Kintex-7 FPGA Base Targeted Reference Design

User Guide

UG882 (v1.2) August 3, 2012

XILINX®



Notice of Disclaimer
The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx 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) Xilinx 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 connection with, the Materials (including your use of the 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 or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors 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 the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx 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 use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

Automotive Applications Disclaimer
XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.

Fedora Information

Xilinx obtained the Fedora Linux software from Fedora (http://fedoraproject.org/), and you may too. Xilinx made no changes to the software obtained from Fedora. If you desire to use Fedora Linux software in your product, Xilinx encourages you to obtain Fedora Linux software directly from Fedora (http://fedoraproject.org/), even though we are providing to you a copy of the corresponding source code as provided to us by Fedora. Portions of the Fedora software may be covered by the GNU General Public license as well as many other applicable open source licenses. Please review the source code in detail for further information. To the maximum extent permitted by applicable law and if not prohibited by any such third-party licenses, (1) XILINX DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE; AND (2) IN NO EVENT SHALL XILINX BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Fedora software and technical information is subject to the U.S. Export Administration Regulations and other U.S. and foreign law, and may not be exported or re-exported to certain countries (currently Cuba, Iran, Iraq, North Korea, Sudan, and Syria) or to persons or entities prohibited from receiving U.S. exports (including those (a) on the Bureau of Industry and Security Denied Parties List or Entity List, (b) on the Office of Foreign Assets Control list of Specially Designated Nationals and Blocked Persons, and (c) involved with missile technology or nuclear, chemical or biological weapons). You may not download Fedora software or technical information if you are located in one of these countries, or otherwise affected by these restrictions. You may not provide Fedora software or technical information to individuals or entities located in one of these countries or otherwise affected by these restrictions. You are also responsible for compliance with foreign law requirements applicable to the import and use of Fedora software and technical information.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/18/12	1.0	Initial Xilinx release.
05/22/12	1.1	Updated last paragraph of Hardware Test Setup Requirements. Updated Figure 3-1. Updated table Table B-8.

Date	Version	Revision
08/03/12	1.2	Updated last paragraph of Hardware Test Setup Requirements. Added Vivado tools to Rebuilding the Base TRD. Added step 6 to Generating the MIG IP Core through CORE Generator Tool. Added Implementing the Design Using the Vivado Tools. Added Vivado tools to Configuration Requirements.

Table of Contents

Revision History 2

Chapter 1: Introduction

The Base Targeted Reference Design 7

Chapter 2: Getting Started

Requirements 11
TRD Demonstration Setup 12
Shutting Down the System 25
Rebuilding the Base TRD 26
Reprogramming the Base TRD 30
Simulation 32

Chapter 3: Functional Description

Hardware Architecture 35
Software Architecture 53

Chapter 4: Performance Estimation

PCI Express Performance 61
Packetized Virtual FIFO Performance 64
Measuring Performance 64

Chapter 5: Designing with the TRD Platform

Software-Only Modifications 67
Top-Level Design Modifications 69
Architectural Modifications 70

Appendix A: Resource Utilization

Appendix B: Register Description

DMA Registers 76
User Space Registers 79

Appendix C: Directory Structure

Appendix D: Compiling Linux Drivers

Appendix E: Additional Resources

Xilinx Resources 91
References 91

XILINX®

Chapter 1

Introduction

The Kintex™-7 Base Targeted Reference Design (TRD) delivers all the basic components of a targeted design platform for high performance in a single package. Targeted Design Platforms from Xilinx provide customers with simple, smart design platforms for the creation of FPGA-based solutions in a wide variety of industries.

This user guide details a TRD developed for high performance on a Kintex-7 FPGA. The aim is to accelerate the design cycle and enable FPGA designers to spend less time developing the infrastructure of an application and more time creating a unique value-add design. The primary components of the Kintex-7 Base TRD are the Kintex-7 FPGA integrated Endpoint block for PCI Express®, Northwest Logic Packet DMA, Memory Interface Solutions for DDR3, and AXI Interconnect IP block.

The Base Targeted Reference Design

The Kintex-7 FPGA Base Targeted Reference Design showcases the capabilities of Kintex-7 FPGAs and the various IP cores developed for this FPGA family. Figure 1-1 shows the block level overview of the architecture of the TRD. With a few custom RTL blocks interfacing with the IP blocks, the TRD can deliver up to 10 Gb/s performance end to end.

This chapter introduces the TRD and summarizes the TRD features.

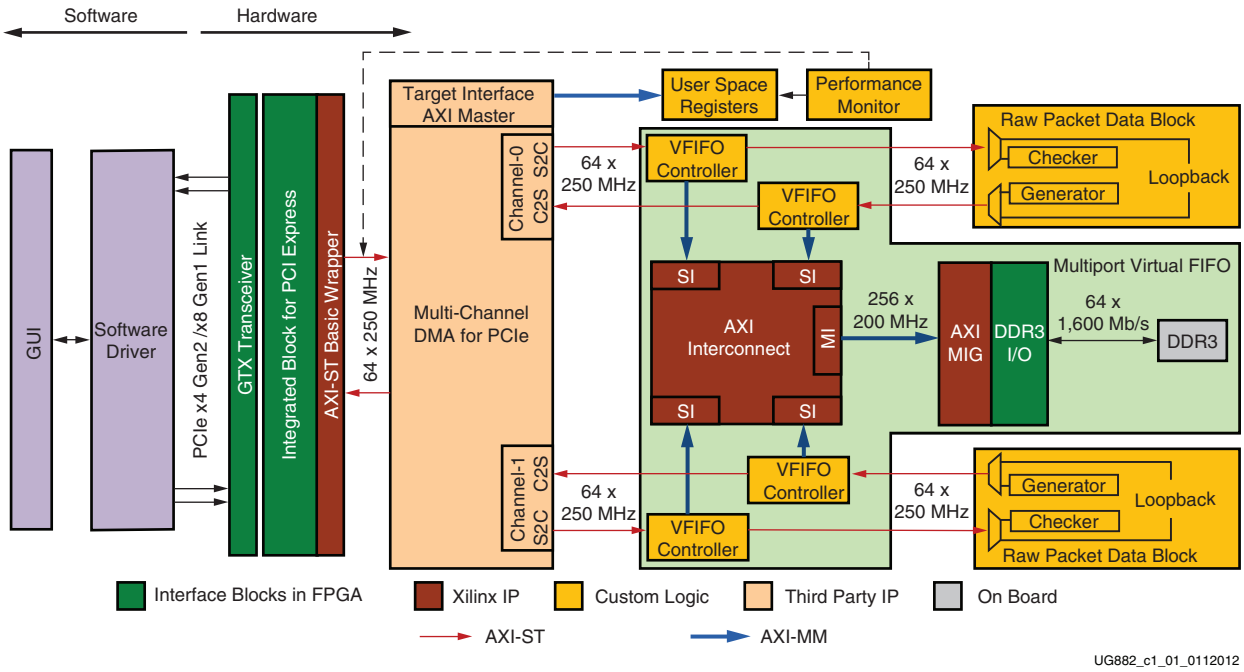


Figure 1-1: **Kintex-7 FPGA Base TRD Block Diagram**

Note: The arrows in Figure 1-1 indicate AXI interface directions from master to slave. They do not indicate data flow directions.

