XAPP1171 (v1.0) November 4, 2013

PCI Express Endpoint-DMA Initiator Subsystem

Author: Brian Martin

Summary

This application note demonstrates a Vivado® Design Suite subsystem for endpoint-initiated Direct Memory Access (DMA) data transfers through PCI Express®. The provided subsystems target the following devices to initiate data transfers between DDR3 memory and an externally connected PCI Express Root Complex:

- Zynq®-7000 ZC706 device
- Kintex®-7 KC705 device

Note: The subsystem can be modified for use in other devices or applications that require data transactions to be initiated from within the FPGA logic.

This application note demonstrates several key features of the Vivado Design Suite and the IP cores used in the design. The key features that are illustrated by this design include:

- Generating a block diagram subsystem using Vivado Design Suite Tcl commands and scripting
- PCI Express Endpoint configuration
- DMA initiated data transfers over PCI Express
- Achieving high-throughput into the Zynq-7000 device processing system (PS) through the High-Performance AXI interface
- Dynamic Address Translation between a 64-bit Root Complex (Host) address space and a 32-bit FPGA (AXI) address space
- A methodology to perform DMA Scatter Gather (SG) operations using Dynamic Address Translation

Each of these features are demonstrated to provide a reference for using Vivado Design Suite and the associated IP cores. This can be used as a base system to begin development or customized for a specific application. Using Tcl increases the rate of development and allows you to focus on the unique features of your application. The Tcl scripts used to generate these designs can be referenced or used to create a custom design.

This design demonstrates how to use the AXI Memory Mapped to PCI Express IP to perform high-throughput data transfers over a PCI Express link. To accomplish this, a Scatter Gather capable DMA engine is paired with the PCI Express IP. The DMA engine allows the FPGA to manage the data transfer over the PCI Express link to increase throughput and decrease processor utilization on the Root Complex side of the PCI Express link. This approach is important specifically for high-throughput PCI Express applications, which can include using the Zynq-7000 PS high-performance ports or double data rate (DDR) memory.

This application note also demonstrates the Dynamic Address Translation capability of the AXI Memory Mapped to PCI Express IP core and how Dynamic Address Translation can be used in DMA transactions. By using the Dynamic Address Translation capability a 32-bit addressable FPGA design can access the entire address range of a 64-bit addressable Root Complex, or Host processing system. This is vital for designs that have a 32-bit address space (for example, Zynq-7000 PS, MicroBlaze™ processor, or 32-bit addressable AXI Interconnect), and that interface with a Root Complex or Host system that has a 64-bit address space.

While a DDR3 memory is used as the target for DMA data transfers in this design, it can be replaced entirely with user logic or accessed by a custom processing system as well.

Introduction

This design was created for the Vivado Design Suite 2013.3 using Vivado IP integrator.

To generate the design no hardware is needed. Bit files can be generated to run on either the ZC706 or KC705 development boards depending on your configuration.

To test this targeted design platform in hardware, you need a host machine capable of PCI Express Gen1 x4, and the appropriate drivers and software that allow you to initiate PCI Express traffic to the FPGA endpoint. The drivers and software are not provided with this application note, and must be custom developed. Throughout this document, the custom driver and software are referred to as the *driver* and *software application*.

Requirements

The following hardware and software are required to generate the reference systems.

Hardware Requirements

- Xilinx Zynq ZC706 or Kintex-7 KC705 development boards
- One USB (Type A to Type B)
- JTAG platform USB cable
- Power cable to the board

Software Requirements

- Vivado Design Suite 2013.3
- Software Development Kit (SDK)14.7
- Customer provided drivers and application software required to generate PCI Express transactions

Resource Utilization

Table 1 illustrates the utilization for the default configurations of the PCIe CDMA subsystem configurations.

Table 1: **Included Design Files Description**

Device	Zynq-7000 Device Processing System	Block RAM	Registers	LUTs
Zynq ZC706	Yes	31	23,362	28,010
Kintex-7 KC705	No	31	33,327	39,213

Hardware System Specifics

The hardware design includes the following IP:

- Zynq-7000 Processing System (ZC706 design only)
- MIG AXI DDR3 Memory Controller (KC705 design only)
- AXI Memory Mapped to PCI Express (AXI PCI Express)
- AXI Central Direct Memory Access (CDMA)
- AXI Interface Block RAM Controller
- Block Memory Generator
- AXI4 Interconnect
- Processor System Reset

A PCI Express Root Complex or Host PC acts as the controller for this system. The FPGA design itself is configured as a PCI Express Endpoint device. The provided system is expected to provide you with a starting point for creating your own application or more complex systems. This system demonstrates commonly used features and capabilities, decreases initial development time, and allows you to focus on the custom portions of your design.

Included Systems

The reference design included with this application note has been implemented and verified in hardware on the ZC706 and KC705 development boards. The reference design is available at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=344579

Table 2 provides a description of the files provided in this package.

Table 2: **Design Files Description**

File or Directory	Description
readme.txt	Readme file describing the contained files.
kintexSubsystemFiles/	KC705 generation scripts and design files directory.
kintexGenerationScript.tcl	Subsystem generation script.
kintexDesignWrapper.v	Top-level design wrapper.
kintexConstraints.xdc	XDC constraints file.
kintexDdrConfiguration.prj	DDR3 configuration for the MIG DDR3 Memory controller.
zynqSubsystemFiles/	ZC706 generation scripts and design files directory.
zynqGenerationScript.tcl	Subsystem generation script
zynqDesignWrapper.v	Top-level design wrapper.
zynqConstraints.xdc	XDC constraints file.

The reference design checklist is shown in Table 3.

Table 3: **Reference Design Checklist**

Parameter	Description
General	
Developer name	Xilinx
Target devices (stepping level, ES, production, speed grades)	ZC706 device KC705 device
Source code provided	Yes
Source code format	Tcl, Verilog
Implementation	
Synthesis software tools/version used	Vivado Design Suite 2013.3
Implementation software tools/versions used	Vivado Design Suite 2013.3
Software development tools/versions used	SDK 14.7
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ZC706 board KC705 board Lecroy PCIe emulation platform

Generating the Subsystem

Use the following steps to generate either the ZC706 or KC705 systems:

1. Download the appropriate files for your device (ZC706 or KC705).

   The directory where you save these files is referred to as the <*Download Directory*>.
2. Navigate to the <*Download Directory*>.
3. Open the Vivado Design Suite and source the Tcl script to generate the subsystem using the following command:

   ```
   vivado -source <Download Directory>/<Device>SubsystemFiles/
   <Device>GenerationScript.tcl
   ```

   Generating the block diagram for the subsystem might take a few minutes.

   This script performs the following actions:

   - Adds, configures, and connects all the required IP.
   - Adds the top-level ports.
   - Generates the IP Integrator block diagram.
   - Adds the top-level Verilog and constraints files.
   - Generates the SDK project files if required.

After the block diagram is generated, you can:

4. View the default setting of the generated IP blocks, or perform additional customizations through the Vivado IP integrator interface.
5. Synthesize and implement the design by clicking **Generate Bitstream** in the Flow Navigator.

   Note: Bitstream generation of the subsystem is a lengthy process because of the complexity of the IP cores used in this subsystem.

