp/ Information Streams Interface Port Descriptions

Port	Direction	Width	Description
s_axis_ppkt_aclk	Input	1	**Input Data Stream Clock**
s_axis_ppkt_aresetn	Input		**Reset:** Active Low.
s_axis_ppkt_tdata	Input	512	**Input Data:** The input data bus width is fixed at 512 bits.
s_axis_ppkt_tvalid	Input	1	**Input Data Valid:** Asserted when data is valid on **s_axis_ppkt_tdata**.
s_axis_ppkt_tuser	Input	80	**Sideband Information:** This is the user defined sideband information received alongside the data stream. **tuser [63:0] – Timestamp[63:0]** **tuser [64] – Start of Packet (SOP)** **tuser [66:65] – Data Format => 0 = 8–bit; 1 = 16–bit; 2 = 24–bit; 3 = 32–bit** **tuser [67] – Data Type => 0 = Real; 1 = I/Q** **tuser [75:68] – channel [7:0]** **tuser [79:76] – user defined data[3:0] (Considered valid on tlast only)**
s_axis_ppkt_tkeep	Input	32	**Data Keep:** The **tkeep** signal indicates the valid data sample bytes on **s_axis_ppkt_tdata**. Each bit corresponds to a 16–bit word in **s_axis_ppkt_tdata** i.e., bit 0 corresponds to the least significant 16 bits and bit 1 to the most significant. When it is asserted, the data is considered valid. All **tkeep** bits must be '1' contiguously until the **tlast**.
s_axis_ppkt_tlast	Input	1	**Data Last:** When asserted, this marks the last data in the current data frame.
s_axis_ppkt_tready	Output		**Data Ready:** The AXI4–Stream to PCIe DMA 512–Bit Core asserts this signal when there is room in the input FIFO to accept more data. Often, in the user application the data is coming from a data source that cannot be held off such as an ADC. In this case, **tready** will be ignored and the input FIFO can overflow.

3.2 AXI4–Stream Core Interfaces (continued)

3.2.2 PCIe Requester Request (PCIE_RQ) Interface

Table 3–4 defines the ports in the PCIE_RQ Interface of the AXI4–Stream to PCIe DMA 512–Bit Core. This is an AXI4–Stream Master Interface that can be used to submit Requester write request packets to the Xilinx PCIe Core. The AXI4–Stream PCIe RQ Master Interface is compatible with the Xilinx PCIe Core's Requester Request Interface in 256–bit address–aligned configuration. See the Requester Request Interface Section of *Xilinx Gen3 Integrated Block for PCI Express Product Guide* for more details.

Table 3-4: PCIe Requester Request Interface Port Descriptions

Port	Direction	Width	Description
aclk	Input	1	**Clock: 250MHz**
aresetn	Input		**Reset:** Active Low.
m_axis_pcie_rq_tdata	Output	512	**Requester Request Data Bus:** This is the Requester request data from the DMA core to the Xilinx PCIe Core. It has a fixed width of 512 bits and is therefore only compatible with only the 512–bit wide version of the Xilinx PCIe Core which is only available on Ultrascale+ FPGAs. This data follows address–aligned format.
m_axis_pcie_rq_tlast	Output	1	**TLAST Indication for the Requester Request Data:** The AXI4–Stream to PCIe DMA 512–Bit Core asserts this signal in the last cycle of a data transfer to indicate the end of the packet.
m_axis_pcie_rq_tvalid	Output		**Requester Request Data Valid:** This core asserts this signal whenever it is driving valid data on the **m_axis_ pcie_rq_tdata** signal and keeps it asserted during the transfer of a packet. The Xilinx PCIe Core paces the data transfer using the **m_axis_pcie_rq_tready** signal.
m_axis_pcie_rq_tuser	Output	137	**Requester Request User Data:** This signal contains the sideband information for the TLP being transferred. This signal is valid when **m_axis_pcie_rq_tvalid** is High. **Table 3–5** defines the bit definitions of **m_axis_pcie_rq_ tuser[59:0].**

Table 3-4: PCIe Requester Request Interface Port Descriptions (Continued)

Port	Direction	Width	Description
m_axis_pcie_rq_tkeep	Output	64	**TKEEP Indication for the Requester Request Data:** The assertion of bit *i* of this bus during a transfer indicates that dword *i* (in this case a dword is 8 bits) of the **m_axis_pcie_rq _tdata** bus contains valid data. This bit is set to 1 contiguously for all dwords, starting from the first dword of the descriptor to the last dword of the payload. Thus, **m_axis_pcie_rq_tkeep** is set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of data bus.
m_axis_pcie_rq_tready	Input	1	**Requester Request Ready:** This signal is asserted by the Xilinx PCIe Core to indicate that it is ready to accept data from the AXI4–Stream to PCIe DMA 512–Bit Core. Data is transferred across this interface when both **m_axis_ pcie_rq_tready** and **m_axis_pcie_rq _tvalid** are High on the same cycle. If the Xilinx PCIe core deasserts the ready signal when **m_axis _pcie_rq_tvalid** is High, the DMA core maintains the data on the bus and keeps the valid signal asserted until the PCIe core has asserted the ready signal. The standard 1–bit **tready** signal from the DMA core is converted by the Pentek PCIe Requester Interface Gasket core and transferred to the Xilinx PCIe core in the format compatible by it's Requester Request Interface.

3.2 AXI4–Stream Core Interfaces (continued)

3.2.2 PCIe Requester Request (PCIE_RQ) Interface (continued)

Table 3–5 shows the bit definitions of the PCIe Requester Request Interface user data (**m_axis_pcie_rq_tuser**).

Table 3–5: PCIe Requester Request Interface User Data Bit Definitions

Bit Index	Name	Width	Description
59:28	parity	32	**Parity:** Parity is not supported by the AXI4–Stream to PCIe DMA 512–Bit Core. The Xilinx PCIe Core must be configured to disable parity checking. These bits are always set to 0x000000.
27:24	seq_num	4	**Sequence Number:** Not supported. Set to 0x0.
23:16	tph_st_tag	8	**Transaction Processing Hint:** This is not supported by this core.
15	tph_indirect_ tag_en	1	
14:13	tph_type	2	
12	tph_present	1	
11	discontinue		**Discontinue:** This signal is asserted during a transfer if a DMA abort has been requested. The Xilinx PCIe Core nullifies the corresponding TLP on the PCIe link to avoid data corruption.
10:8	addr_offset	3	**Address Offset:** The AXI4–Stream to PCIe DMA 512–Bit Core outputs the dword number where the payload data begins on the data bus through these address offset bits. This enables the Xilinx PCIe Core to determine the alignment of the data block being transferred. The Xilinx PCIe Core samples this field in the first beat of a packet when **m_axis_pcie _rq_tvalid** and **m_axis_pcie_rq_tready** are both asserted.
7:4	last_be	4	**Byte Enables for the Last dword:** The AXI4–Stream to PCIe DMA 512–Bit Core only supports data aligned to dword boundaries. Therefore, these four bits always output 0xF.
3:0	first_be		**Byte Enables for the First dword:** The AXI4–Stream to PCIe DMA 512–Bit Core only supports data aligned to dword boundaries. Therefore, these four bits always output 0xF.

3.2 AXI4–Stream Core Interfaces (continued)

3.2.2 PCIe Requester Request (PCIE_RQ) Interface (continued)

When meta data packets are written to the Xilinx PCIe Core by setting the **`write_meta_data`** bit to '1' in the Control Word of the Linked list Descriptor (see Table 5–3), the output data packet is 128 bits wide with the following bit definitions.

Table 3-6: Meta Data Packet Bit Definitions

Bit Index	Name	Width	Description
127:126	**reserved**	2	**Reserved**
125	**last_is_eop**	1	**End of Packet:** This bit indicates the end of the packet. Active High.
124	**first_is_sop**		**Start of Packet:** This bit indicates the start of the packet. Active High.
123	**type**		**Type of First Sample:** This bit indicates the type of the first sample of data. 0 = I 1 = Q
122	**data_type**		**Data Type:** 0 = Real 1 = I/Q
121:120	**data_format**	2	**Data Format:** 00 = 8–bit 01 = 16–bit 10 = 24–bit 11 = 32–bit
119:112	**chan_num**	8	**Channel Number:** These bits indicate the channel number of the ADC channel in the user design the data is being transferred from.
111:100	**counter**	12	**Meta Data Packet Counter:** These bits indicate the value of a counter implemented to increment each time a meta data packet is generated. The counter resets to 0x000 when the DMA resets or a DMA abort is initiated.
99:96	**user_bits**	4	**User–defined Bits**
95:64	**valid_bytes**	32	**Number of Valid Bytes:** These bits indicate the number of valid bytes being transferred in the data packet.
63:0	**timestamp**	64	**Timestamp:** These bits indicate the timestamp of the data. The timestamp information is received in the sideband user data of the AXI4–Stream Slave Interface.

3.2 AXI4–Stream Core Interfaces (continued)

3.2.3 PCIe Miscellaneous Control (CNTL) Interface

This interface is used by the AXI4–Stream to PCIe DMA 512–Bit Core to receive PCIe link status data and byte swap data from the user design. **Table 3–7** defines the ports in the **CNTL** interface of the core. This interface is an AXI4–Stream Master Interface and is associated with **aclk** signal.