Base TRD Features

The Kintex-7 FPGA Base Targeted Reference Design has these components:

- 7 Series FPGAs integrated Endpoint block for PCI Express core
  - Configured with either 4 lanes at a 5 Gb/s data rate (Gen2) or 8 lanes at a 2.5 Gb/s data rate (Gen1) for PCI Express v2.0
  - Provides a user interface compliant with AXI4 stream interface protocol
  - A performance monitor tracks the integrated block's AXI4 stream interface for PCIe transactions
- Bus mastering Scatter Gather Packet DMA Engine from Northwest Logic, a multichannel DMA
  - Supports full-duplex operation with independent transmit and receive paths
  - Provides an AXI4 stream interface on the back end
  - Monitors the performance of data transfers in the receive and transmit directions
  - Provides an AXI4 memory mapped target interface to access user-defined registers

Note: The Northwest Logic Packet DMA shipped with the Base TRD is an evaluation version and expires after 12 hours of run time. To get the full version, contact Northwest Logic [Ref 15].

- Multiport Virtual FIFO
  - DDR3 SDRAM (64 bits at 1,600 Mb/s, 800 MHz) is used for buffering packets. The Memory Controller is delivered through the Memory Interface Generator (MIG) tool and interfaces to the DDR3 SRAM memory.

    - AXI Interconnect IP core with the Memory Controller supports multiple ports on the memory.
    - The Packetized Virtual FIFO controller controls the addressing of the DDR3 memory for each port, allowing the DDR3 memory to be used as Virtual Packet FIFO.
- Software driver for a 32-bit Linux platform
    - Configures the hardware design parameters
    - Generates and consumes traffic
    - Provides a Graphical User Interface (GUI) to report status and performance statistics

The 7 Series FPGAs Integrated Block for PCI Express core and the Packet DMA are responsible for data transfers from the host system to the Endpoint card (S2C) and Endpoint card to host system (C2S). Data to and from the host is stored in a Virtual FIFO built around the DDR3 memory. This Multiport Virtual FIFO abstraction layer around the DDR3 memory allows the traffic to be moved efficiently without the need to manage addressing and arbitration on the memory interface. It also provides a larger depth when compared to storage implemented using Block RAMs.

The Integrated Block for PCI Express core, Packet DMA, and Multiport Virtual FIFO can be considered as the base system. The base system can bridge the host system to any user application running on the other end. The Raw Data Packet module is a dummy application which generates and consumes packets. It can be replaced by any user-specific protocol like Aurora or XAUI.

The software driver runs on the host system. It generates raw data traffic for transmit operations in the S2C direction. It also consumes the data looped back or generated at the application end in the C2S direction.

The modular architecture of the Base TRD hardware and software components simplifies reuse and customization of the architecture to specific user requirements.

XILINX®

Chapter 2

Getting Started

This chapter is a quick start guide enabling the user to test the Kintex-7 FPGA Base Targeted Reference Design (TRD) in hardware with the software driver provided and also simulate it. It provides step-by-step instructions for testing the design in hardware.

Note: The screen captures in this document are conceptual representatives of their subjects and provide general information only.

Requirements

This section lists the prerequisites for hardware testing and simulation of the Base TRD.

Hardware Test Setup Requirements

The prerequisites required to run and test the Base TRD include:

- KC705 board with the XC7K325T-2FFG900C FPGA
- Design files provided on a USB memory stick as a zipped collection including:
  - Design source files
  - Device driver files
  - Board design files
  - Documentation
- ISE® Design Suite, Logic Edition v13.4 or later
- Micro USB cable
- PCIe adapter cable, 4-Pin to 6-Pin
- Fedora 16 Live DVD for Intel-compatible PCs or pre-installed Fedora 16 Linux OS
- PC with PCIe v2.0 slot

For a list of all known issues, refer to the Kintex-7 FPGA Base Targeted Reference Design Release Notes and Known Issues Master Answer Record (http://www.xilinx.com/support/answers/45679.htm).

Simulation Requirements

The tools required to simulate the Base TRD are:

- ISE Design Suite, Logic Edition
- ModelSim simulation software, v6.6d or later

TRD Demonstration Setup

This section provides the procedures for setting up the KC705 board and using the application GUI in preparation for demonstrating the Base TRD.

Note: When following the demonstration setup steps for the Kintex-7 FPGA Base TRD, if the behavior is not as described, refer to the known issues on the xilinx.com website [Ref 12].

Board Configuration and Bring Up

This section describes how to set up and install the KC705 board to demonstrate the Base TRD.

Configure KC705 Board Switches and Jumpers

1. Confirm the KC705 board jumpers and switches are configured as shown in Table 2-1 and Figure 2-1.

Table 2-1: **Switch and Jumper Settings**

Jumper	Function		Setting
J32	PCIe configuration width — 4 lane design		Jump 3-4
Switch	**Function or Type**		**Setting**
SW15	Board power slide-switch		off
SW11	User GPIO DIP switch		
	4		off
	3		off
	2		off
	1		off
SW13	DIP switch SW13 positions 1 and 2 control the setting of address bits of the flash. DIP switch SW13 positions 3, 4, and 5 control which configuration mode.		
	5 (M0)	M2 =0 M1=1 M0=0 — Master BPI M2 =0 M1=0 M0=1 — Master SPI M2 =1 M1=0 M0=1 — JTAG	off
	4 (M1)		on
	3 (M2)		off
	2		off
	1		off