System Overview

The PCIe® CDMA subsystem is a common building block that can be used in many applications. The subsystem itself can be used to perform DMA transactions, including Scatter Gather operations, between an external host device and internal AXI connected peripherals over PCI Express. To accomplish this, the subsystem uses AXI CDMA, AXI PCI Express, and AXI Interconnect IP cores. The subsystem can also perform dynamic address translation between the 64-bit address space of a host device and the 32-bit address space of the AXI Interconnect internal to the FPGA. To facilitate dynamic address translation during DMA Scatter Gather operations, a Translation BRAM was added to the subsystem and can be used to temporarily store PCI Express address translation vectors. For more information, see Initiating a DMA Scatter Gather Operation, page 14. The subsystem has been implemented in two similar designs, one targeting a Zynq-7000 device and one targeting a Kintex-7 FPGA, to demonstrate the use of the subsystem in each of these devices.

The PCIe CDMA subsystem is created as the *pcie_cdma_subsystem* design hierarchy. In the Vivado IP integrator block diagram view, double-click the instance to display the subsystem hierarchy, shown in Figure 1. The subsystem consists of various IP cores which can be individually selected and configured. A port description of this subsystem is provided in Table 4.

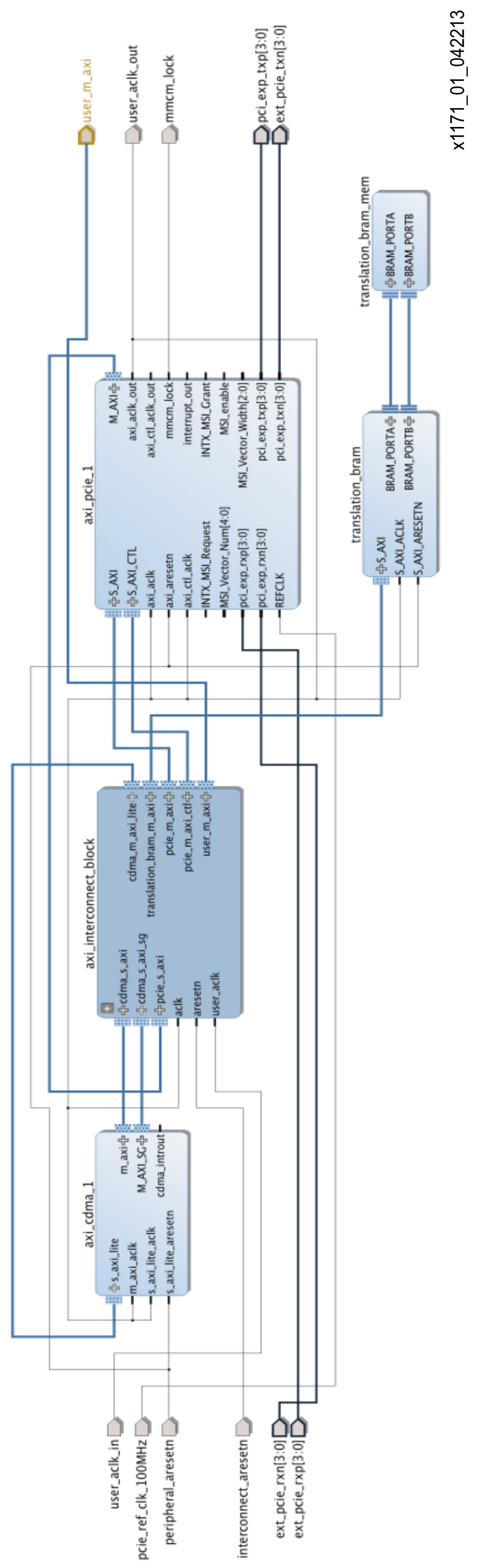

Figure 1: **PCIe CDMA Subsystem**

Table 4: **PCIe CDMA Subsystem Port Description**

Port Name	Direction	Description
pcie_ref_clk_100MHz	Input	This is the input reference clock used by the AXI PCI Express core. This clock is sourced by AXI PCI Express edge connector pins and should operate at 100 MHz. The AXI PCI Express core generates the transceiver and interface clocks required by the IP. The AXI PCI Express interface clock is used as the main system clock and operates at 125 MHz.
interconnect_aresetn	Input	This is the reset port used by the AXI Interconnect and is an active-Low reset.
peripheral_aresetn	Input	This is the reset port used by peripherals that are connected to the AXI Interconnect and is an active-Low reset.
mmcm_lock	Output	This port is generated by the AXI PCI Express IP and is asserted High when the MMCM internal to the core is locked.
user_m_axi	Output Interface Bus	This is an AXI bus interface that connects to the CDMA target peripheral. The target peripheral for this implementation is a DDR3 Memory controller.
user_aclk_out	Output	This is a 125 MHz interface clock that is generated in the AXI PCI Express IP. This clock is used as the main system clock for the AXI Interconnect and connected peripherals. This clock is brought out in order to drive the AXI interface clock for the target peripheral if required. If the target peripheral generates its own interface clock, this port should be left unconnected. Modifying the AXI PCI Express link speed and link width might change the frequency of this clock output (see the *LogiCORE IP AXI Bridge PCI Express Product Guide* (PG055) [Ref 5]).
user_aclk_in	Input	This port should be connected to the user interface clock for the target peripheral. If the target peripheral does not generate its own interface clock, this port and the AXI clock port (`ACLK`) of the connected peripheral should be driven by `user_aclk_out`.
ext_pcie_rxp	Input	AXI PCI Express RX transceiver port which should be driven by pins from the AXI PCI Express edge connector.
ext_pcie_rxn	Input	AXI PCI Express RX transceiver port which should be driven by pins from the AXI PCI Express edge connector.
ext_pcie_txp	Output	AXI PCI Express TX transceiver port which should drive pins on the AXI PCI Express edge connector.
ext_pcie_txn	Output	AXI PCI Express TX transceiver port which should drive pins on the PCI Express edge connector.

Reference System Address Maps

The AXI Interconnect IP provides the communication network for the PCI Express CDMA subsystem and peripherals. Table 5 and Table 6 show the Address Map used for the Zynq-7000 device and Kintex-7 device PCI Express-CDMA subsystems, respectively.

Table 5: **ZC706 AXI Address Map**

AXI Peripheral Port	Instance	Base Address	High Address	Size
ZYNQ_HP0_DDR	zynq_PS	0x00000000	0x3fffffff	1 gigabyte
AXI_PCIe_DM	axi_pcie_1 AXI:BAR1	0x80000000	0x807fffff	8 megabyte
AXI_PCIe_SG	axi_pcie_1 AXI:BAR0	0x80800000	0x80ffffff	8 megabyte
Translation BRAM	translation_bram	0x81000000	0x81000fff	32 kilobyte
AXI_PCIe_CTL	axi_pcie_1	0x81008000	0x8100bfff	16 kilobyte
AXI_CDMA_LITE	axi_cdma_1	0x8100c000	0x8100ffff	16 kilobyte
ZYNQ_HP0_OCM	zynq_PS	0xfffc0000	0xffffffff	256 kilobyte

Table 6: **KC705 AXI Address Map**

AXI Peripheral Port	Instance	Base Address	High Address	Size
AXI_DDR3_MEM	MIG	0x00000000	0x7fffffff	2 gigabyte
AXI_PCIe_DM	axi_pcie_1 AXI:BAR1	0x80000000	0x807fffff	8 megabyte
AXI_PCIe_SG	axi_pcie_1 AXI:BAR0	0x80800000	0x80ffffff	8 megabyte
Translation_BRAM	translation_bram	0x81000000	0x81000fff	32 kilobyte
AXI_PCIe_CTL	axi_pcie_1	0x81008000	0x8100bfff	16 kilobyte
AXI_CDMA_LITE	axi_cdma_1	0x8100c000	0x8100ffff	16 kilobyte

PCIe to AXI BAR Configuration and Addressing

The PCIe to AXI Bar configuration refers to the PCI Express BARs as seen on the PCI Express link, and it receives downstream PCI Express traffic from the PCI Express Root Complex or Host system. The PCI Express core configures PCIe:BAR0 to perform the address translation for PCIe:BAR0 to the base address of the Translation BRAM or address `0x81000000`. PCIe:BAR0 is configured to have a 64-kilobyte address space so that PCIe:BAR0 can be used to perform read/write transactions to the Translation BRAM, AXI_PCIe_CTL, and AXI_CDMA_LITE interfaces. To access these ports the calculation in Equation 1 should be used to construct the address of the desired interface.