Table 3–7: PCIe Miscellaneous Control Interface Port Descriptions

Port	Direction	Width	Description
s_axis_cntl_tdata	Input	8	**PCIe DMA Control Data Bus:** This contains the information about the byte swap, PCIe link maximum payload size and maximum read request size. It has a fixed width of 8 bits. **s_axis_cntl_tdata[2:0]** – Maximum PCIe Packet Payload Size 000 – 128 Bytes maximum packet payload size 001 – 256 Bytes maximum packet payload size 010 – 512 Bytes maximum packet payload size 011 – 1024 Bytes maximum packet payload size 100 – 2048 Bytes maximum packet payload size 101 – 4096 Bytes maximum packet payload size **s_axis_cntl_tdata[6:4]** – Maximum PCIe Read Request Size 000 – 128 Bytes maximum read request size 001 – 256 Bytes maximum read request size 010 – 512 Bytes maximum read request size 011 – 1024 Bytes maximum read request size 100 – 2048 Bytes maximum read request size 101 – 4096 Bytes maximum read request size **s_axis_cntl_tdata[7]** – Byte Swap 0 – Not Swapped 1 – Swapped
s_axis_cntl_tvalid		1	**PCIe DMA Control Data Valid:** The core asserts this signal to indicate valid data on the **s_axis_cntl_tdata** signal. This signal is always High.

3.3 I/O Signals

The I/O port/signal descriptions of the top level module of the AXI4–Stream to PCIe DMA 512–Bit Core are discussed in Table 3–8.

Table 3–8: I/O Signals

Port/Signal Name	Type	Direction	Description
fifo_full_led	std_logic	Output	**Input FIFO Full Status Output:** Active High. This output indicates whether the input FIFO of the AXI4–Stream to PCIe DMA 512–Bit Core is full. Typically, this output is connected to a status LED to indicate the FIFO status.
fifo_rst_out_n	std_logic	Output	**Input FIFO Reset Output:** This output is the Input FIFO Flush signal from the FIFO Flush Register. See **Section 4.5**.

This page is intentionally blank

Chapter 4: Register Space

This chapter provides the memory map and register descriptions for the register space of the AXI4–Stream to PCIe DMA 512–Bit Core. The memory map is provided in Table 4–1.

NOTE: There are two seperate memory map addresses for this IP Core: one for the core control registers and one for the Linked List Descriptor RAM (Chapter 5). Base Addresses mentioned in this chapter are Register Space Base Adresses.

Table 4–1: Register Space Memory Map

Register Name	Address (Register Space Base Address +)	Access	Description
DMA Restart	0x00	R/W	Controls the reset of the DMA engine.
DMA Advance	0x04		Controls the start of link chain execution after a DMA restart.
DMA Abort	0x08		Controls the abort operation of the DMA.
DMA Start Link Address	0x0C		Controls the DMA Start Link Descriptor Address.
FIFO Flush	0x10		Controls the reset of Input FIFO.
Reserved	0x14		Reserved
	0x18		
	0x1C	R	
DMA Status	0x20		Indicates the DMA Status.
Current Link Address	0x24		Indicates the current link address.
Last Link Address	0x28		Indicates the last link address.
Bytes Last Transferred	0x2C		Indicates the number of bytes transferred in the last executed DMA link.
FIFO Status Register	0x30		Indicates the number of dwords available to be read from the Input FIFO.
Interrupt Enable	0x34	R/W	Interrupt enable bits
Interrupt Status	0x38	R	Interrupt status bits
Interrupt Flag	0x3C	R/Clr	Interrupt flag bits

4.1 DMA Restart Register

This register is a control register that controls the reset of the DMA engine. It resets the Linked List State Machine of the core. The DMA Restart Register is illustrated in Figure 4–1 and described in Table 4–2. The control bit of this register must be toggled once to start execution of a link chain after power–up and DMA configuration, or after a DMA abort sequence, or after a “chain end” (when end of link chain is reached).

Figure 4–1: DMA Restart Register

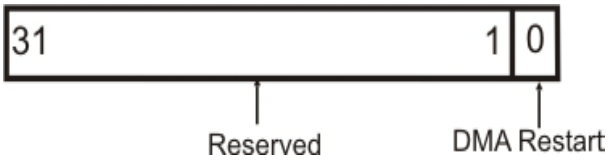


Table 4–2: DMA Restart Register (Base Address + 0x00)

Bits	Field Name	Default Value	Access Type	Description
31:1	Reserved	N/A	N/A	**Reserved**
0	dma_restart	0	R/W	**DMA Restart:** When this bit is toggled ‘1’ then ‘0’, the DMA is reset.

4.2 DMA Advance Register

The DMA Advance Register is used to advance the DMA linked list. The DMA Advance Register control bit must be toggled to start execution of the first DMA link descriptor in a chain regardless of whether that link is set to auto or manual mode of operation. It is then used to start execution of any link in the chain that is set to manual start mode. A link that is set to manual start mode will indicate it is waiting to start with the "waiting for advance" status bit of the DMA Status register (see Table 4–7). This register is illustrated in Table 4–2 and described in Table 4–3.

Figure 4–2: DMA Advance Register

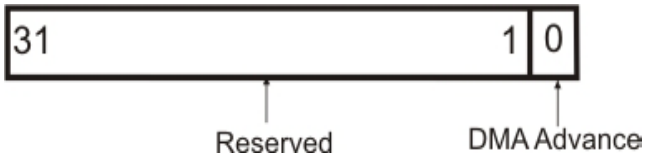


Table 4–3: DMA Advance Register (Base Address + 0x04)

Bits	Field Name	Default Value	Access Type	Description
31:1	Reserved	N/A	N/A	**Reserved**
0	dma_advance	0	R/W	**DMA Advance:** When toggled '1' then '0', the Linked List State Machine starts the execution of the first link descriptor after reset. The address of the first link descriptor in the linked list RAM is given by the DMA Start Link Address Control Register (see **Section 4.4**). This bit must also be toggled to start any link in a chain if that link descriptor is set to manual execution mode.

4.3 DMA Abort Register

The DMA Abort Register is used to control the generation of a DMA engine abort sequence. When an abort sequence is enabled, the DMA packets that have already been constructed by the core will be allowed to be transmitted while new DMA activity will be inhibited. When all packets have been transmitted, a DMA reset will automatically be generated. After a DMA abort, the DMA Restart Register control bit must be toggled to start execution of a new link chain. This register is illustrated in Figure 4–3 and described in Table 4–4.

Figure 4–3: DMA Abort Register

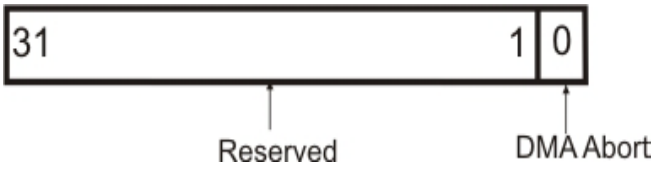


Table 4–4: DMA Abort Register (Base Address + 0x08)

Bits	Field Name	Default Value	Access Type	Description
31:1	Reserved	N/A	N/A	**Reserved**
0	dma_abort	0	R/W	**DMA Abort:** When toggled '1' then '0' a DMA abort sequence will commence.

4.4 DMA Start Link Address Register

The DMA Start Link Address Register is used to control the address of the Linked list Descriptor RAM from where the DMA engine starts execution after reset. This register is illustrated in Figure 4–4 and described in Table 4–5.

Figure 4–4: DMA Start Link Address

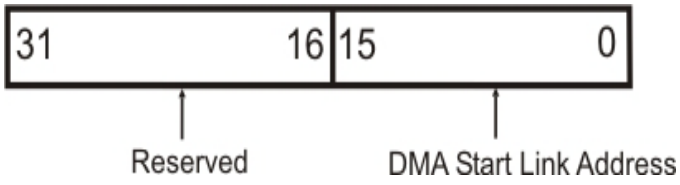


Table 4–5: DMA Start Link Address Register (Base Address + 0x0C)

Bits	Field Name	Default Value	Access Type	Description
31:16	Reserved	N/A	N/A	**Reserved**
15:0	start_link_ descr_addr	0x0000	R/W	**DMA Start Link Descriptor Address:** These bits control the value of the start address of the Link List Descriptor RAM whose link definition is to be executed by the DMA after a reset.

4.5 FIFO Flush Register

The FIFO Flush Register is used to reset the Input FIFO and clear its contents. Typically, it is a good practice to flush the FIFO before starting the execution of a new DMA link chain in case any extraneous requests or data was left in the FIFO. This register is illustrated in Figure 4–5 and described in Table 4–6.

Figure 4–5: FIFO Flush Register

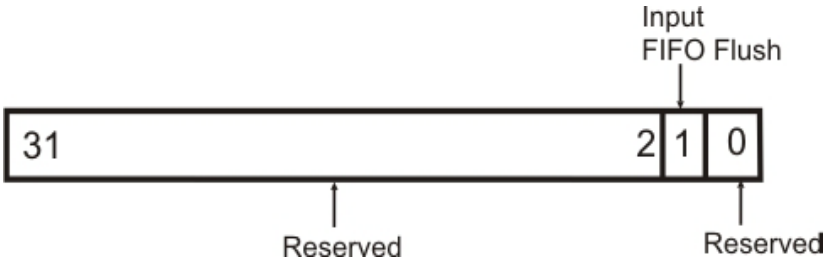


Table 4–6: FIFO Flush Register (Base Address + 0x10)

Bits	Field Name	Default Value	Access Type	Description
31:2	Reserved	N/A	N/A	**Reserved**
1	in_fifo_flush	0	R/W	**Input FIFO Flush:** This bit is used to reset the Input FIFO. The input FIFO is used to store the incoming AXI4–Stream data across the Packetized Sample Data/ Timestamp/ Data Information AXI4–Stream Slave Interface. 0 = Run 1 = Reset
0	Reserved	N/A	N/A	**Reserved**

4.6 DMA Status Register

The DMA Status Register indicates the status of the DMA engine and the input FIFO. This register is illustrated in Figure 4–6 and described in Table 4–7.