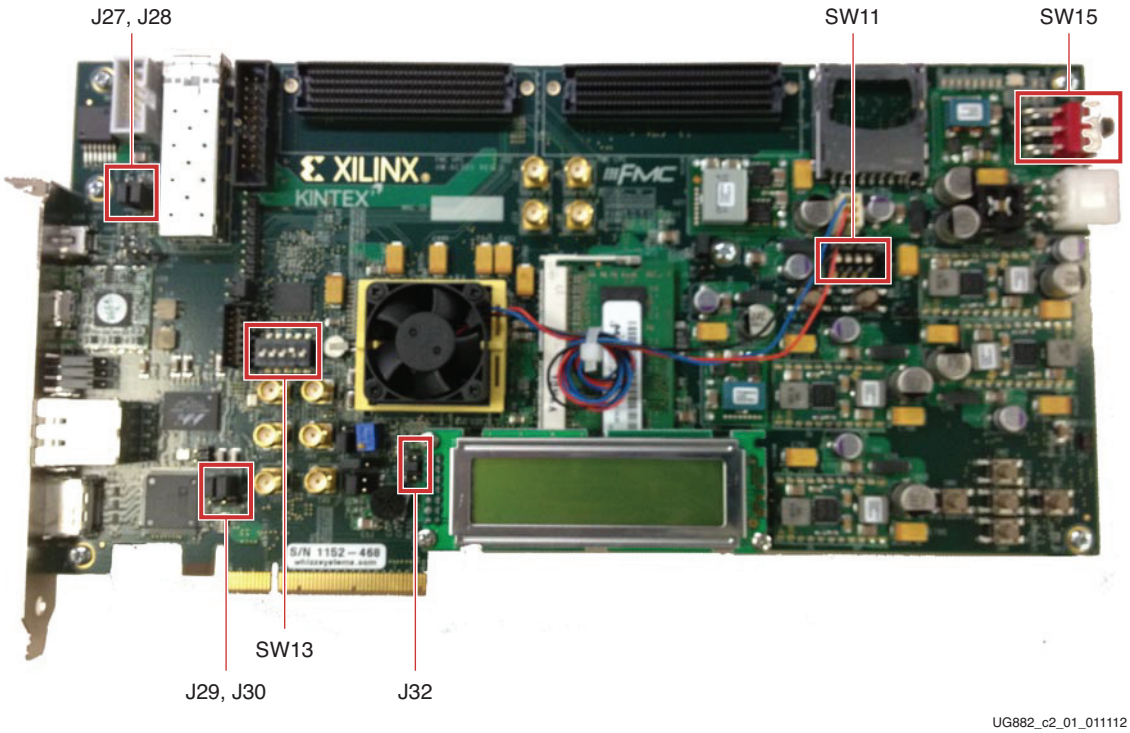
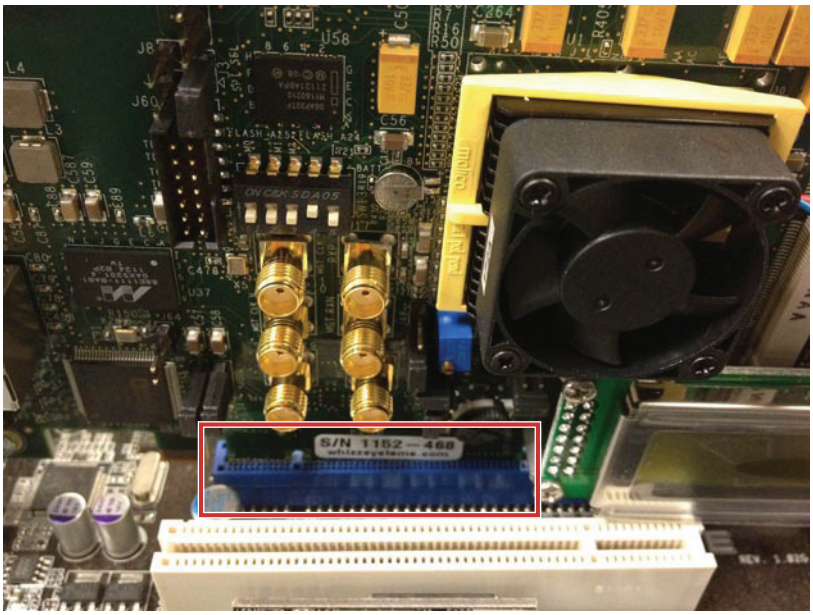


Figure 2-1: **Switch and Jumper Locations**

KC705 Board Installation

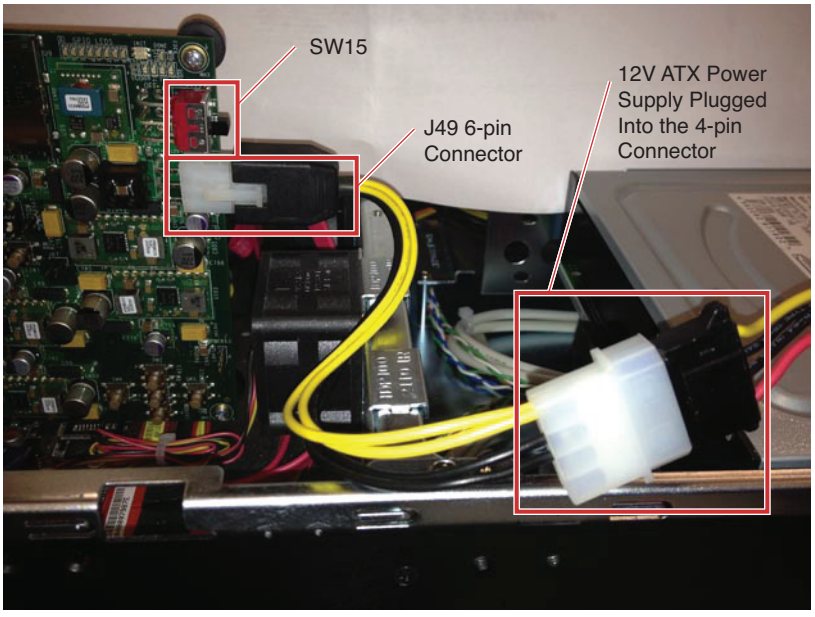
1. With the host PC switched off, insert the KC705 board in the PCIe slot through the PCI Express x8 or x16 edge connector (Figure 2-2). The Base TRD programmed on the KC705 board has a 4-lane PCIe v2.0 configuration, running at a 5 Gb/s link rate per lane. The PCI Express specification allows for a smaller lane width Endpoint to be installed into a larger lane width PCIe connector.



UG882_c2_02_011112

Figure 2-2: **KC705 Board Plugged Into a PCIe x16 Slot**

2. Connect one of the spare 4-pin connectors from the PC's 12V ATX power supply to J49 on the KC705 board using a 4-pin to 6-pin PCIe adapter cable. Toggle the KC705 board power switch SW15 to the ON position. Figure 2-3 shows the 12V power supply connection and power switch SW15.



UG882_c2_03_011112

Figure 2-3: **Power Supply Connection**

3. Confirm the connectors are latched tight and power on the PC.

   Note: If the user wishes to boot Linux from the Fedora 16 Live DVD, place the DVD in the PC's CD-ROM drive as soon as the PC system is powered on.

4. Check the status of the design on the KC705 board LEDs. The Base TRD provides status on the GPIO LEDs on the front side of the KC705 board near the upper right edge (Figure 2-4). When the PC is powered on and the Base TRD has successfully configured on the FPGA, the LED status (right to left) should indicate:
   - LED 0 - ON (the PCIe link is up)
   - LED 1 - FLASHING (the PCIe user clock is present)
   - LED 2 - ON (lane width is what is expected, else LED 2 flashes—expected lane width is 4 for a x4 design and 8 for a x8 design)
   - LED 3 - ON (Memory calibration is done)
   - LED 4 to LED 7 - Not connected

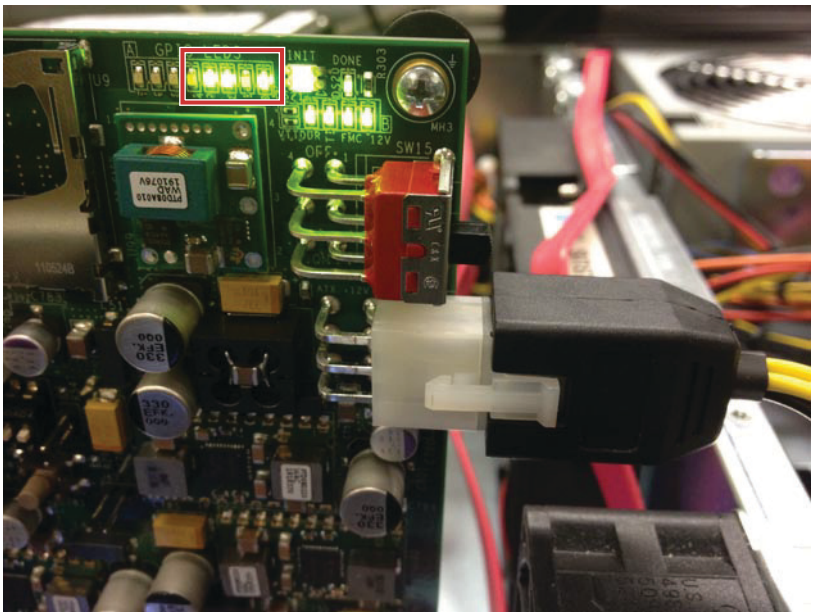


Figure 2-4: **GPIO LEDs Indicate Base TRD Status**

Linux Boot Up and Driver Installation

Booting Fedora Live

If Fedora 16 Linux OS is installed on the PC system's hard disk, boot as a root-privileged user and go to step 2, otherwise go to step 1.