<PCIe:BAR0 Host Memory Base Address> + <Address Offset of the desired Interface> + <Register Offset within that Interface> *Equation 1*

The PCIe:BAR0 Host Memory Base Address is assigned when PCIe:BAR0 is recognized by the Root Complex or Host system. This address should be captured and accessed through customer provided application software and drivers. On a Linux system, the following command can be used to obtain the PCIe:BAR0 Host Memory Base Address from the host machine:

```
lspci -v -d 10ee:*
```

The Address Offset for the Translation BRAM, AXI_PCIe_CTL, and AXI_CDMA_LITE interfaces is the address offset from 0x81000000 required for the interface. The offset for each interface is shown in Table 7.

Table 7: **PCIe:BAR0 Address Offset for the accessible Interfaces**

Interface	Interface Address Offset
Translation_BRAM	0x00000000
AXI_PCIe_CTL	0x00008000
AXI_CDMA_LITE	0x0000c000

The register offset within each interface is the register offset from 0x00000000 to the register you are accessing. The register offset within each interface is defined by the register specification for the interface. The key portions of the register specifications for the AXI_CDMA_LITE and AXI_PCIe_CTL interfaces have been included in this application note for reference. A full description of the register specification for each interface can be obtained in the product guides available for the AXI CDMA and AXI Memory Mapped to PCI Express cores. See *LogiCORE IP AXI Central Direct Memory Access Product Guide* (PG034) [Ref 6], and *LogiCORE IP AXI Bridge for PCI Express Product Guide* (PG055) [Ref 5].

The following examples illustrate how to perform this calculation. The examples assume the Host System has enumerated PCIe:BAR0 and assigned it a base address of 0x00000000_def00000.

Example 1

Desired Action: The Host System performs a memory write to the Translation BRAM at address 0x0d10. The Translation BRAM can be used to store the translation vectors required for Dynamic Address Translation during DMA operations.

Calculation:

PCIe:BAR0 Base Address	0x00000000_def00000
Translation_BRAM offset	0x00000000
Translation_BRAM Register offset	0x00000d10

Action Performed by the Host System: PCI Express writes to address 0x00000000_def00d10.

Example 2

Desired Action: The Host System performs a memory write to the AXI_PCIe_CTL interface at address 0x020c. This address corresponds to the AXI:BAR0 Lower Address Translation Register which is used to perform Dynamic Address Translation updates.

Calculation:

PCIe:BAR0 Base Address	0x00000000_def00000
Translation_BRAM offset	0x00008000
Translation_BRAM Register offset	0x0000020c

Action Performed by the Host System: PCI Express writes to address 0x00000000_def0820c.

Example 3

Desired Action: The Host System performs a memory read from the AXI_CDMA_LITE interface at address 0x0004. This address corresponds to the CDMA Status register and can be used to check the status of the DMA Engine.

Calculation:

PCIe:BAR0 Base Address	0x00000000_def00000
Translation_BRAM offset	0x0000c000
Translation_BRAM Register offset	0x00000004

Action Performed by the Host System: PCI Express reads to address 0x00000000_def0c004.

There is no requirement that the PCIe AXI Master port performs read/write transaction to the `AXI_PCIe_SG`, `AXI_PCIe_DM`, or `AXI_DDR3_MEM` port, because the DMA performs the reads/writes to these AXI Slave ports. Up to two additional PCIe BARs can be added to the AXI PCI Express IP if additional connectivity is needed.

AXI to PCIe BAR Configuration and Addressing

The AXI to PCIe Bar configuration refers to the AXI interfaces on the AXI PCI Express IP as seen from the FPGA AXI Interconnect and ultimately generates upstream PCI Express traffic to the PCI Express Root Complex or Host system. AXI:BAR0 and AXI:BAR1 have been configured to generate upstream PCI Express traffic and perform dynamic address translation between a 64-bit Root Complex or Host device and the 32-bit AXI address space.

AXI:BAR0 is designated as the Scatter Gather port for Scatter Gather DMA transactions (`AXI_PCIe_SG` port). Through this port the DMA reads and processes descriptors from the PCI Express Root Complex device. By default, this AXI:BAR0 is configured with an 8-megabyte address space.

AXI:BAR1 is designated as the Data Mover port for DMA transactions (`AXI_PCIe_DM` port). Through this port the DMA reads/writes data from/to the PCI Express Root Complex device. By default, this AXI:BAR1 is configured with an 8-megabyte address space, because this is the maximum size of data transfers allowable by the AXI CDMA IP.

While each AXI:BAR is capable of 64-bit dynamic address translation the default translation for each AXI:BAR is initialized to 0x00000000_a0000000 for AXI:BAR0 and 0x00000000_c0000000 for AXI:BAR1. This means that a read/write on AXI:BAR0 (AXI address: 0x80800000) translates into a PCI Express read/write transaction to the Root Complex at a base address of 0x00000000_a0000000. The translation for each AXI:BAR can be independently modified by performing read/write transactions to the control registers that are accessible through the `AXI_PCIe_CTL` port. Table 8 shows the address map for the AXI to PCIe translation registers for AXI:BAR0 and AXI:BAR1.

Note: In Table 8, the listed addresses are offset from the AXI_PCIe_CTL Base Address (0x81008000).

Table 8: **Address Map for the AXI to PCIe Address Translation Registers**

Offset	Bits	Register Name	Description
0x208	31-0	AXIBAR2PCIEBAR_0U	AXI:BAR0 Upper Address Translation register for the `AXI_PCIe_SG` port. This provides translation for bits [63:32] in the Host address space. Initialized to 0x00000000.
0x20c	31-0	AXIBAR2PCIEBAR_0L	AXI:BAR0 Lower Address Translation register for the `AXI_PCIe_SG` port. This provides translation for bits [31:0] in the Host address space. Initialized to 0xa0000000.
0x210	31-0	AXIBAR2PCIEBAR_1U	AXI:BAR1 Upper Address Translation register for the `AXI_PCIe_DM` port. This provides translation for bits [63:32] in the Host address space. Initialized to 0x00000000.
0x214	31-0	AXIBAR2PCIEBAR_1L	AXI:BAR1 Lower Address Translation register for the `AXI_PCIe_DM` port. This provides translation for bits [31:0] in the Host address space. Initialized to 0xc0000000

The Lower Address Translation registers are limited by the address width of each AXI:BAR. Because both the AXI:BAR0 (`AXI_PCIe_SG`) and AXI:BAR1 (`AXI_PCIe_DM`) ports are configured with an 8 megabyte address space, the least significant 23 bits (bits [22:0]) of the Lower Address Translation Registers (offsets 0x20c and 0x214) should always be zero.

Up to four additional upstream AXI to PCIe BARs can be added to the PCI Express core if additional connectivity is required for your application. For additional information or a complete address map for the AXI Memory Mapped to PCI Express IP, see *LogiCORE IP AXI Bridge for PCI Express Product Guide (PG055)* [Ref 5].