Figure 4–6: DMA Status Register

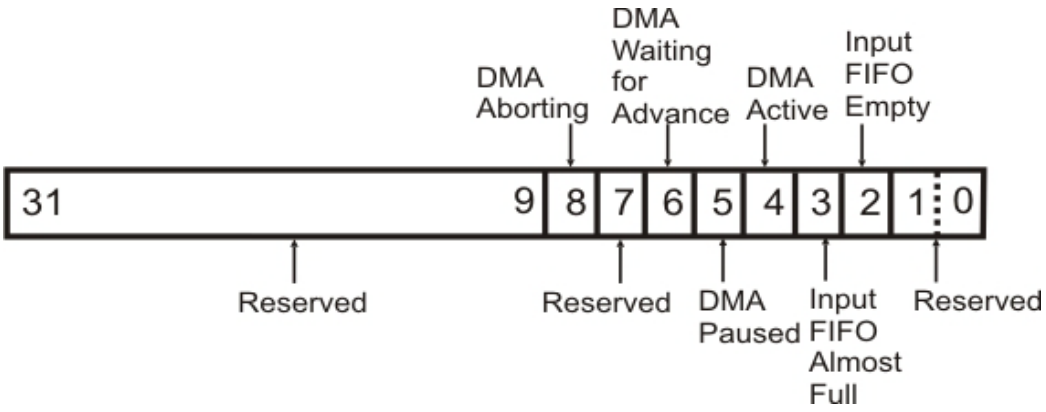


Table 4-7: DMA Status Register (Base Address + 0x20)

Bits	Field Name	Default Value	Access Type	Description
31:9	Reserved	N/A	N/A	**Reserved**
8	dma_aborting	0	R	**DMA Aborting:** When this bit '0', the DMA is not in an abort cycle. When this bit '1', a DMA abort is in progress. 0 = Normal 1 = DMA abort
7	Reserved	N/A	N/A	**Reserved**
6	dma_waiting_for_adv	0	R	**DMA Waiting for Advance:** This bit indicates that the DMA is waiting for the DMA advance bit of the DMA Advance control register to be toggled. 0 = Not waiting 1 = Waiting for advance
5	dma_paused			**DMA Paused:** This bit indicates that the DMA is paused. 0 = DMA running 1 = DMA paused
4	dma_active			**DMA Active:** This bit indicates that the DMA is executing a Link Descriptor. 0 = DMA inactive 1 = DMA active

Table 4-7: DMA Status Register (Base Address + 0x20) (Continued)				
Bits	**Field Name**	**Default Value**	**Access Type**	**Description**
3	in_fifo_afl	0	R	**Input FIFO Almost Full:** This bit indicates that the input FIFO of the AXI4–Stream to PCIe DMA 512–Bit Core is almost full. 0 = FIFO not full 1 = FIFO almost full
2	in_fifo_empty			**Input FIFO Empty:** This bit indicates that the Input FIFO of the AXI4–Stream to PCIe DMA 512–Bit Core is empty. 0 = FIFO not empty 1 = FIFO empty
1:0	Reserved	N/A	N/A	**Reserved**

4.7 Current Link Address Register

The Current Link Address Register is a status register which indicates the current link address of the Link Descriptor being executed. The default value of the current link address is the start link address defined by the DMA Start Link Address register, at reset. This register is illustrated in Figure 4–7, below, and described in Table 4–8.

Figure 4–7: Current Link Address Register

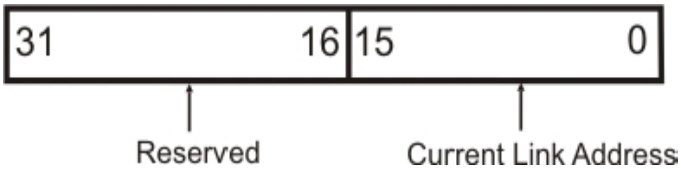


Table 4–8: Current Link Address Register (Base Address + 0x24)

Bits	Field Name	Default Value	Access Type	Description
31:16	Reserved	N/A	N/A	**Reserved**
15:0	current_link _address	start link descriptor address	R	**Current Link Address:** These bits indicate the current link address of the Link descriptor in the Linked List Descriptor RAM.

4.8 Last Link Address Register

The Last Link Address Register is a status register which indicates the link address of the Link Descriptor previously executed. This register is illustrated in Figure 4–8 and described in Table 4–9.

Figure 4–8: Last Link Address Register

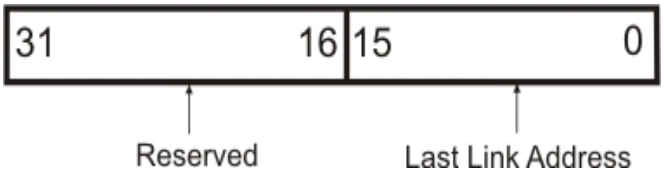


Table 4–9: Last Link Address Register (Base Address + 0x28)

Bits	Field Name	Default Value	Access Type	Description
31:16	Reserved	N/A	N/A	**Reserved**
15:0	last_link_address	0x0000	R	**Last Link Address:** These bits indicate the link address of the previously executed Link descriptor in the Linked List Descriptor RAM.

4.9 Bytes Last Transferred Register

The Bytes Last Transferred Register is a status register which indicates the number of bytes transferred in the last executed link descriptor. This register is illustrated in Figure 4–9 and described in Table 4–10.

Figure 4–9: Bytes Last Transferred Register

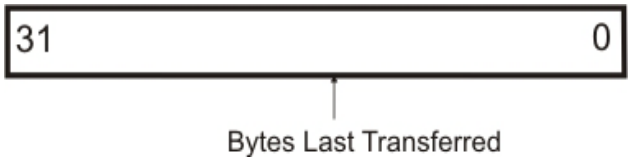


Table 4–10: Bytes Last Transferred Register (Base Address + 0x2C)

Bits	Field Name	Default Value	Access Type	Description
31:0	bytes_last_ transferred	0x00000000	R	**Bytes Last Transferred:** These bits indicate the number of bytes transferred in the last executed link descriptor.

4.10 FIFO Status Register

The FIFO Status register indicates the dword count of the maximum dwords that can be read from the input FIFO and the dwords currently present in the input FIFO. This register is illustrated in Figure 4–10 and described in Table 4–11.

Figure 4–10: FIFO Status Register

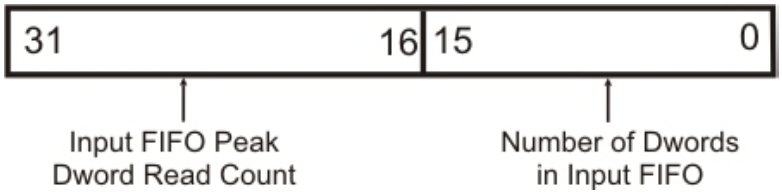


Table 4–11: FIFO Status Register (Base Address + 0x30)

Bits	Field Name	Default Value	Access Type	Description
31:16	rd_peak	0x0000	R	**Input FIFO Peak dword Read Count:** These bits indicate the maximum number of dwords of data that can be read from the Input FIFO. In this case each dword corresponds to 16 bits of data.
15:0	rd_data_count			**Number of dwords in Input FIFO:** These bits indicate the number of dwords (16–bit) of data currently available for reading from the Input FIFO of the AXI4–Stream to PCIe DMA 512–Bit Core.

4.11 Interrupt Enable Register

The bits in the interrupt enable register are used to enable (or disable) the generation of interrupts based on the condition of certain circuit elements, known as interrupt sources. When a bit in this register associated with a given interrupt source is High, an interrupt will be generated by the rising edge of that source's Interrupt Status Register bit (See Section 4.12). This register is illustrated in Figure 4–11 with the bits described in Table 4–12.

Figure 4–11: Interrupt Enable Register

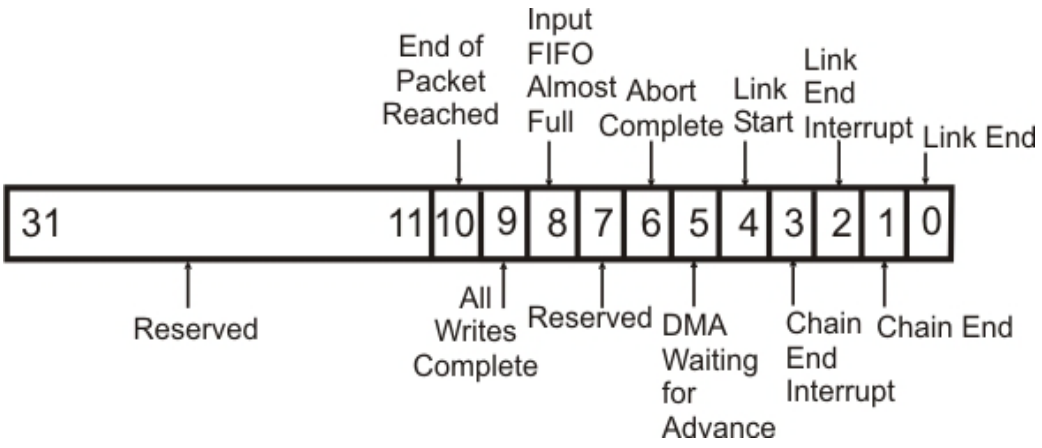


Table 4–12: Interrupt Enable Register (Base Address + 0x34)