1. To boot from the Fedora 16 Live DVD provided in the kit, place the DVD in the PC's CD-ROM drive.

   The Fedora 16 Live Media is for Intel-compatible PCs. For more details, see Fedora Information, page 2. The DVD contains a complete, bootable 32-bit Fedora 16 environment with the proper packages installed for the Base TRD demonstration environment. The PC boots from the CD-ROM drive and logs into a liveuser account. This account has kernel development root privileges required to install and remove device driver modules.

Note: The BIOS boot order settings might have to be changed to make sure that the CD-ROM is the first drive in the boot order. To set the boot order, enter the BIOS menu by pressing the DEL or F2 key when the system is powered on. Set the boot order and save the changes.

The DEL or F2 key is used by most PC systems to enter the BIOS setup. Some PCs might have a different way to enter the BIOS setup.

While booting, from the CD-ROM drive, the PC displays the images shown in Figure 2-5.

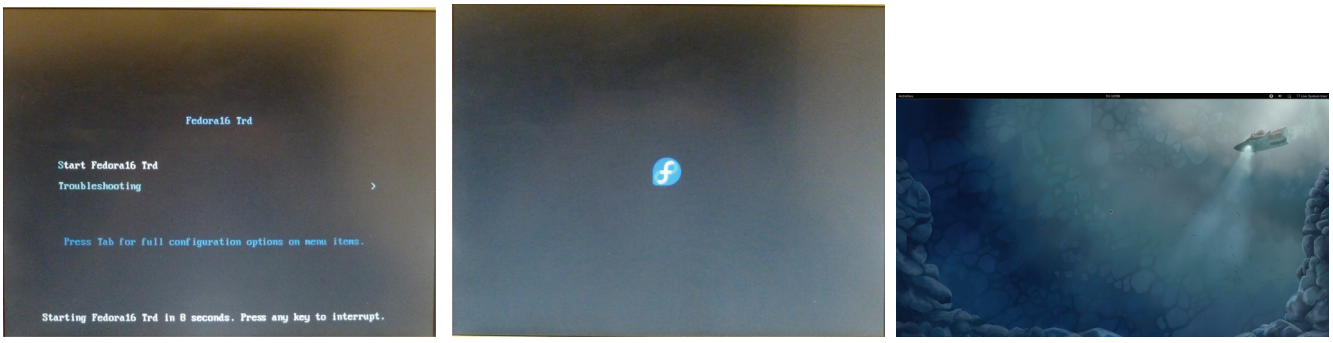


Figure 2-5: **Fedora 16 Live DVD Boot Images**

After the Fedora Core boots, log in as liveuser.

2. After Fedora Core boots, open a terminal window (click **`Activities > Application`**, scroll down, and click the **`Terminal`** icon).

   To find out if the PCIe integrated Endpoint block is detected, at the terminal command line, type:

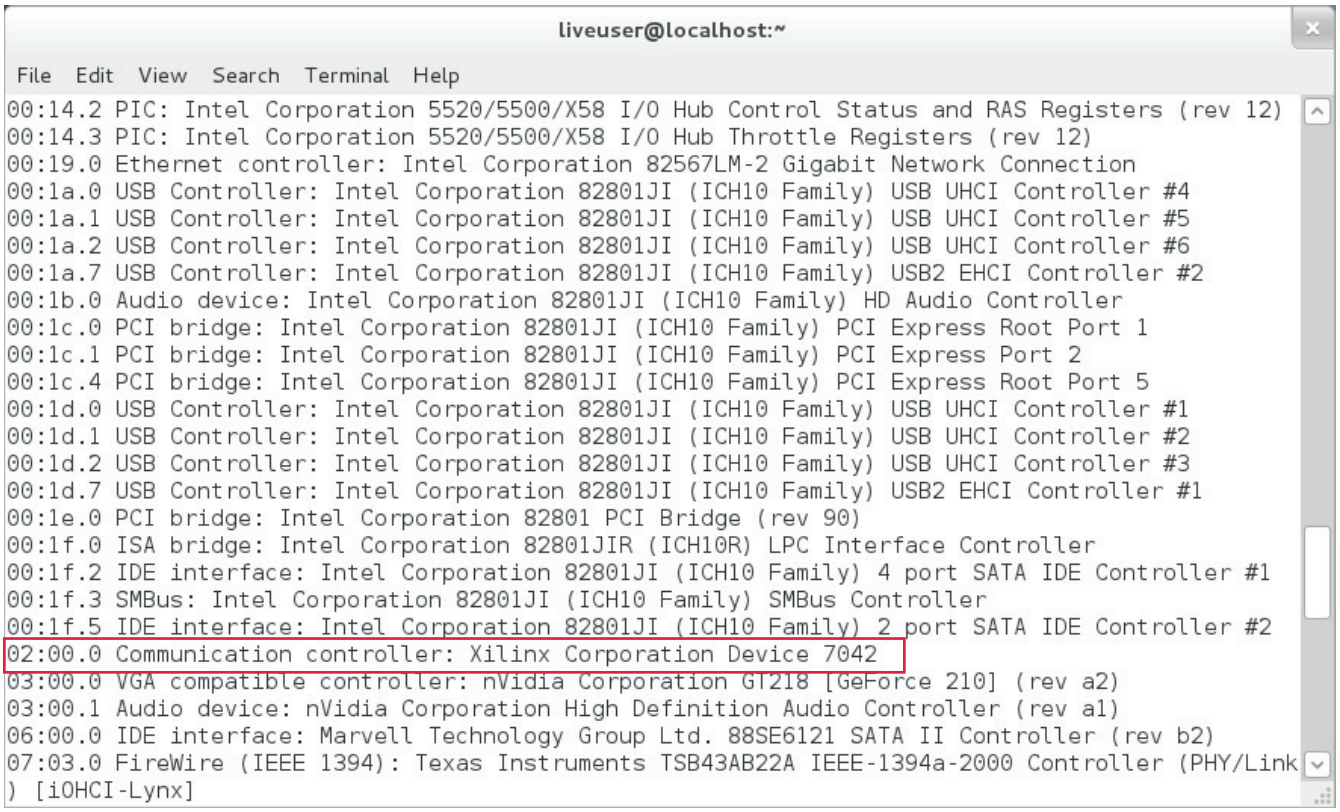
   ```
   $ lspci
   ```

   The **`lspci`** command displays the PCI and PCI Express buses of the PC. On the bus of the KC705 card slot is the message

   ```
   Communication controller: Xilinx Corporation Device 7042
   ```

   This message confirms that the design programmed into the KC705 board has been found by the BIOS and the Fedora 16 OS. The bus number varies depending on which PC motherboard and slot is used.