Using the Subsystem to Initiate PCI Express Data Transfers

Scatter Gather is a mechanism that allows for automated DMA transfer scheduling through a pre-programmed instruction list of transfer descriptors. Using your custom software application, this instruction list should be programmed into a memory-resident data structure on the Root Complex side of the PCI Express connection. This list of instructions is organized into a transfer descriptor chain. Each descriptor describes a single data transfer and has an address pointer to the next sequential descriptor to be processed. The last descriptor in the chain generally points back to the first descriptor but this is not required.

The CDMA Scatter Gather engine processes the descriptors in the chain sequentially until the Tail Descriptor is reached. By reallocating descriptors after the DMA Engine has processed them and updating the CDMA Tail Descriptor address, the software application can create continuous DMA transactions over the PCI Express link. Creating a descriptor chain in memory on the Root Complex device must be done through customer-provided driver and software-level support. The software is not provided with this application note, but can be developed and customized for your platform, usage, and system requirements.

Scatter Gather Transfer Descriptor Definition

Prior to initiating CDMA Scatter Gather operations, the software application run on the Root Complex must set up a transfer descriptor chain. After the AXI CDMA begins a Scatter Gather operation, it fetches, processes, and then updates the descriptors. A transfer descriptor consists of eight 32-bit words (32 bytes). The descriptor contains the control and status information needed for a single DMA transfer plus address linkage to the next sequential descriptor. Each descriptor can define a single DMA transfer of up to 8,388,607 bytes of data. A descriptor chain is defined as a series of descriptors that are sequentially linked through the address linkage built in the descriptor format. The AXI CDMA SG Engine traverses the descriptor chain following the linkage until the Tail Descriptor has been processed. Table 9 provides a description of the descriptors that must be created in memory by your software application or driver running on the Root Complex.

Note: The Transfer Descriptors must be aligned with 16 32-bit word alignment (64-byte aligned). Valid example offsets are 0x00, 0x40, 0x80, and 0xc0.

Table 9: **Transfer Descriptor Word Summary**

Address Space Offset(1)	Name	Descriptor
0x00	NXTDESC_PNTR	Next Descriptor Pointer Bits[5:0] get truncated
0x04	RESERVED	N/A
0x08	SA	Source Address 32-bit
0x0c	RESERVED	N/A
0x10	DA	Destination Address 32-bit
0x14	RESERVED	N/A
0x18	CONTROL	Transfer Control BTT[22:0]
0x1c	STATUS	Transfer Status TransferCompleted(31) ErrorCodes[30:28]

Notes:

1. The Address Space Offset is relative to the address of the first word of the Transfer Descriptor in system memory.

A description of each portion of the descriptor is as follow:

- **Transfer Descriptor NXTDESC_PNTR (Next Descriptor Pointer – Offset 0x00)**: This word provides the address pointer to the first word of the next transfer descriptor in the descriptor chain. This address must be 64-byte aligned. Valid offsets for Descriptor Pointers are 0x00, 0x40, 0x80, 0xc0. This address should be offset from the `AXI_PCIe_SG` port base address of `0x80800000` and is translated through the AXI PCI Express core to the Root Complex address space. Table 10 provides a description of the bits within the Transfer Descriptor NXTDESC_PNTR.

Table 10: **Transfer Descriptor NXTDESC_PNTR Word Details**

Bits	Field Name	Description
31 to 6	NxtDescPntr	This field contains the address pointer (most significant 26 bits) to the first word of the next transfer descriptor to be processed.
5 to 0	Reserved	These bits are fixed to zeros to ensure the address of the next descriptor is 64-byte aligned.

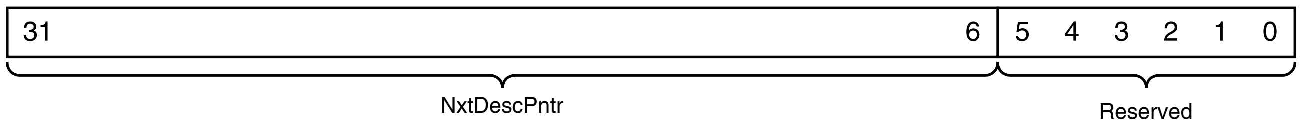

Figure 2: **Transfer Descriptor NXTDESC_PNTR Word**

- **Transfer Descriptor SA Word (Source Address – Offset 0x08)**: This word provides the starting address for the data read operations of the associated DMA transfer. The address value can be at any byte offset.
- **Transfer Descriptor DA Word (Destination Address – Offset 0x10)**: This word provides the starting address for the data write operations of the associated DMA transfer. Because data realignment is disabled (to conserve FPGA resources) and the DMA is configured with a 64-bit data width, the least significant 3 bits of the destination address must be the same as the least significant 3 bits of the source address. The least significant 3 bits are referred to as the byte offset within a 64-bit data beat.

  Note: If required by your application, Data Realignment can be enabled. This results in increased resource utilization but allows the destination address to specify any byte offset regardless of the source address byte offset.
- **Transfer Descriptor Control Word (Control – Offset 0x18)**: This word specifies the Bytes to Transfer (BTT) of the associated DMA transfer. Valid values for this field are 1–8,388,607 (0x007fffff). Table 11 provides a description of the bits within the Transfer Descriptor Control Word.

Table 11: **Transfer Descriptor CONTROL Word Details**

Bits	Field Name	Description
31 to 23	Reserved	These bits are reserved and should be set to zero.
22 to 0	BTT	This field specifies the desired number of bytes to DMA from the Source Address to the Destination Address. A maximum of 8,388,607 (0x7fffff) bytes of data can be specified by this field for the associated transfer. A value of zero is not allowed and causes a DMA internal error.

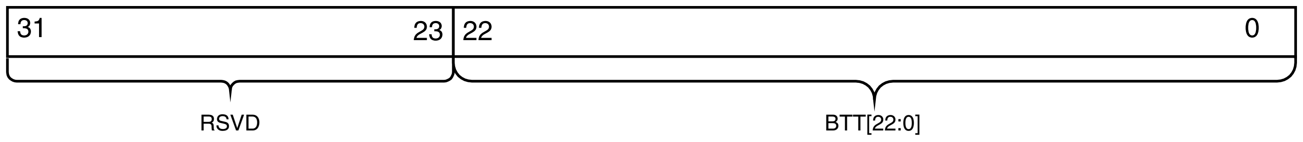

Figure 3: **Transfer Descriptor CONTROL Word**

- **Transfer Descriptor Status Word (Status – Offset `0x1c`)**: This value provides a status to the software application running on the Root Complex regarding the execution of the Transfer Descriptor by the AXI CDMA. This status word should be zeros when the transfer descriptor is programmed by the software application. When the AXI CDMA SG Engine has completed execution of the transfer descriptor, the status word is updated with the status of the processed DMA transaction by the SG Engine. This allows the software application to track the progress of the DMA engine as it processes descriptors. After a descriptor has been processed, as identified by the Status word, the software application running on the root complex can reallocate the descriptor for AXI CDMA use. Table 12 provides a description of the bits within the Transfer Descriptor Status Word.