Bits	Field Name	Default Value	Access Type	Description
31:11	Reserved	N/A	N/A	**Reserved**
10	eop_reached	0	R/W	**End of Packet reached:** This bit enables/ disables the end of packet reached interrupt source. The end of packet reached interrupt source indicates that the end of packet of the input data stream has been reached. 0 = Disable interrupt 1 = Enable interrupt
9	all_wr_cmplt	0	R/W	**All Writes Complete:** This bit enables/ disables the all writes complete interrupt source. The all writes complete interrupt source indicates that all writes to the Xilinx PCIe Core for a write request have been completed. 0 = Disable interrupt 1 = Enable interrupt

Table 4–12: Interrupt Enable Register (Base Address + 0x34) (Continued)				
Bits	**Field Name**	**Default Value**	**Access Type**	**Description**
8	in_fifo_afl	0	R/W	**Input FIFO Almost Full:** This bit enables/ disables the input data FIFO almost full interrupt source. 0 = Disable interrupt 1 = Enable interrupt
7	Reserved	N/A	N/A	**Reserved**
6	abort_cmplt	0	R/W	**Abort Complete:** This bit enables/ disables the DMA abort complete interrupt source. 0 = Disable interrupt 1 = Enable interrupt
5	waiting_adv	0	R/W	**Waiting for DMA Advance:** This bit enables/ disables the waiting for DMA advance interrupt source. The waiting for DMA advance interrupt source indicates that the DMA is waiting for the DMA advance signal, after a reset. 0 = Disable interrupt 1 = Enable interrupt
4	link_start	0	R/W	**Link Start:** This bit enables/ disables the link start interrupt source. The link start interrupt source indicates the start of execution of a link descriptor by the DMA after reset. 0 = Disable interrupt 1 = Enable interrupt
3	chain_end_int	0	R/W	**Chain End Interrupt:** This bit enables/ disables the chain end interrupt source. The chain end interrupt source indicates that the end of link descriptor chain has been reached. This interrupt source will be set only when the Chain End Interrupt Enable bit of the Link Descriptor Control Word is set to '1' (see **Table 5–3**). 0 = Disable interrupt 1 = Enable interrupt
2	link_end_int	0	R/W	**Link End Interrupt:** This bit enables/ disables the link end interrupt source. The link end interrupt source indicates that the current link execution is complete. This interrupt source will be set only when the Link End Interrupt Enable bit of the Link Descriptor Control Word is set to '1' (see **Table 5–3**). 0 = Disable interrupt 1 = Enable interrupt
1	chain_end	0	R/W	**Chain End:** Reserved for test purposes.
0	link_end	0	R/W	**Link End:** Reserved for test purposes.

4.12 Interrupt Status Register

The Interrupt Status Register has read–only access associated with each interrupt condition. A status bit changes to '1' when the source interrupt occurs. When a status bit in this register changes to '1' the corresponding flag bit in the Interrupt Flag Register is set to '1'. A status bit in this register clears to '0' when that interrupt condition clears, whereas the associated flag bit in the Interrupt Flag Register remains at logic '1' until it is explicitly cleared by the user.

Some of the interrupt sources are transient and so may not appear in the Interrupt Status Register at the time it is read. In such cases use the Interrupt Flag Register to see the interrupt conditions that have occurred. This register is illustrated in Figure 4–11 with the bits described in Table 4–12.

Figure 4–12: Interrupt Status Register

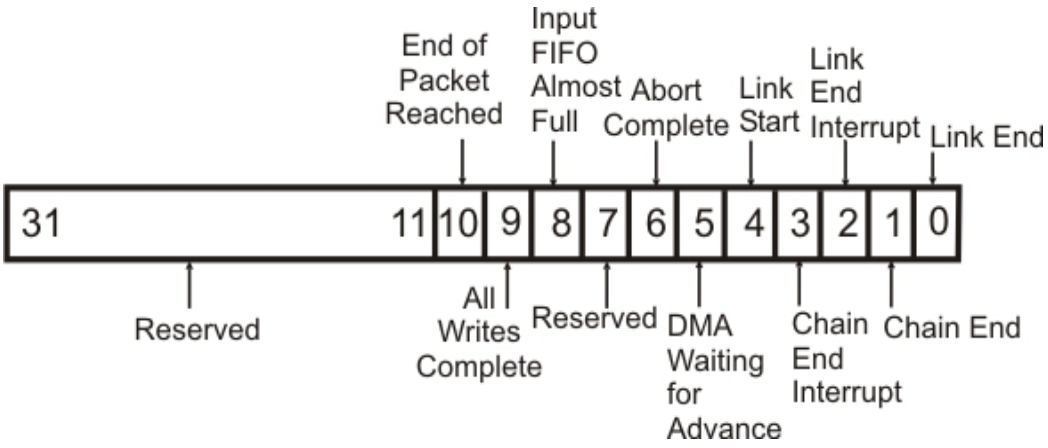


Table 4–13: Interrupt Status Register (Base Address + 0x38)

Bits	Field Name	Default Value	Access Type	Description
31:11	Reserved	N/A	N/A	**Reserved**
10	eop_reached	0	R	**End of Packet reached:** This bit indicates the status of the end of packet reached interrupt source. The end of packet reached interrupt source indicates that the end of packet of the input data stream has been reached. 0 = No interrupt 1 = Interrupt condition asserted

Table 4–13: Interrupt Status Register (Base Address + 0x38) (Continued)

Bits	Field Name	Default Value	Access Type	Description
9	all_wr_cmplt	0	R	**All Writes Complete:** This bit indicates the status of the all writes complete interrupt source. The all writes complete interrupt source indicates that all writes to the Xilinx PCIe Core for a write request have been completed. 0 = No interrupt 1 = Interrupt condition asserted
8	in_fifo_afl	0	R	**Input FIFO Almost Full:** This bit indicates the status of the input data FIFO almost full interrupt source. 0 = No interrupt 1 = Interrupt condition asserted
7	Reserved	N/A	N/A	**Reserved**
6	abort_cmplt	0	R	**Abort Complete:** This bit indicates the status of the DMA abort complete interrupt source. 0 = No interrupt 1 = Interrupt condition asserted
5	waiting_adv	0	R	**Waiting for DMA Advance:** This bit indicates the status of the waiting for DMA advance interrupt source. The waiting for DMA advance interrupt source indicates that the DMA is waiting for the DMA advance signal, after a reset. 0 = No interrupt 1 = Interrupt condition asserted
4	link_start	0	R	**Link Start:** This bit indicates the status of the link start interrupt source. The link start interrupt source indicates the start of execution of a link descriptor by the DMA after reset. 0 = No interrupt 1 = Interrupt condition asserted
3	chain_end_int	0	R	**Chain End Interrupt:** This bit indicates the status of the chain end interrupt source. The chain end interrupt source indicates that the end of link descriptor chain has been reached. This interrupt source will be set only when the Chain End Interrupt Enable bit of the Link Descriptor Control Word is set to '1' (see **Table 5–3**). 0 = No interrupt 1 = Interrupt condition asserted
2	link_end_int	0	R	**Link End Interrupt:** This bit indicates the status of the link end interrupt source. The link end interrupt source indicates that the current link execution is complete. This interrupt source will be set only when the Link End Interrupt Enable bit of the Link Descriptor Control Word is set to '1' (see **Table 5–3**). 0 = No interrupt 1 = Interrupt condition asserted

Table 4–13: Interrupt Status Register (Base Address + 0x38) (Continued)				
Bits	**Field Name**	**Default Value**	**Access Type**	**Description**
1	chain_end	0	R	**Chain End:** Reserved for test purposes.
0	link_end	0	R	**Link End:** Reserved for test purposes.

4.13 Interrupt Flag Register

The Interrupt Flag Register has read/clear access associated with each interrupt condition. When reset, this register has all bits set to '0' (cleared). Each flag bit in this register latches an interrupt occurrence. A '1' in any flag bit in this register indicates that an interrupt has occurred.

Note that when any status bit in the Interrupt Status Register, changes to '1' the corresponding flag bit in this register will also be set to '1'. However, when a status bit in the Interrupt Status Register clears from '1' to '0', the corresponding latched flag bit in this register does not clear, but remains at '1'. To clear the flag bits, write '1's to the desired bits. The flags are not affected by the Interrupt Enable Register. This register is illustrated in Figure 4–11 with the bits described in Table 4–12.