   Figure 2-6 shows an example of the output from the **`lspci`** command. The red highlight show that Xilinx device 7042 has been found by the BIOS on bus number 2 (`02:00.0` = `bus:dev.function`).



UG882_c2_06_010612

Figure 2-6: **PCI and PCI Express Bus Devices**

Copy Base TRD Files

The Base TRD design files are provided on a USB flash drive delivered as a part of the kit. The contents of the USB drive are also available on the Kintex-7 FPGA Evaluation Kit web page. Check for updates to Base TRD at the same location [Ref 13].

1. Insert the USB flash drive into a USB connector on the PC System and allow Fedora 16 to mount the USB device. An icon will pop up on the desktop when the USB flash drive is mounted.

   Note: The USB drive must always be unmounted before powering down the system or removing the flash drive. File corruption or kernel crash might occur otherwise. To unmount the USB flash drive, right-click the USB flash drive icon and select **`Safely Remove Drive`**.

2. Double click the USB flash drive icon and copy the `k7_pcie_dma_ddr3_base` folder into any directory.

Driver Installation

To set up and run the TRD demonstration, the software driver should be installed on the PC system.

Installation of the software driver involves:

- Building the kernel objects and the GUI.
- Inserting the driver modules into the kernel.

After the driver modules are loaded, the application GUI can be invoked. The user can set parameters through the GUI and run the TRD.

When the user is done running the TRD, the application GUI can be closed and the drivers can be removed.

A script is provided to execute these actions. To run this script:

1. Double-click `k7_trd_lin_quickstart` in the `k7_pcie_dma_ddr3` folder (Figure 2-7).

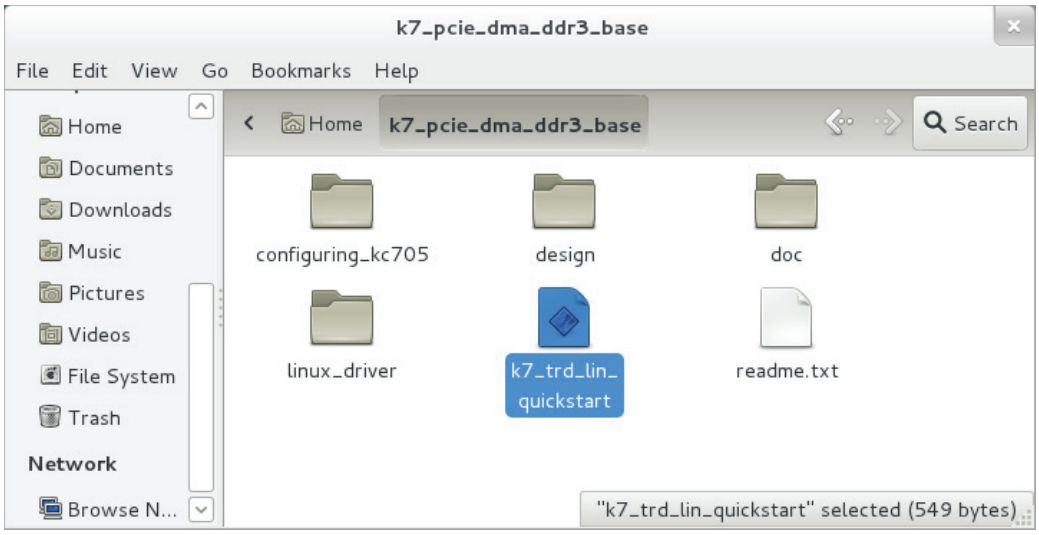


Figure 2-7: **k7_pcie_dma_ddr3 Folder**

2. When the window prompt shown in Figure 2-8 appears. Click **`Run in Terminal`**.

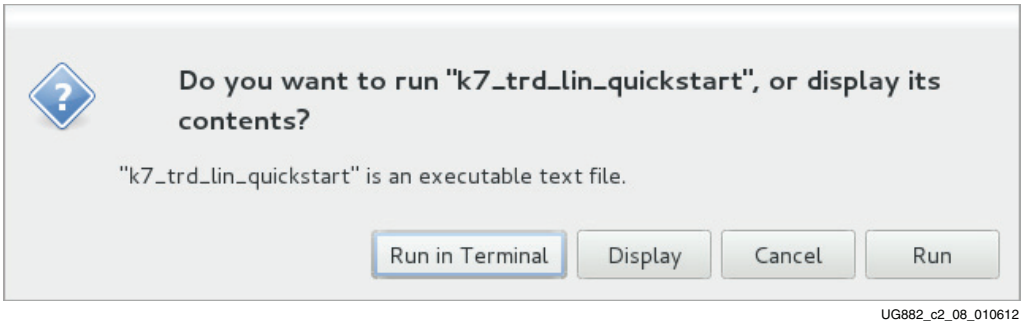


Figure 2-8: **Run in Terminal**

The application GUI is invoked. Proceed to Using the Application GUI to set design parameters and run the Base TRD.

In case issues are encountered or if the user wants to understand driver details, the user can run the individual steps detailed in Appendix D, Compiling Linux Drivers.

Using the Application GUI

After the drivers are loaded and the GUI is invoked, the application can be configured for sending and receiving data. The GUI shows the Base TRD status and performance statistics collected over time.

This section provides a screen-by-screen description of the GUI.

Test Setup and Payload Statistics

This screen shows up as soon as the GUI is invoked. It defines the various test options provided for the raw data paths.

For each raw data path: The user can input a fixed packet size in bytes. While executing the test, the software driver builds packets of fixed length. The packet size can range from 64 bytes to 32,768 bytes. Select **`Enable Loopback`** to loopback the transmit data and send it in the receive direction. This loopback is done at the application end (Raw Packet Data block). Click **`Start Test`** to begin packet generation. As packets are generated, the GUI plots the number of bytes transmitted and received by the Packet DMA for each raw data path. Click **`Stop Test`** to stop packet generation. The screen in Figure 2-9 shows the data throughput obtained from the C2S and S2C DMA engines for the raw data Path0 with **`Enable TX→ RX Loopback`** selected.