Table 12: **Transfer Descriptor STATUS Word Details**

Bits	Field Name	Description
31	Cmplt	Transfer Complete. This bit indicates to the software application that the CDMA Engine has completed the transfer as described by the associated descriptor. The software application can manipulate any descriptor with the Completed bit set to 1. 0 = Descriptor not completed and should not be modified by the software application. 1 = Descriptor completed and can be modified by the software application. If the CDMA SG Engine fetches a descriptor with this bit set to 1, the descriptor is considered a stale descriptor. An SGIntErr is flagged in the AXI CDMA Status Register and the AXI CDMA engine halts with no update to the descriptor.
30	DMADecErr	DMA Decode Error. This bit indicates that an AXI decode error was received by the AXI CDMA Data Mover. This error occurs if the Data Mover issues an address that does not have a mapping assignment to a slave device. This error condition causes the AXI CDMA to gracefully halt. 0 = No CDMA Decode Errors. 1 = CDMA Decode Error received. CDMA Engine halts at this descriptor.
29	DMASlvErr	DMA Slave Error. This bit indicates that an AXI slave error response was received by the AXI CDMA Data Mover during the AXI transfer (read or write) associated with this descriptor. This error condition causes the AXI CDMA to gracefully halt. 0 = No CDMA Slave Errors. 1 = CDMA Slave Error received. CDMA Engine halts at this descriptor.
28	DMAIntErr	DMA Internal Error. This bit indicates that an internal error was encountered by the AXI CDMA Data Mover on the data transport channel during the execution of this descriptor. This error can occur if a 0 value BTT is fed to the AXI Data Mover, or the Data Mover has an internal processing error. A BTT of 0 only happens if the BTT field in the transfer descriptor Control word is programmed with a value of 0. This error condition causes the AXI CDMA to gracefully halt. 0 = No CDMA Internal Errors. 1 = CDMA Internal Error detected. CDMA Engine halts at this descriptor.
27 to 0	Reserved	These bits are reserved and should be set to 0.

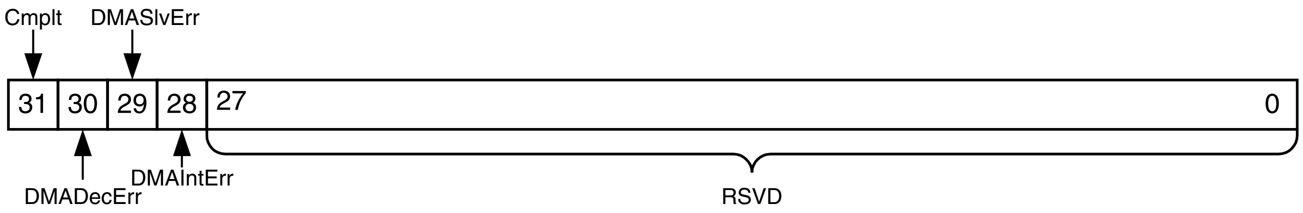

Figure 4: **Transfer Descriptor Status Word**

By creating and reallocating a descriptor ring, the software application running on the host device can create a chain of continuous PCI Express DMA operations. The software application can then track the status and completion of associated DMA data transfers by checking the status register of each descriptor. In this subsystem, there are two types of descriptors that are used: Address Translation Descriptors and Target Data Transfer Descriptor. Both of these descriptor types follow the structure outlined above and facilitate data transfers between peripherals.

Note: It is important to note that the descriptor addresses are and must be described in terms of the address space as seen from the DMA Engine. This means that addresses used in transfer descriptors must be defined by the AXI Interconnect Address Mapping rather than address space defined by the Host system.

Address Translation Descriptors

Address Translation Descriptors are used to update the AXI to PCIe Translation vectors through writes to the `AXI_PCIe_CTL` port. This means that the Address Translation Descriptors generally have:

- A Source Address corresponding to the Translation BRAM (address range 0x8100000-81007fff).
- A Destination Address corresponding to the `AXI_PCIe_CTL` port AXI:BAR1 address translation registers (addresses 0x81008210 and 0x81008214).
- A Control value corresponding to an 8 byte data transfer (0x00000008).

When processed, the DMA Engine copies an 8 byte (64-bit) translation vector from the Translation BRAM into the translation registers that are accessible through the `AXI_PCIe_CTL` port.

Note: It is important that the translation vectors within the block RAM are updated before the descriptor is processed and that it corresponds to the current set of descriptors being processed. Incorrectly setting or failing to update the translation vectors can result in accessing an unintended region of memory.

Target Data Transfer Descriptors

Target Data Transfer Descriptors are used to transfer data between the PCIe Root Complex device and a target peripheral, which for this application is a DDR3 memory controller. Transfer descriptors of this type generally have Source and Destination addresses which correspond to either the `AXI_PCIe_DM` (0x80000000) or `DDR3 Memory` (0x00000000) port. The DMA Control register can correspond to data transfers between 1 and 8,388,607 (0x007fffff) bytes. When processed, this descriptor copies the specified amount of data between the target peripheral and the PCIe Root Complex device.

Note: It is important that the PCIe translation vector has been updated correctly prior to Target Data Transfer Descriptors being processed. This is accomplished by updating the translation vectors directly or through the use of an Address Translation Descriptor as described above.

Initiating a DMA Scatter Gather Operation

The basic steps to initiate a DMA Scatter Gather operation for the PCIe-CDMA subsystem are described here and illustrated through the use of an example descriptor chain.

1. Set the DMA to Scatter Gather Mode, and enable the appropriate interrupts by writing to the AXI_CDMA_LITE control register at an AXI address 0x8100c000 or a PCIe:BAR0 offset of 0x0000c000.
2. Create a descriptor chain in memory on the Root Complex or Host system. This descriptor chain likely consists of both Address Translation Descriptors and Target Data Transfer Descriptors.
3. Update the PCIe Translation vector for the `AXI_PCIe_SG` port by writing to the AXI:BAR0 translation registers at AXI address 0x81008208-0x8100820c (`AXI_PCIE_CTL` port) or PCIe:BAR0 offset 0x00008208-0x0000820c. The value written to these registers should correspond to the base address of an 8-megabyte Host memory region which contains the descriptor chain that was created in Host system memory.
4. Write the appropriate translation vectors to the Translation BRAM at AXI address range 0x81000000-0x81007fff or PCIe:BAR0 address offset range 0x00000000-0x00007fff. The values written to the Translation BRAM should correspond to 8 megabytes regions of Host memory which contain the data that is transferred by the DMA Engine.
5. Write a valid pointer to the DMA CURDESC_PNTR register at AXI address 0x8100c008 or PCIe:BAR0 offest 0x0000c008.
6. Write a valid pointer to the DMA TAILDESC_PNTR register at AXI address 0x8100c010 or PCIe:BAR0 offset 0x0000c010. Writing this register starts the DMA fetching and processing descriptors.
7. The DMA Scatter Gather operation continues until the descriptor at TAILDESC_PNTR is processed or a descriptor that has already been completed is reached.

The following illustrate these steps. This example assumes that the Host system has enumerated PCIe:BAR0 and assigned it to a Host memory address of 0x00000000_def00000.