Figure 4–13: Interrupt Flag Register

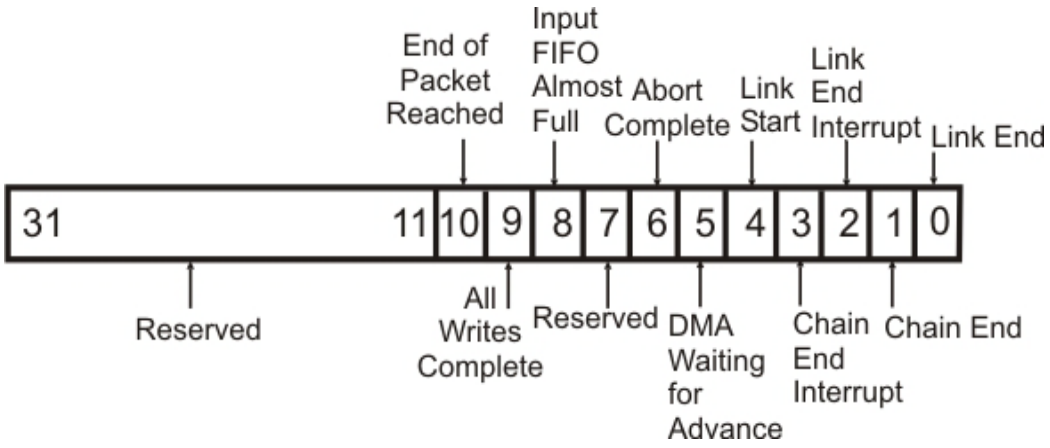


Table 4–14: Interrupt Flag Register (Base Address + 0x3C)

Bits	Field Name	Default Value	Access Type	Description
31:11	Reserved	N/A	N/A	**Reserved**
10	eop_reached	0	R/Clr	**End of Packet reached:** This bit indicates the end of packet reached interrupt flag. The end of packet reached interrupt source indicates that the end of packet of the input data stream has been reached. **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch

Table 4–14: Interrupt Flag Register (Base Address + 0x3C) (Continued)

Bits	Field Name	Default Value	Access Type	Description
9	all_wr_cmplt	0	R/Clr	**All Writes Complete:** This bit indicates the all writes complete interrupt flag. The all writes complete interrupt source indicates that all writes to the Xilinx PCIe Core for a write request have been completed. **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch
8	in_fifo_afl	0	R/Clr	**Input FIFO Almost Full:** This bit indicates the input data FIFO almost full interrupt flag. **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch
7	Reserved	N/A	N/A	**Reserved**
6	abort_cmplt	0	R/Clr	**Abort Complete:** This bit indicates the DMA abort complete interrupt flag. **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch
5	waiting_adv	0	R/Clr	**Waiting for DMA Advance:** This bit indicates the waiting for DMA advance interrupt flag. The waiting for DMA advance interrupt source indicates that the DMA is waiting for the DMA advance signal, after a reset. **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch
4	link_start	0	R/Clr	**Link Start:** This bit indicates the link start interrupt flag. The link start interrupt source indicates the start of execution of a link descriptor by the DMA after reset. **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch

Table 4–14: Interrupt Flag Register (Base Address + 0x3C) (Continued)				
Bits	**Field Name**	**Default Value**	**Access Type**	**Description**
3	chain_end_int	0	R/Clr	**Chain End Interrupt:** This bit indicates the chain end interrupt flag. The chain end interrupt source indicates that the end of link descriptor chain has been reached. This interrupt source will be set only when the Chain End Interrupt Enable bit of the Link Descriptor Control Word is set to '1' (see **Table 5–3**). **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch
2	link_end_int	0	R/Clr	**Link End Interrupt:** This bit indicates the link end interrupt flag. The link end interrupt source indicates that the current link execution is complete. This interrupt source will be set only when the Link End Interrupt Enable bit of the Link Descriptor Control Word is set to '1' (see **Table 5–3**). **Read:** 0 = No interrupt 1 = Interrupt latched **Clear:** 1 = Clear latch
1	chain_end	0	R/Clr	**Chain End:** Reserved for test purposes.
0	link_end	0	R/Clr	**Link End:** Reserved for test purposes.

Chapter 5: Linked List Descriptor RAM Memory Maps

Table 5–1 defines the AXI4–Stream to PCIe DMA 512–Bit Core's Linked List Descriptor RAM memory maps. The RAM can store up to 1024 link descriptors.

NOTE: There are two seperate memory map addresses for this IP Core: one for the core control registers (Chapter 4) and one for the Linked List Descriptor RAM

Table 5–1: Linked List Descriptor RAM Memory Map

Descriptor	Address (Descriptor RAM Base Address +)	Access	Description
Descriptor 0	0x0000	R/W	Control Word
	0x0004		Number of Bytes to transfer[31:0]
	0x0008		PCIe Destination Address [31:0]
	0x000C		PCIe Destination Address and Loop Counter
	0x0010		PCIe Destination metadata Address[31:0]
	0x0014		PCIe Destination metadata Address [63:32]
	0x0018		Loop Address Increment Value[31:0]
	0x001C		Next Link Address [15:0]
Descriptor 1	0x0020 – 0x003C		
Descriptor 2	0x0040 – 0x005C		
.....			
Descriptor 1023	0x7FE0	R/W	Control Word
	0x7FE4		Number of Bytes to transfer[31:0]
	0x7FE8		PCIe Destination Address [31:0]
	0x7FEC		PCIe Destination Address and Loop Counter
	0x7FF0		PCIe Destination metadata Address[31:0]
	0x7FF4		PCIe Destination metadata Address [63:32]
	0x7FF8		Loop Address Increment Value[31:0]
	0x7FFC		Next Link Address [15:0]

Table 5–2 defines the fields in each link descriptor stored in the Linked List Descriptor RAM.

Table 5-2: Link Descriptor Field Definitions

Address (Descriptor RAM Base address+)	Bits	Field Name	Access Type	Description
0x0000	**31:0**	control_word	R/W	**Control Word:** These bits define the control bits for the link descriptor. The individual bits of the control word are explained in the **Table 5–3**.
0x0004	**63:32**	bytes_to_transfer	R/W	**Bytes to Transfer:** These bits indicate the number of bytes of data to be written to the PCIe address.
0x0008	**95:64**	dest_addr	R/W	**PCIe Destination Address:** These bits hold the value of the PCIe address where the PCIe write request data is to be written.
0x000C	**127:96**			
0x0010	**159:128**	dest_meta_ data_addr	R/W	**PCIe Destination metadata Address:** These bits hold the value of the PCIe address where the PCIe write request metadata is to be written.
0x0014	**191:160**			
0x0018	**223:192**	loop_incr	R/W	**Loop Address Increment Value:** These bits define the address value by which the PCIe destination address in the link increments when the loop increment mode is enabled. Loop increment mode is defined in bit [3] in the control word of a link descriptor. (see **Table 5–3**)
0x001C	**239:224**	next_link_addr	R/W	**Next Link Address:** These bits indicate the link descriptor RAM address of the next link descriptor to be executed.
	255:240	Reserved	N/A	**Reserved**

Table 5–3 defines the bits in the Control Word of the link descriptor stored in the Linked List Descriptor RAM.

Table 5-3: Link Descriptor Control Word Bit Definitions

Bits	Field Name	Default Value	Access Type	Description
31:16	incr_loop_cnt	0x0000	R/W	**Increment Loop Count:** These bits indicate the number of times the current link is executed before moving to the next link when the loop increment mode (control word [3]) is enabled.
15:14	Reserved	N/A	N/A	**Reserved**
13:12	pcie_at	0	R/W	**PCIe Address Type:** These bits define the address type of the memory transaction. These bits are used by the Xilinx PCIe Core to define the address type in the header of the request TLP. Please refer to the *Xilinx PCIe Core Product Guide* for more details. 00 : Address in the request is untranslated 01 : Transaction is a transaction request 10 : Address in the request is a translated address 11 : Reserved
11	write_meta_data	0	R/W	**Write metadata:** This bit is used to enable (or disable) writing of metadata packets into the Xilinx PCIe Core. Metadata packet bit definitions are available in **Table 3–6**. 0 = Disable 1 = Enable
10	chain_end	0	R/W	**Chain End:** This bit indicates whether the link descriptor executed is the end of the link chain. 0 = This link is not the end of the chain 1 = This link is the end of the chain
9	chain_end_int	0	R/W	**Chain End Interrupt Enable:** This bit enables (or disables) the generation of an interrupt when the link chain ends. 0 = No interrupt is generated 1 = Generates interrupt when the link chain is completely executed
8	link_end_int	0	R/W	**Link End Interrupt enable:** This bit enables (or disables) the generation of an interrupt when the link has been executed. 0 – No interrupt is generated 1 – Generates an interrupt when a link is executed

Table 5-3: Link Descriptor Control Word Bit Definitions (Continued)				
Bits	**Field Name**	**Default Value**	**Access Type**	**Description**
7	end_on_eop	0	R/W	**End on End of Packet Mode:** This bit is used to enable (or disable) the end of DMA transfer when the end of a packet is reached. 0 = Does not end the DMA transfer when an end of packet is reached and continues until the DMA transfer length is achieved. 1 = Ends DMA transfer when an end of packet is reached.
6:4	Reserved	N/A	N/A	**Reserved**
3	incr_mode	0	R/W	**Loop Increment Mode:** This bit enables (or disables) the loop increment mode of the DMA engine. When this mode is enabled the current link is executed for a loop count defined by the increment loop count value and the PCIe destination address increments by a value defined by the loop address increment value after each execution of the link. 0 = Disable 1 = Enable
2	sop_start	0	R/W	**Start on Start of Packet Mode:** This bit is used to enable (or disable) the start of DMA transfer only when gate signal is observed. 0 = Starts the DMA transfer irrespective of the gate signal. 1 = Starts DMA transfer only when the start of packet is observed when the gate signal is ‘1’.
1	Reserved	N/A	N/A	**Reserved**
0	start_mode	0	R/W	**Start Mode:** This bit indicates the mode of operation of the DMA engine. 0 = Manual mode: DMA advance bit in the DMA Advance Register must be toggled by the user to start execution of the link descriptor 1 = Auto mode: Execution of the link descriptor starts automatically Note: The first link in a link descriptor chain requires the DMA Advance Register’s advance bit to be toggled to start execution regardless of the setting of this bit.

Chapter 6: Designing with the Core

This chapter provides guidelines and additional information to facilitate designing with the AXI4–Stream to PCIe DMA 512–Bit Core.