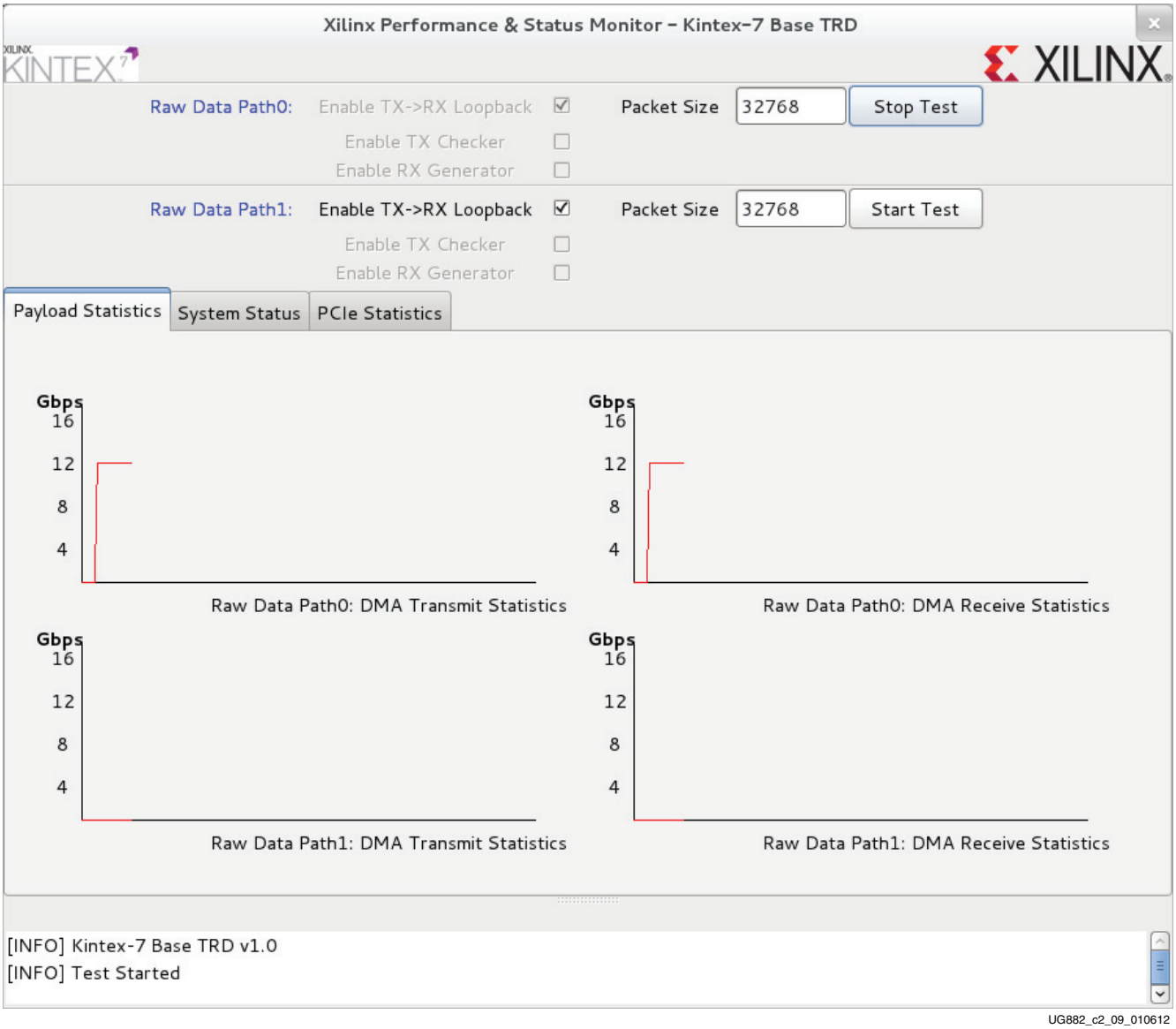


Figure 2-9: **Test Setup and Payload Statistics Screen – Raw Data TX>RX Loopback**

Unselect **`Enable Loopback`** to select **`Enable TX Checker`** or **`Enable RX Generator`** or both. Select **`Enable TX Checker`** and click **`Start Test`** to enable the data checker implemented in hardware. The packets generated by the driver are transferred via the Packet DMA and are verified at the application end (Raw Packet Data block) by the checker. The GUI plots the number of bytes transmitted by the Packet DMA. Click **`Stop Test`** to stop packet generation in the transmit path The screen in Figure 2-10 shows the data throughput obtained from the S2C DMA engine for the raw data Path0 with **`Enable TX Checker`** selected.

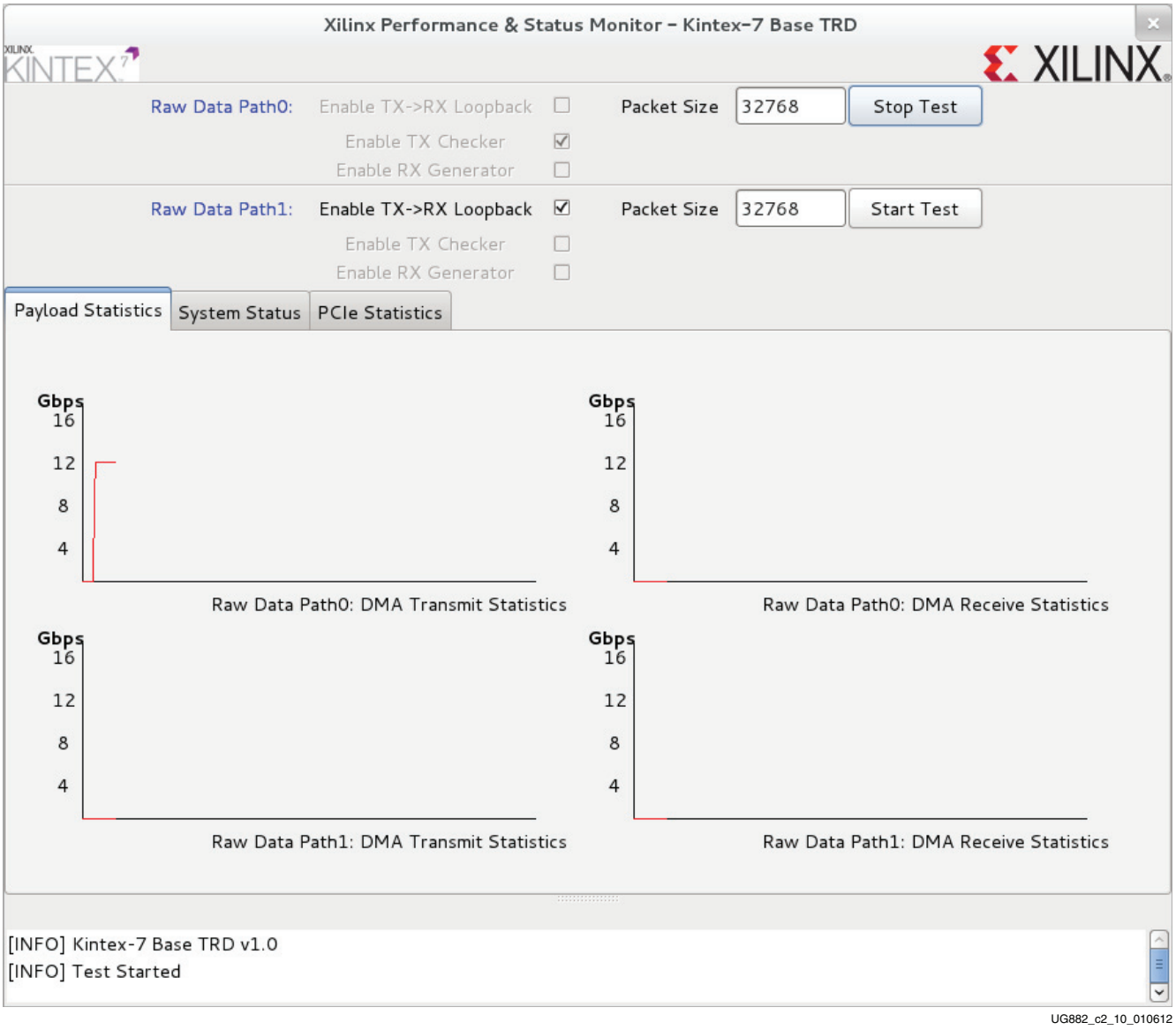


Figure 2-10: **Test Setup and Payload Statistics Screen – Raw Data TX Only**

Uncheck **`Enable TX Checker`**, select **`Enable RX Generator`**, and click **`Start Test`** to enable the data generator implemented in hardware. The packets generated are transferred via the Packet DMA to the host system and are spot checked by the driver. The GUI plots the number of bytes received by the Packet DMA. Click **`Stop Test`** to stop packet generation in the receive path. The screen in Figure 2-11 shows the data throughput obtained from the S2C DMA engine for the raw data Path0 with **`Enable RX Generator`** selected.