1. To set the DMA to Scatter Gather Mode, perform a 32-bit PCIe write from the Root Complex to PCIe:BAR0 address 0x00000000_def0c0000 with a data value of 0x0a000100.
2. To view the descriptor chain that consists of four descriptors, which is used for this illustration, see Figure 5. These descriptors start at address 0x00000001_00000000 in the Host system memory.
3. To update the PCIe Translation vector for the `AXI_PCIe_SG` port, perform a 64-bit PCIe write from the Root Complex to PCIe:BAR0 address 0x00000000_def08208 with a data value of 0x00000001_00000000. This value corresponds to the memory offset of the DMA descriptors on the Root Complex device.
4. To write the appropriate translation vectors to the Translation BRAM:
    a. Perform a 64-bit PCIe write from the Root Complex to PCIe:BAR0 address 0x00000000_def00000 with a data value of 0x0000ccc0_c0000000. This value corresponds to the address translation vector which is updated during Descriptor 1 and applied to Descriptor 2.
    b. Perform a 64-bit PCIe write from the Root Complex to PCIe:BAR0 address 0x00000000_def00008 with a data value of 0x0000ddd0_d0000000. This value corresponds to the address translation vector which is updated during Descriptor 3 and applied to Descriptor 4.
5. To write a valid pointer to the DMA CURDESC_PNTR register, perform a 32-bit PCIe write from the Root Complex to PCIe:BAR0 address 0x00000000_def0c008 with a data value of 0x80800000. This value corresponds to the AXI_PCIe_SG Base Address + Root Complex Descriptor Offset from 0x00000001_00000000. This AXI address is translated to the Root

Complex address that was written in step 2, and allows the CDMA Scatter Gather port to fetch the descriptors contained in the Host memory at address 0x00000001_00000000.

6. To write a valid pointer to the DMA TAILDESC_PNTR register, perform a 32-bit PCIe write from the Root Complex to PCIe:BAR0 address 0x00000000_def0c010 with a data value of 0x808000c0. This value corresponds to the AXI_PCIe_SG Base Address + Root Complex Descriptor Offset from 0x00000001_00000000. Similar to step 5 this address is translated to the Root Complex Device memory at address 0x00000001_000000c0.

   Writing this register initiates the DMA Scatter Gather operation.
7. The DMA Scatter Gather operation continues until the descriptor at the TAILDESC_PNTR (address 0x00000001_000000c0 of Host memory) is processed.

The descriptor chain for this illustration is shown in Figure 5. This descriptor chain is created in memory on the Root Complex device. Descriptors 1 to 4 are located at the 64-bit address of 0x00000001_00000000 at offsets of 0x00, 0x40, 0x80 and 0x0c respectively. While these descriptors are not required to be contiguous, they should be contained within an 8 megabyte region which corresponds to the width of the `AXI_PCIe_SG` port. As noted in Figure 5, the address translation for the `AXI_PCIe_SG` port is also set to `0x00000001_00000000`. This means that when the DMA reads descriptors from or writes status updates to the `AXI_PCIe_SG` port (address range 0x80800000-0x80ffffff), the address is translated to a Root Complex PCIe address in the range of 0x00000001_00000000-0x00000001_007fffff where the descriptors can be accessed. Figure 5 also identifies the translation for the `AXI_PCIe_DM` port as being set dynamically by descriptors 1 and 3. This means that descriptors 1 and 3 are Address Translation Descriptors while descriptors 2 and 4 are Target Data Transfer Descriptors.

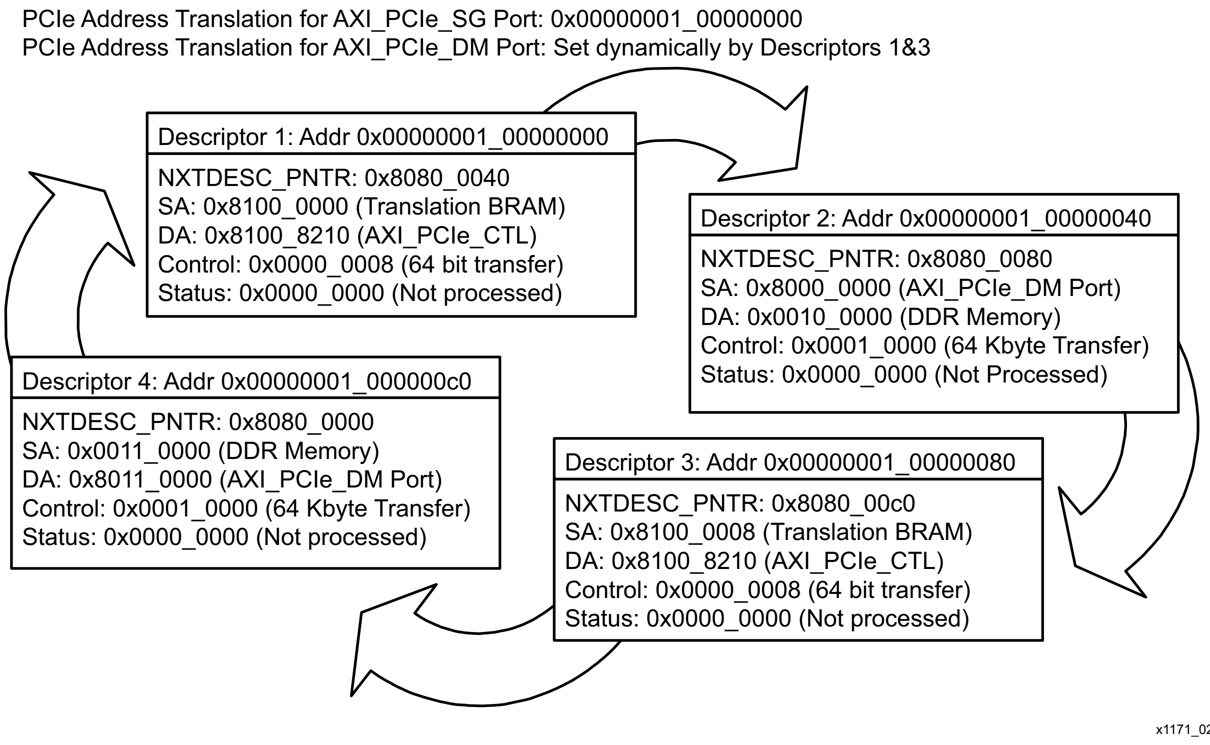
PCIe Address Translation for AXI_PCIe_SG Port: 0x00000001_00000000
PCIe Address Translation for AXI_PCIe_DM Port: Set dynamically by Descriptors 1&3
Descriptor 1: Addr 0x00000001_00000000
NXTDESC_PNTR: 0x8080_0040
SA: 0x8100_0000 (Translation BRAM)
DA: 0x8100_8210 (AXI_PCIe_CTL)
Control: 0x0000_0008 (64 bit transfer)
Status: 0x0000_0000 (Not processed)
Descriptor 2: Addr 0x00000001_00000040
NXTDESC_PNTR: 0x8080_0080
SA: 0x8000_0000 (AXI_PCIe_DM Port)
DA: 0x0010_0000 (DDR Memory)
Control: 0x0001_0000 (64 Kbyte Transfer)
Status: 0x0000_0000 (Not Processed)
Descriptor 3: Addr 0x00000001_00000080
NXTDESC_PNTR: 0x8080_00c0
SA: 0x8100_0008 (Translation BRAM)
DA: 0x8100_8210 (AXI_PCIe_CTL)
Control: 0x0000_0008 (64 bit transfer)
Status: 0x0000_0000 (Not processed)
Descriptor 4: Addr 0x00000001_000000c0
NXTDESC_PNTR: 0x8080_0000
SA: 0x0011_0000 (DDR Memory)
DA: 0x8011_0000 (AXI_PCIe_DM Port)
Control: 0x0001_0000 (64 Kbyte Transfer)
Status: 0x0000_0000 (Not processed)
x1171_02_042213

Figure 5: **Descriptors**

- After the DMA TAILDESC_PNTR register is written, the DMA Scatter Gather port begins fetching the descriptors at the address specified in step 5 or AXI address 0x80800000. This address corresponds to the `AXI_PCIe_SG` port or PCIe Root Complex address 0x00000001_00000000 as a result of the address translation.