6.1 General Design Guidelines

The AXI4–Stream to PCIe DMA 512–Bit Core provides the required logic to generate PCIe write requests from the input packed AXI data streams. The user can control the DMA operation by setting the desired values to the control registers in the Register Space as described in Chapter 4.

6.2 Clocking

Main Clock: **aclk**

This clock is used to clock all the ports on the AXI4–Stream to PCIe DMA 512–Bit Core, including the control/status register (CSR) interface.

AXI4–Stream Clock: **s_axis_ppkt_aclk**

This is input AXI4–Stream clock across the Packetized Sample Data/ Timestamp/ Data Information AXI4–Stream Slave Interface of the core.

6.3 Resets

Main reset: **aresetn**

This is an active low synchronous reset associated with **aclk**. When this reset is asserted, all state machines in the core are reset and FIFOs are flushed. Caution should be exercised that this reset is not asserted while the DMA is running, in order to avoid generation of incomplete or malformed packets at the PCIe Requestor Request Interface. When possible, an abort sequence should be used instead when the user design needs to stop the DMA operation. This reset does not affect the control/status registers, or link descriptor RAM contents.

CSR Reset: **s_axi_csr_aresetn**

This is an active low synchronous reset associated with **aclk**. When asserted, this reset will clear all control registers back to their initialized default states. It does not reset any of the state machines or flush any of the FIFOs in the core.

NOTE: No reset can clear the contents of the descriptor RAM. To clear the values in these memory blocks, the user design must manually write to each location.

6.4 Interrupts

This core has an edge type (rising edge−triggered) interrupt output. It is synchronous with the **aclk**. On the rising edge of any interrupt signal, a one clock cycle wide pulse is output from the core on its **irq** output. Each interrupt event is stored in two registers, accessible on the **s_axi_csr** bus. The Interrupt Status Register always reflects the current state of the interrupt condition, which may have changed since the generation of the interrupt. The Interrupt Flag Register latches the occurrence of each interrupt, in a bit that retains its state until explicitly cleared.The interrupt flags can be cleared by writing '1' to the associated bit's location. All interrupt sources that are enabled (via the Interrupt Enable Register) are "OR ed" onto the **irq** output.

NOTE: All interrupt sources are latched in the Interrupt Flag Register, even when an interrupt source is not enabled to create an interrupt.

NOTE: Because this core uses edge−triggered interrupts, the fact that an interrupt condition may remain active after servicing will not cause the generation of a new interrupt. A new interrupt will only be generated by another rising edge on an interrupt source.

6.5 Interface Operation

- **CSR Interface:** This is the control/status register interface. It is associated with **aclk**. It is a standard AXI4−Lite Slave interface. Typically, this interface is connected along with other cores' AXI4−Lite interfaces through an AXI Lite Crossbar core or a series of AXI Lite Crossbar cores that route AXI Lite accesses through to the desired core based on the address range.
- **Linked List Descriptor (DESCR) RAM Interface:** This is the DMA Linked List Descriptor RAM interface which is used by the host to configure the DMA descriptors. It is associated with **aclk** and is a standard AXI4−Lite Slave interface. Typically, this interface is connected along with other cores' AXI4−Lite interfaces through an AXI Lite Crossbar core or a series of AXI Lite Crossbar cores that route AXI Lite accesses through to the desired core based on the address range.
- **PCIe Requester Request Interface:** This is the PCIe Requester Request Interface which is associated with **aclk** and is used to transfer PCIe write requests from the AXI4−Stream to PCIe DMA 512−Bit Core to the Xilinx PCIe Core. This is a standard AXI4−Stream Master interface which is compatible with the Xilinx PCIe Core's Requester Request Bus when it is set up to be 256 bits wide and operating in address−aligned mode.

  Typically, this interface is connected along with other DMA cores through an AXI4−Stream Switch Core that arbitrates multiple input streams into one output stream, to the Xilinx PCIe Core's Requester Request Bus.

  This interface must be connected to Xilinx PCIe Core through a Pentek PCIe Requester Interface Gasket core in order to convert the standard **tkeep** and **tready** signals of the PCIe request from the AXI4−Stream to PCIe DMA 512−Bit Core into format compatible with the Xilinx PCIe Core's Requester Request Bus signals

6.5 Interface Operation (continued)

- **Packetized Sample Data/ Timestamp/ Information Streams (PPKT) Interface:** This core implements an AXI4−Stream Slave interface across the input to receive AXI PPKT streams and is associated with **`s_axis_ppkt_aclk`**. For more details about this interface, refer to **Section 3.2**.

6.6 Programming Sequence

This section briefly describes the programming sequence for the AXI4−Stream to PCIe DMA 512−Bit Core.

1) Set up the linked list RAM.
2) Set the control registers with the required values.
3) Enable DMA interrupts.
4) Reset the DMA.
5) Start the DMA.
6) When done, check the interrupt flag register and clear the interrupts.

6.7 Timing Diagrams

The timing diagram for the AXI4−Stream to PCIe DMA 512−Bit Core is shown in Figure 7−4. This timing diagram is obtained by running the simulation of the test bench of the core in Vivado VSim environment. For more details about the test bench, refer to Section 7.5.

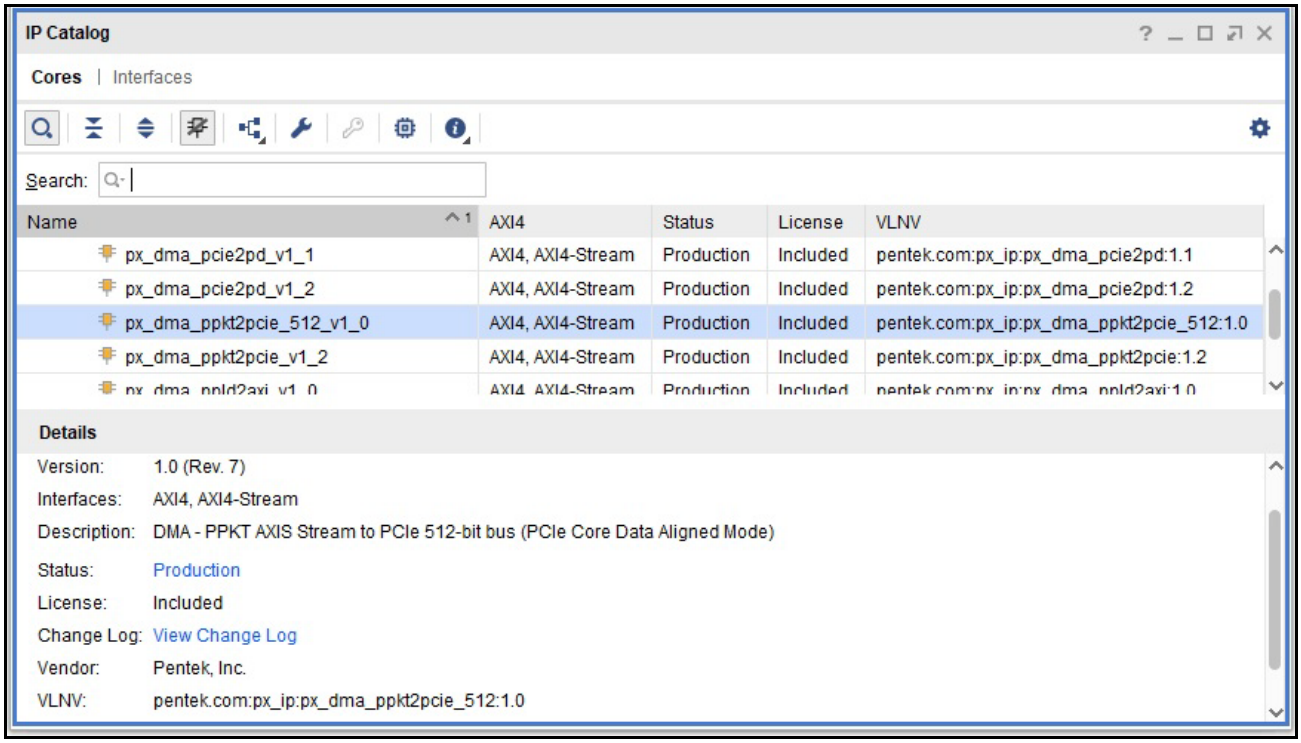
This page is intentionally blank

Chapter 7: Design Flow Steps

7.1 Pentek IP Catalog

This chapter describes customization and generation of the Pentek AXI4–Stream to PCIe DMA 512–Bit Core. It also includes simulation, synthesis, and implementation steps that are specific to this IP core. This core can be generated from the Vivado IP Catalog when the Pentek IP Repository has been installed. It will appear in the IP Catalog list as **px_dma_ppkt2pcie_512_v1_0** as shown in Figure 7–1.