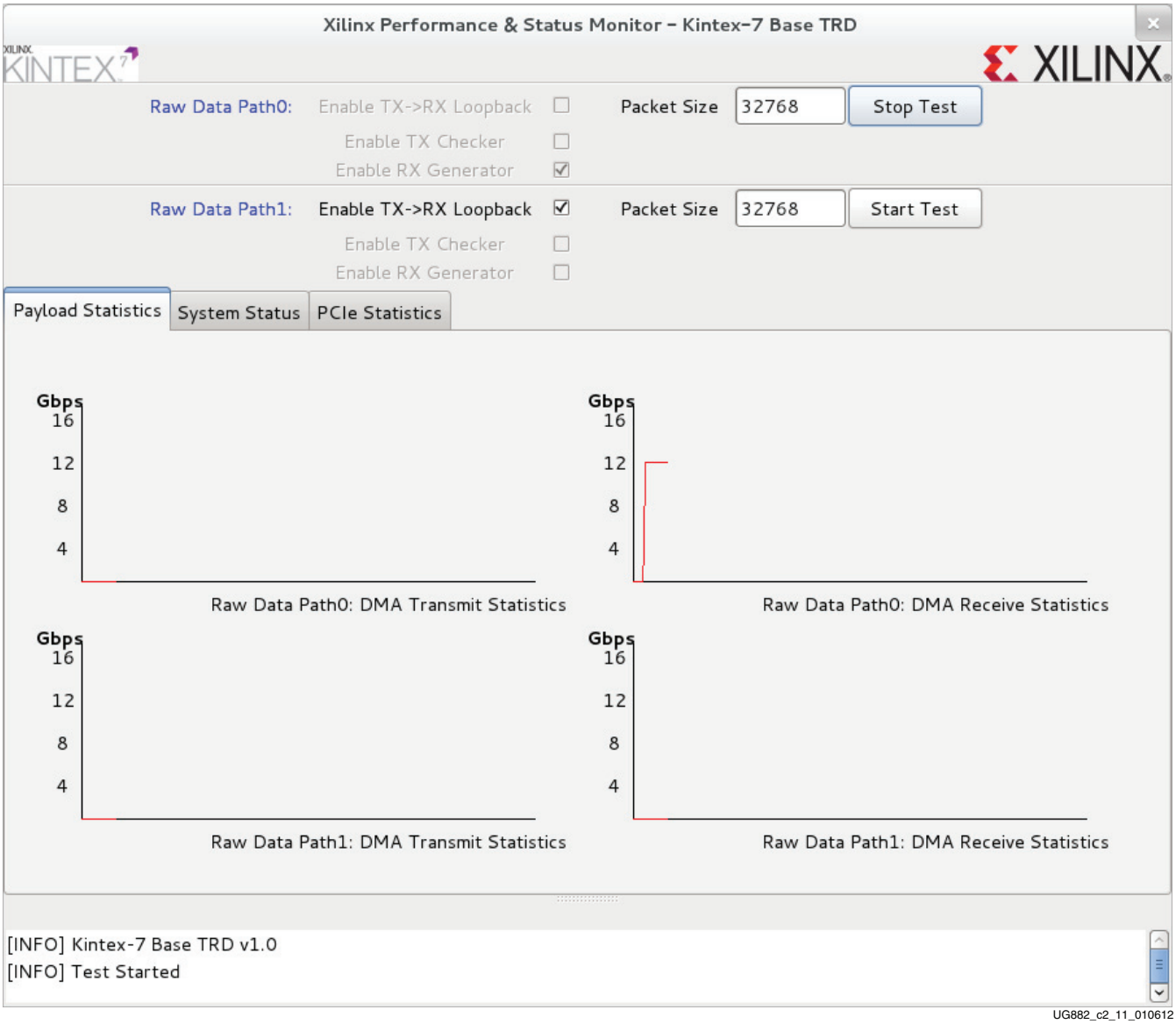


Figure 2-11: **Test Setup and Payload Statistics Screen – Raw Data RX Only**

Select **`Enable TX Checker`** and **`Enable RX Generator`** and click **`Start Test`** to enable both the data checker and the data generator. Packets are generated and checked in both directions. The GUI plots the number of bytes transmitted and received by the Packet DMA. Click **`Stop Test`** to stop packet generation. The screen in Figure 2-12 shows the data throughput obtained from the S2C and C2S DMA engines for the raw data Path0 with **`Enable TX Checker`** and **`Enable RX Generator`** selected.