- The DMA engine fetches and buffers descriptors as it begins processing them through the DMA Data Mover port. When Descriptor 1 is processed, it copies 8 bytes (64-bits) of data from Address 0x81000000 in the Translation BRAM to address 0x81008210 in the AXI_PCIe_CTR registers. This register corresponds to the translation vector for the `AXI_PCIe_DM` port. Subsequent transactions that access the `AXI_PCIe_DM` port (AXI base address of 0x80000000) are thus translated to a PCIe Root Complex address of 0x0000ccc0_c0000000.
- After Descriptor 1 is completed the DMA Scatter Gather port updates the status for this descriptor and the DMA Data Mover port moves on to process the second descriptor.
- The second descriptor then moves 64 kilobytes of data from the Root Complex device at address 0x0000ccc0_c0000000 (after AXI to PCIe Translation) to the DDR memory at address 0x00100000.
- Following the data move, the status register for Descriptor 2 is updated with a completion status. Figure 6 identifies that Descriptors 1 and 2 have been completed, while Descriptors 3 and 4 have not yet been processed.

Note: It is important that Descriptors 3 and 4 are not modified after the DMA has been initiated because the DMA might have already buffered the descriptors inside the DMA Engine.

Because Descriptors 1 and 2 have been completed as shown in their status registers, they can be reallocated for use in the system. To reuse these descriptors, update their content, initialize the status register to zero, and update the DMA TAILDESC_PNTR register.

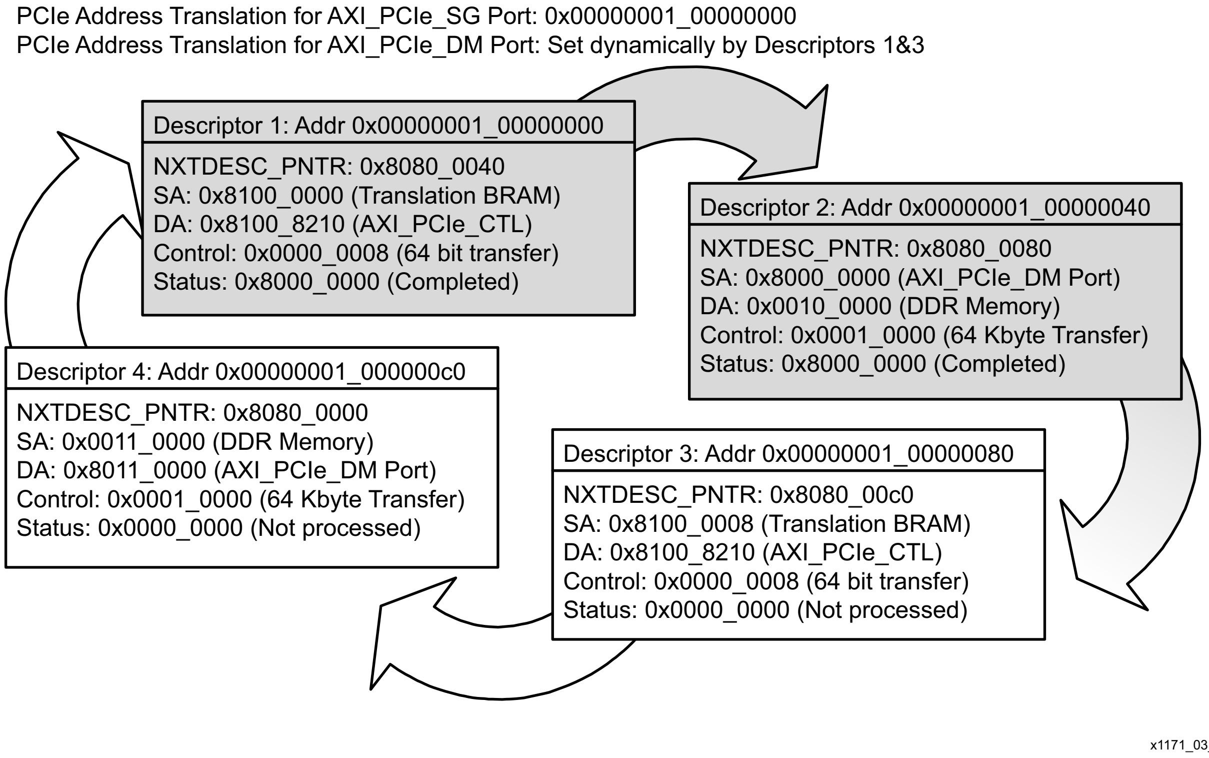
PCIe Address Translation for AXI_PCIe_SG Port: 0x00000001_00000000
PCIe Address Translation for AXI_PCIe_DM Port: Set dynamically by Descriptors 1&3
Descriptor 1: Addr 0x00000001_00000000
NXTDESC_PNTR: 0x8080_0040
SA: 0x8100_0000 (Translation BRAM)
DA: 0x8100_8210 (AXI_PCIe_CTL)
Control: 0x0000_0008 (64 bit transfer)
Status: 0x8000_0000 (Completed)
Descriptor 2: Addr 0x00000001_00000040
NXTDESC_PNTR: 0x8080_0080
SA: 0x8000_0000 (AXI_PCIe_DM Port)
DA: 0x0010_0000 (DDR Memory)
Control: 0x0001_0000 (64 Kbyte Transfer)
Status: 0x8000_0000 (Completed)
Descriptor 3: Addr 0x00000001_00000080
NXTDESC_PNTR: 0x8080_00c0
SA: 0x8100_0008 (Translation BRAM)
DA: 0x8100_8210 (AXI_PCIe_CTL)
Control: 0x0000_0008 (64 bit transfer)
Status: 0x0000_0000 (Not processed)
Descriptor 4: Addr 0x00000001_000000c0
NXTDESC_PNTR: 0x8080_0000
SA: 0x0011_0000 (DDR Memory)
DA: 0x8011_0000 (AXI_PCIe_DM Port)
Control: 0x0001_0000 (64 Kbyte Transfer)
Status: 0x0000_0000 (Not processed)
x1171_03_042213

Figure 6: **Descriptions in Process**

Similar to Descriptors 1 and 2, when Descriptor 3 is processed it updates the translation vector for the `AXI_PCIe_DM` port to 0x0000ddd0_d0000000 and thus Descriptor 4 transfers data from the DDR Memory to the Root Complex device at address 0x0000ddd0_d0000000.

For more information on using the AXI Central Direct Memory Access IP, see *LogiCORE IP AXI Central Direct Memory Access Product Guide(PG034)* [Ref 6]. This product guide contains a full description of the core including performance, utilization, usage, and a full register map for the AXI4-Lite control interface.

Hardware Bring-Up and Verification

The following steps describe how to download the generated bitstream and system configuration files to certify that the system is working in hardware. This system cannot be verified in hardware without the ability to generate targeted AXI PCI Express transactions. This can be done through the creation of a software application and driver, or through the use of AXI PCI Express emulation test equipment. Hardware verification for this subsystem was accomplished using a Lecroy PCI Express emulation platform.

1. Connect a USB JTAG cable from your computer to the development board.
2. Load the bitstream to the development platform.
   a. From the Vivado Flow Navigator, select **Open Hardware Session**.

Steps 3-5 apply only to the ZC706 setup. For the KC705 implementations skip to step 6.