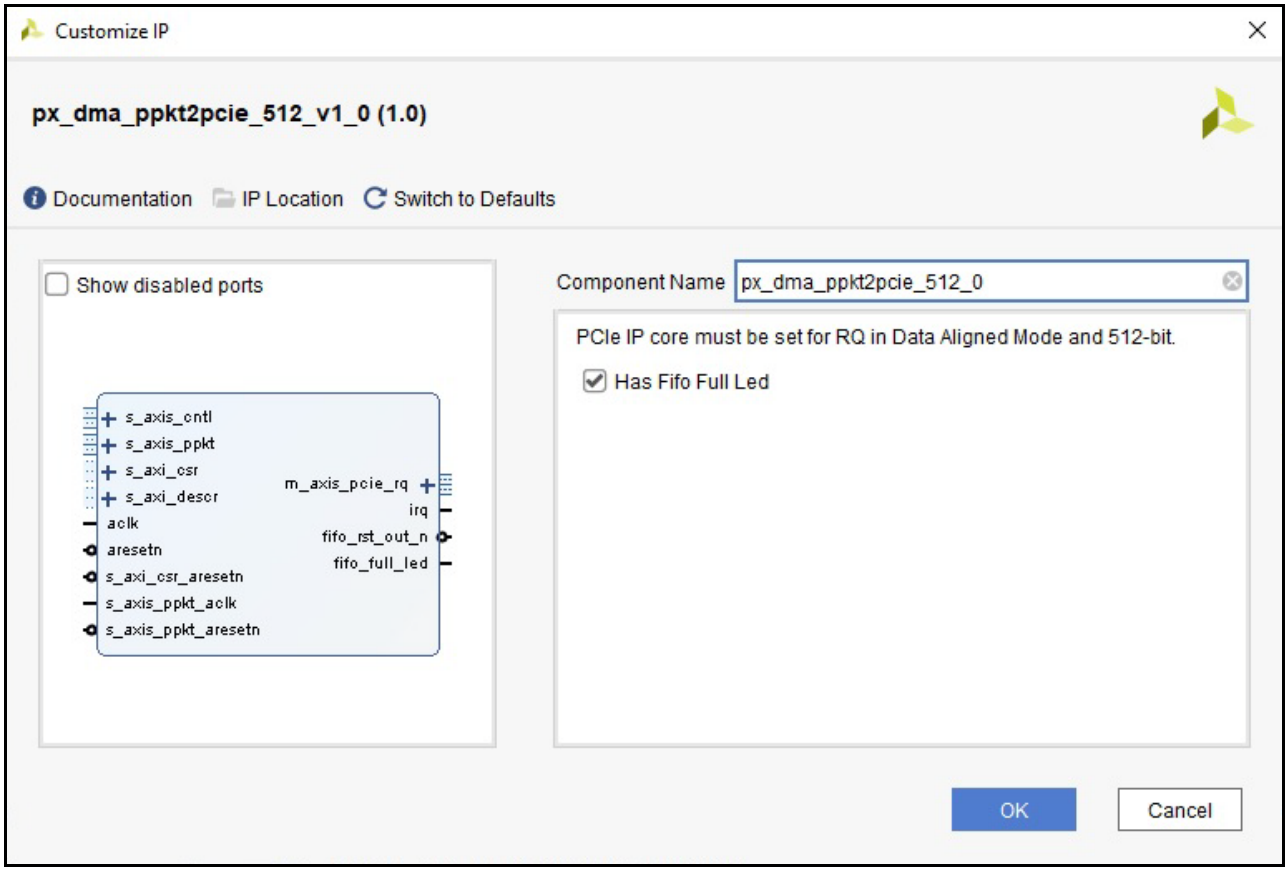
Figure 7–1: AXI4–Stream to PCIe DMA 512–Bit Core in Pentek IP Catalog



7.1 Pentek IP Catalog (continued)

When you select the **px_dma_ppkt2pcie_512_v1_0** core, a screen appears that shows the core's symbol and the core's parameters (see Figure 7–2). The core's symbol is the box on the left side.

Figure 7–2: AXI4–Stream to PCIe DMA 512–Bit Core IP Symbol



7.2 User Parameters

The user parameters of this AXI4–Stream to PCIe DMA 512–Bit Core are explained in Section 2.5 of this user manual.

7.3 Output Generation

For more details about generating and using IP in the Vivado Design Suite, refer to the *Vivado Design Suite User Guide – Designing with IP*.

7.4 Constraining the Core

This section contains information about constraining the AXI4−Stream to PCIe DMA 512−Bit Core in the Vivado Design Suite.

Required Constraints

The XDC constraints are not provided with the AXI4−Stream to PCIe DMA 512−Bit Core. Clock constraints can be applied in the top−level module of the user design.

Device, Package, and Speed Grade Selections

This IP works for the Ultrascale+ FPGAs.

Clock Frequencies

The main clock (**`aclk`**) and the AXI4−Stream clock (**`s_axis_ppkt_aclk`**) of the AXI4−Stream to PCIe DMA 512−Bit Core can both take frequencies up to 250 MHz.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking and Placement

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

7.5 Simulation

The AXI4–Stream to PCIe DMA 512–Bit Core has a test bench which generates output waveforms using the Vivado VSim environment. This test bench is designed to run at 200 MHz AXI4–Stream clock frequency (**s_axis_ppkt_aclk**) and 250 MHz main clock frequency (**aclk**).

The DMA writes to two buffer spaces whose link descriptors are defined in a **test_parameters.txt** file. This file also contains the required values for the control registers within the Register Space of the AXI4–Stream to PCIe DMA 512–Bit Core. The linked list Descriptor RAM is set up by the test bench by reading from the **test_parameters.txt** file. The DMA control registers are also set with desired values as defined in the test parameters file. The DMA is made operational by toggling the control bit of the DMA Advance control register. The programming sequence of the DMA is the same as described in Section 6.6.

The contents of the **test_parameters.txt** file along with descriptions of the parameters are provided in Table 7–1..

Table 7-1: Test Parameters File Contents and Parameter Descriptions

Parameter	Type	Value	Description
gate_length	Integer	256	**Gate Length:** This parameter indicates the length in 16 bit samples of active gate time for the input data stream.
gate_inactive_time	Integer	1024	**Gate Inactive Time:** This parameter indicates the length in 16 bit samples of inactive gate time between active gates. This value must be a multiple of input data stream word width (16–bit words).
gate_repeat	Integer	2	**Gate Repetitions:** This indicates the number of times the active gate period is to be repeated.
pcie_max_payload_size	std_logic_vector	1	**Maximum PCIe Payload Size:** This value is determined by the PCIe host. 000 – 128 Bytes maximum packet payload size 001 – 256 Bytes maximum packet payload size 010 – 512 Bytes maximum packet payload size 011 – 1024 Bytes maximum packet payload size 100 – 2048 Bytes maximum packet payload size 101 – 4096 Bytes maximum packet payload size
link_strt_addr	std_logic_vector	0000	**Link Start Address:** This parameter indicates the address of the first link descriptor to be executed.
pcie_a_address	std_logic_vector	0x00000000 100C0000	**PCIe Destination Start Address:** This is the PCIe destination start address where the request data from Buffer A is to be written.

Table 7-1: Test Parameters File Contents and Parameter Descriptions (Continued)

Parameter	Type	Value	Description
pcie_a_meta_address	std_logic_vector	0x00000000 20000000	**PCIe metadata Destination Start Address:** This is the PCIe metadata destination start address where metadata from the input data stream is to be written during the execution of the first link descriptor.
buffera_bytesize	Integer	256	**Buffer A Bytes to Transfer:** This parameter indicates the number of bytes to be transferred to the PCIe during the execution of the first link descriptor.
pcie_b_address	std_logic_vector	0x00000000 300C0000	**PCIe Destination Start Address:** This is the PCIe destination start address where data from Buffer B is to be written.
pcie_b_meta_address	std_logic_vector	0x00000000 40000000	**PCIe metadata Destination Start Address:** This is the PCIe metadata destination start address where metadata from input data stream is to be written during the execution of the second link descriptor.
bufferb_bytesize	Integer	256	**Buffer B Bytes to Transfer:** This parameter indicates the number of bytes to be transferred to the PCIe core during the execution of the second link descriptor.
Link Descriptor #1 (Refer to Table 5–2 and Table 5–3)			
a_auto	Boolean	True	**Start Mode:** This parameter indicates the mode of operation of the DMA engine. When set to True, the DMA works in auto mode, and when set to False the DMA runs in manual mode.
a_write_meta	Boolean	False	**Write metadata:** When set to False, this parameter disables writing of metadata packets to the Xilinx PCIe Core.
a_incr_mode	Boolean	False	**Loop Increment Mode:** When set True, this parameter enables the link descriptor to be executed for a loop count defined by **a_loop_length** with the PCIe Destination address incrementing by a value defined by **a_loop_incr_value** after each link execution.
a_sop_start	Boolean	False	**Start on Start of Packet mode:** When set to False, this parameter disables start of DMA transfer when start of packet marker in the input data stream is observed.
a_eop_end	Boolean	True	**End on End of Packet Mode:** When set to True, the DMA ends a DMA transfer after end of packet marker is reached in the input data stream.

Table 7-1: Test Parameters File Contents and Parameter Descriptions (Continued)

Parameter	Type	Value	Description
a_link_end_int_en	Boolean	True	**Link End Interrupt Enable:** When set to True, this parameter enables the generation of an interrupt when a link is executed.
a_chain_end_int_en	Boolean	False	**Chain End Interrupt Enable:** When set to True, this parameter enables generation of an interrupt when a link chain is completely executed.
a_chain_end	Boolean	False	**Chain End:** When set to False, this parameter indicates that this link descriptor is not the end of the link chain.
a_loop_length	Integer	5	**Loop Length:** This parameter indicates the number of times the current link is to executed before moving to the next link, when the loop increment mode is enabled.
a_loop_incr_value	std_logic_ vector	0x00000400	**Loop Increment Value:** This parameter indicates the address value by which the PCIe destination address in the link descriptor increments when loop increment mode is enabled.
Link Descriptor #2			
b_auto	Boolean	True	**Start Mode:** This parameter indicates the mode of operation of the DMA engine. When set to True, the DMA works in auto mode, and when set to False the DMA runs in manual mode.
b_write_meta	Boolean	False	**Write metadata:** When set to False, this parameter disables writing of metadata packets to the Xilinx PCIe Core.
b_incr_mode	Boolean	False	**Loop Increment Mode:** When set True, this parameter enables this link descriptor to be executed for a loop count defined by **b_loop_length** and the Destination PCIe address increments by a value define by **b_loop_incr_value** after each link execution. When set to False, this mode is disabled.
b_sop_start	Boolean	False	**Start on Start of Packet mode:** When set to False, this parameter disables start of DMA transfer only when start of packet in the input data stream is reached.
b_eop_end	Boolean	False	**End on End of Packet Mode:** When set to True, the DMA ends a DMA transfer after end of packet marker is reached in the input data stream.

Table 7-1: Test Parameters File Contents and Parameter Descriptions (Continued)			
Parameter	**Type**	**Value**	**Description**
b_link_end_int_en	Boolean	True	**Link End Interrupt Enable:** When set to True, this parameter enables the generation of an interrupt when a link is executed.
b_chain_end_int_en	Boolean	False	**Chain End Interrupt Enable:** When set to False, this parameter disables generation of an interrupt when a link chain is completely executed.
b_chain_end	Boolean	False	**Chain End:** When set to False, this parameter indicates that this link descriptor is not the end of the link chain.
b_loop_length	Integer	5	**Loop Length:** This parameter indicates the number of times the current link is to executed before moving to the next link, when the loop increment mode is enabled.
b_loop_incr_value	std_logic_ vector	0x00000400	**Loop Increment Value:** This parameter indicates the address value by which the PCIe address in the link increments when loop increment mode is enabled.

The input data stream to the DMA is generated by the test bench for a gate length of 256 16–bit samples with the gate repeated twice, which indicates that during each gate period 512 bytes of data is received. The input data stream width is set to 128 bits.

This test bench has Link Descriptor 1 defined to transfer 256 bytes of data from the input FIFO to the PCIe core, with loop increment mode disabled. The PCIe core has a maxi–mum payload size of 256 bytes.

- The first 256 bytes of input data is transferred to Buffer A within the PCIe core during the execution of the first link descriptor.
- The second link descriptor is also defined to transfer 256 bytes of data.
- After the execution of the first link descriptor, link descriptor 2 is executed, during which the next 256 bytes of data of the first gate acquisition period are transferred.

This process repeats for the second gate period.

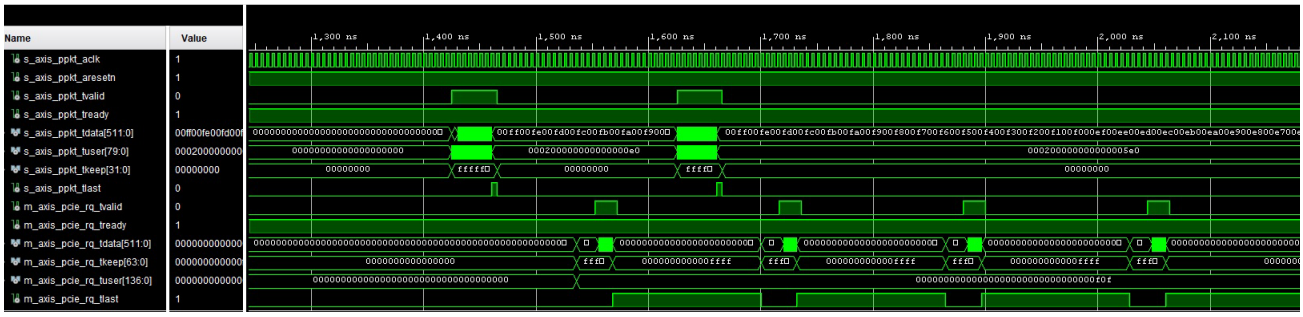
The two gate periods with the generated PCIe request for link descriptor 1 are shown in Figure 7–3, where each gate acquisition period can be distinguished based on the input **`tvalid`** signal. When the test bench is run, the simulation produces the results shown in Figure 7–4. The generated PCIe write request data streams by the DMA are stored in **`test_results.txt`** file. This output file contains information of all the generated PCIe write requests. The contents of the **`test_results.txt`** file are discussed in the Table 7–2.

7.5 Simulation (continued)

Table 7–2: Test Results File Contents

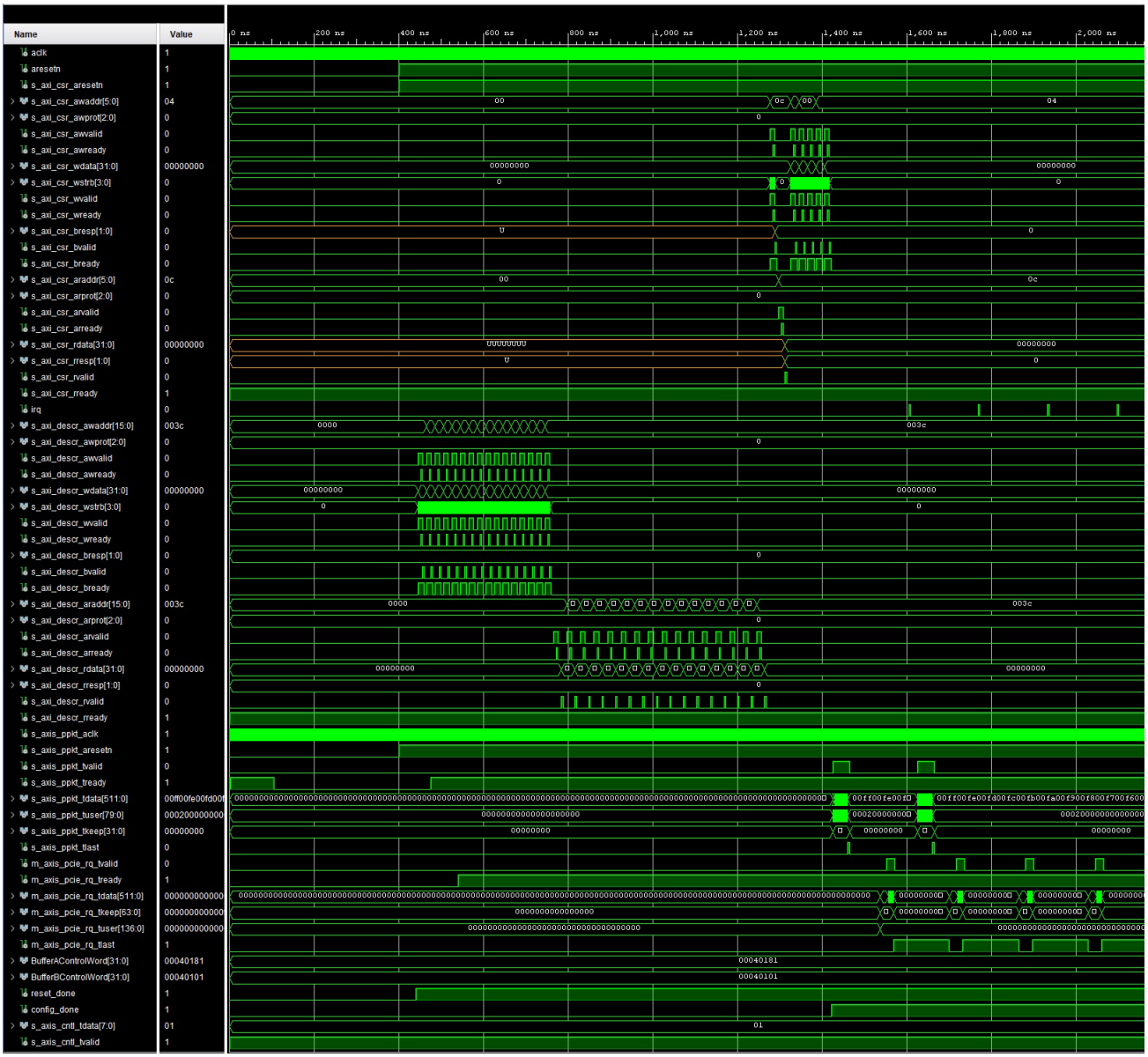
Parameter	Description
header	**Request Header:** This is a 16–byte long request descriptor which is transmitted during the start of packet transfer to the PCIe core and is followed by payload data. For more details about the descriptor refer to the *Xilinx PCIe Core Product Guide*.
at	**PCIe Address Type:** This parameter defines the address type of the memory transaction. These bits are used by the Xilinx PCIe Core to define the address type in the header of the request TLP. 0 : Address in the request is untranslated 1 : Transaction is a transaction request 2 : Address in the request is a translated address 3 : Reserved
dest_addr	**Destination Address:** This parameter indicates the PCIe destination address where the data is being written.
dword_cnt	**Dword Count:** This field indicates the number of 32–bit dwords being transferred in this PCIe write request.
dword_offset	**Dword Offset:** This is the dword number where the payload data begins on the data bus. This enables the Xilinx PCIe Core to determine the alignment of the data block being transferred.
req_type	**Request Type:** This indicates the type of PCIe request. It can either be a write request or wrong type request.
tag	**Tag:** This is the tag number uniquely identifying the request.
addr	**Address and Data:** This field in the output text file indicates the PCIe address where the data is being written, followed by the 32 bytes of data written to that address.

Figure 7–3: AXI4–Stream to PCIe DMA 512–Bit Core Test Bench Simulation Output – 1



7.5 **Simulation** (continued)

Figure 7–4: AXI4–Stream to PCIe DMA 512–Bit Core Test Bench Simulation Output – 2



7.6 Synthesis and Implementation

For details about synthesis and implementation see the *Vivado Design Suite User Guide – Designing with IP*.