3. Launch the Software Development Kit (SDK) from the Vivado Design Suite as follows:
   a. Select **File** > **Export** > **Export Hardware for SDK**.
   b. Use the following settings:

      Source: **design_1.bd**

      Export to: **<Local to Project>**

      Workspace: **<Local to Project>**

      Check **Include bitstream**

      Check **Export Hardware**

      Check **Launch SDK**
   c. Click **OK**.
4. Open the XMD Console from the menu options by selecting **Xilinx Tools > XMD Console**.
5. Enter the following XMD commands into the XMD Console:

```
connect arm hw            # Connects the XMD Consol to the ARM Processor
source ps7_init.tcl       # Adds the TCL setup commands
ps7_init                  # Issue the setup command for the Zynq PS
init_user                 # Enables the Zynq user interfaces
mwr 0x00100000 0x01234567 # Perform a memory write to the DDR3
mrd 0x00100000            # Perform a memory read from the DDR3 memory
```

6. Descriptor chain setup:
   a. Create a descriptors chain in memory as described in Using the Subsystem to Initiate PCI Express Data Transfers, page 10.
7. PCI Express transactions:
   a. Perform a 64-bit read from the DMA command and status registers (addr:PCIe:BAR0_BASE_ADDR+0x0000c000).
   b. Perform a 32-bit write to the DMA command register to put it into Scatter Gather Mode (addr:PCIe:BAR0_BASE_ADDR+0x0000c000; data: 0x0a000100).
   c. Perform a 64-bit read from the DMA command and status registers (addr:PCIe:BAR0_BASE_ADDR+0x0000c000).
   d. Write Translation vectors to BRAM (addr:PCIe:BAR0_BASE_ADDR+BRAM offset).
   e. Perform a 64-bit write to update the PCI Express Translation for the SG port to point to the descriptor chain memory address (addr:PCIe:BAR0_BASE_ADDR+0x00008208; data:SG_DESC_BASE_ADDR).
   f. Perform a 32-bit write to the DMA CURDESC_PNTR register with the address of the first descriptor in the chain as seen by the DMA Engine (addr:PCIe:BAR0_BASE_ADDR+0x0000c0008, data:AXI_PCIe_SG_BASE_ADDR+CURDESC_PNTR offset).

g. Perform a 32-bit write to the DMA TAILDESC_PNTR register with the address of the last descriptor in the chain as seen by the DMA Engine (addr:PCIe:BAR0_BASE_ADDR+0x0000c0010, data:AXI_PCIe_SG_BASE_ADDR+TAIL_DESC_PNTR offset).

8. DMA Engine begins processing descriptors after the TAILDESC_PNTR is written.

   After each descriptor is processed, the status register in the descriptor is updated.

9. After the DMA has reached the tail pointer it stops processing descriptors and waits for the tail pointer to be updated.

CDMA Reference Table

The AXI CDMA core register space is summarized in Table 13. The AXI CDMA registers are memory mapped into non-cacheable memory space.

Table 13: **AXI CDMA Register Summary**

Address Space Offset[1]	Name	Description[2]
00h	CDMACR	CDMA Control.
04h	CDMASR	CDMA Status.
08h	CURDESC_PNTR	Current Descriptor Pointer.
0Ch	Reserved	N/A
10h	TAILDESC_PNTR	Tail Descriptor Pointer.
14h	Reserved	N/A
18h	SA	Source Address.
1Ch	Reserved	N/A
20h	DA	Destination Address.
24h	Reserved	N/A
28h	BTT	Bytes to Transfer.

Notes:

1. The Address Space is offset from the AXI_CDMA_LITE Base Address.
2. For a more detailed description, see the *LogiCORE IP AXI Central Direct Memory Access Product Guide* (PG034) [Ref 6].

PCI Express Reference Tables

The PCIe AXI Control register space is summarized in Table 14.

Table 14: **Register Memory Map**

Accessibility	Offset	Contents	Location
RO - EP, R/W - RC	0x000 - 0x124	PCIe Configuration Space Header	Part of the integrated PCIe configuration space.
RO	0x128	Vendor-Specific Enhanced Capability (VSEC) Capability	VSEC of the integrated PCIe configuration space.
RO	0x12C	VSEC Header	

Table 14: **Register Memory Map *(Cont'd)***

Accessibility	Offset	Contents	Location
RO	0x130	Bridge Info	AXI bridge defined memory-mapped register space.
RO - EP, R/W - RC	0x134	Bridge Status and Control	
R/W	0x138	Interrupt Decode	
R/W	0x13C	Interrupt Mask	
RO - EP, R/W - RC	0x140	Bus Location	
RO	0x144	Physical-Side Interface (PHY) Status/Control	
RO - EP, R/W - RC	0x148	Root Port Status/Control	
RO - EP, R/W - RC	0x14C	Root Port MSI Base 1	
RO - EP, R/W - RC	0x150	Root Port MSI Base 2	
RO - EP, R/W - RC	0x154	Root Port Error FIFO Read	
RO - EP, R/W - RC	0x158	Root Port Interrupt FIFO Read 1	
RO - EP, R/W - RC	0x15C	Root Port Interrupt FIFO Read 2	
RO	0x160 - 0x1FF	Reserved (zeros returned on read)	
RO	0x200	VSEC Capability 2	
RO	0x204	VSEC Header 2	
R/W	0x208 - 0x234	AXI Base Address Translation Configuration Registers	AXI bridge defined memory-mapped space.
RO	0x238 - 0xFFF	Reserved (zeros returned on read)	

The PCIe Address Translation register address mapping is shown in Table 15.

Table 15: **AXI Base Address Translation Configuration Registers**

Offset	Bits	Register Mnemonic
0x208	31-0	AXIBAR2PCIEBAR_0U
0x20C	31-0	AXIBAR2PCIEBAR_0L
0x210	31-0	AXIBAR2PCIEBAR_1U
0x214	31-0	AXIBAR2PCIEBAR_1L
0x218	31-0	AXIBAR2PCIEBAR_2U
0x21C	31-0	AXIBAR2PCIEBAR_2L
0x220	31-0	AXIBAR2PCIEBAR_3U
0x224	31-0	AXIBAR2PCIEBAR_3L
0x228	31-0	AXIBAR2PCIEBAR_4U

Table 15: **AXI Base Address Translation Configuration Registers *(Cont'd)***

Offset	Bits	Register Mnemonic
0x22C	31-0	AXIBAR2PCIEBAR_4L
0x230	31-0	AXIBAR2PCIEBAR_5U
0x234	31-0	AXIBAR2PCIEBAR_5L

References

The following documents provide additional information related to this subsystem:

1. *ZC706 Evaluation Board for the Zynq-7000 XC7Z045 All Programmable SoC User Guide* (UG954)
2. *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585)
3. *KC705 Evaluation Board for the Kintex-7 FPGA User Guide* (UG810)
4. *Kintex-7 FPGA KC705 Evaluation Kit Getting Started Guide* (UG883)
5. *LogiCORE IP AXI Bridge for PCI Express Product Guide* (PG055)
6. *LogiCORE IP AXI Central Direct Memory Access Product Guide* (PG034)
7. *7 Series FPGAs Memory Interface Solutions User Guide* (UG586)
8. *LogiCORE IP AXI Block RAM (BRAM) Controller Product Guide* (PG078)
9. *LogiCORE IP Block Memory Generator Product Guide* (PG058)
10. *LogiCORE IP AXI Interconnect Product Guide* (PG059)
11. AMBA AXI4 specifications:
    http://infocenter.arm.com/help/topic/com.arm.doc.set.amba/index.html#specs
12. *FreeRTOS Porting Guide*: http://www.freertos.org/FreeRTOS-porting-guide.html

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
11/04/2013	1.0	Initial Xilinx release.

Notice of Disclaimer