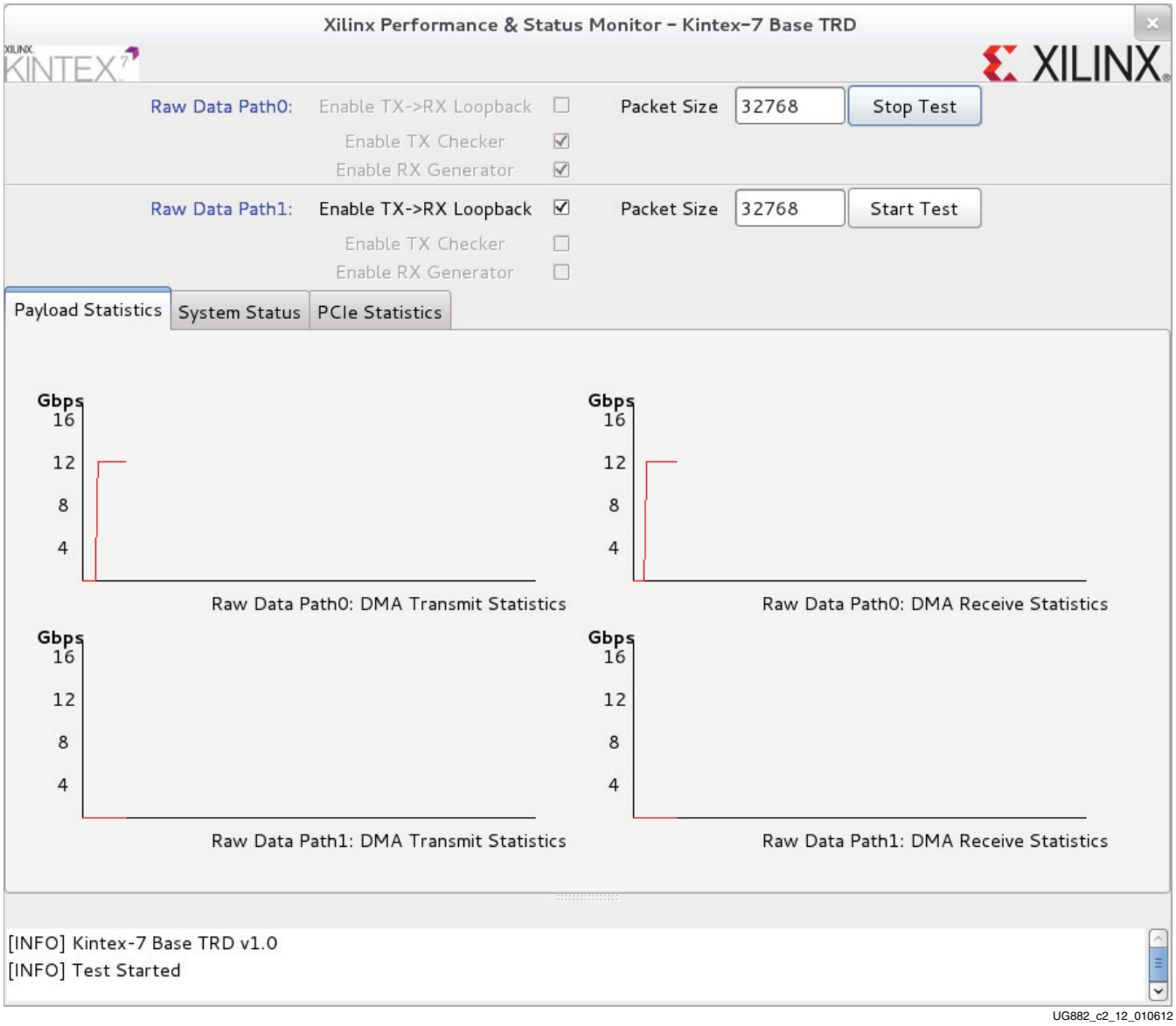


Figure 2-12: **Test Setup and Payload Statistics Screen – Raw Data TX and RX**

Note: If **`Enable Loopback`** is selected, then **`Enable TX Checker`** and **`Enable RX Generator`** options are not available to the user. If **`Enable TX Checker`** is selected, then the **`Enable Loopback`** option is not available to the user. If **`Enable RX Generator`** is selected, then the **`Enable Loopback`** option is not available to the user. The **`Enable TX Checker`** and **`Enable RX Generator`** options can be selected simultaneously. For both raw data paths, all configuration options should be selected before clicking **`Start Test.`** Configuration options that a user changes while a test is running are not taken into account.

System Status

Click the **`System Status`** tab to view the system status screen (see Figure 2-13). This screen shows the throughput numbers reported by the DMA engines for raw data Path0 and the performance monitor on the transaction layer of the Kintex-7 FPGA. For more details on the System Status window, refer to Figure 3-10.

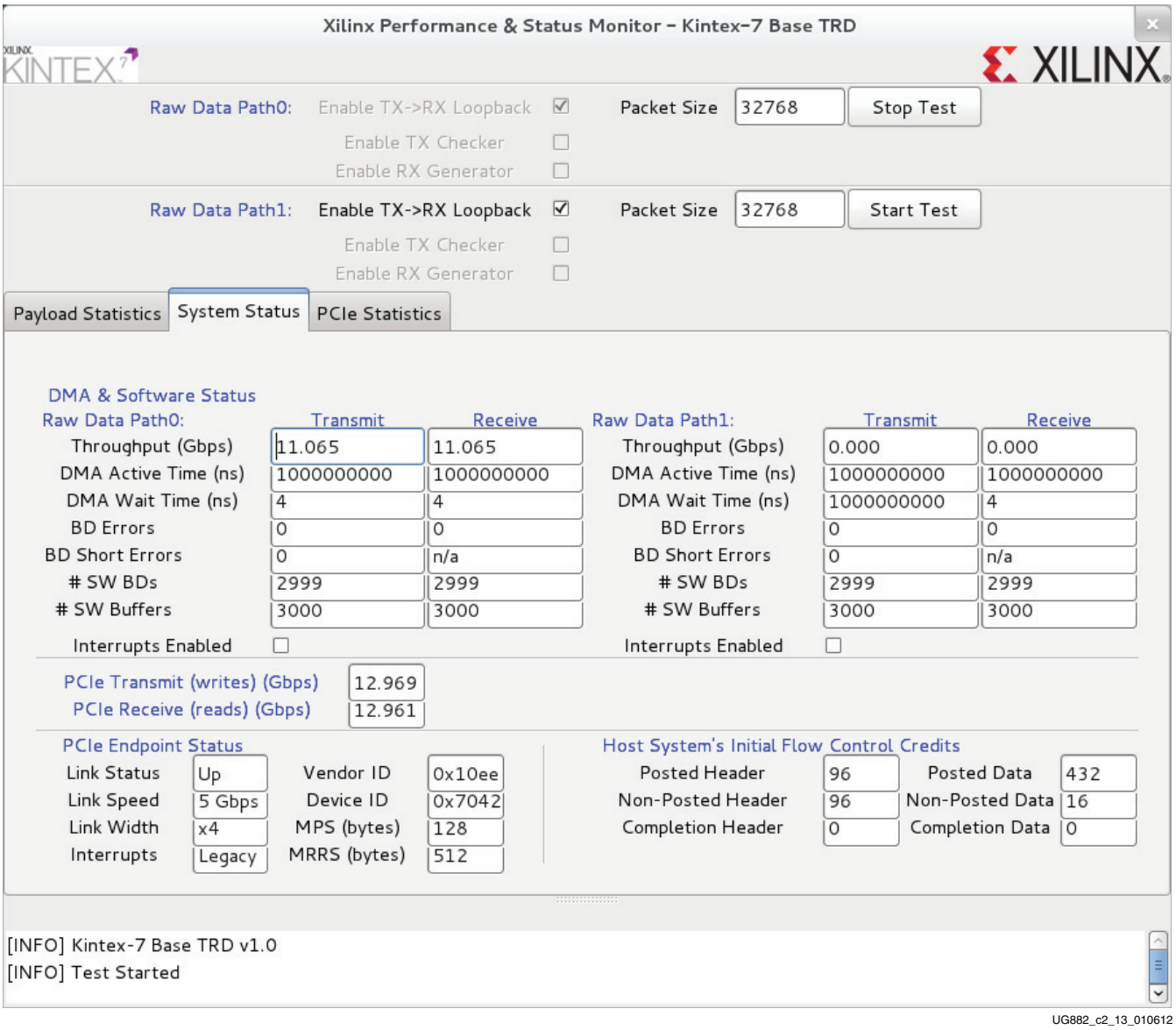


Figure 2-13: **System Status**

Transaction Statistics

Click the **`PCIe Statistics`** tab (Figure 2-14) to view the PCIe transaction statistics screen. This screen plots the data bus utilization statistics on the AXI4-Stream interface. After the base TRD has run successfully, close the application GUI. Wait for the drivers to be removed, and then proceed to Shutting Down the System, page 25.

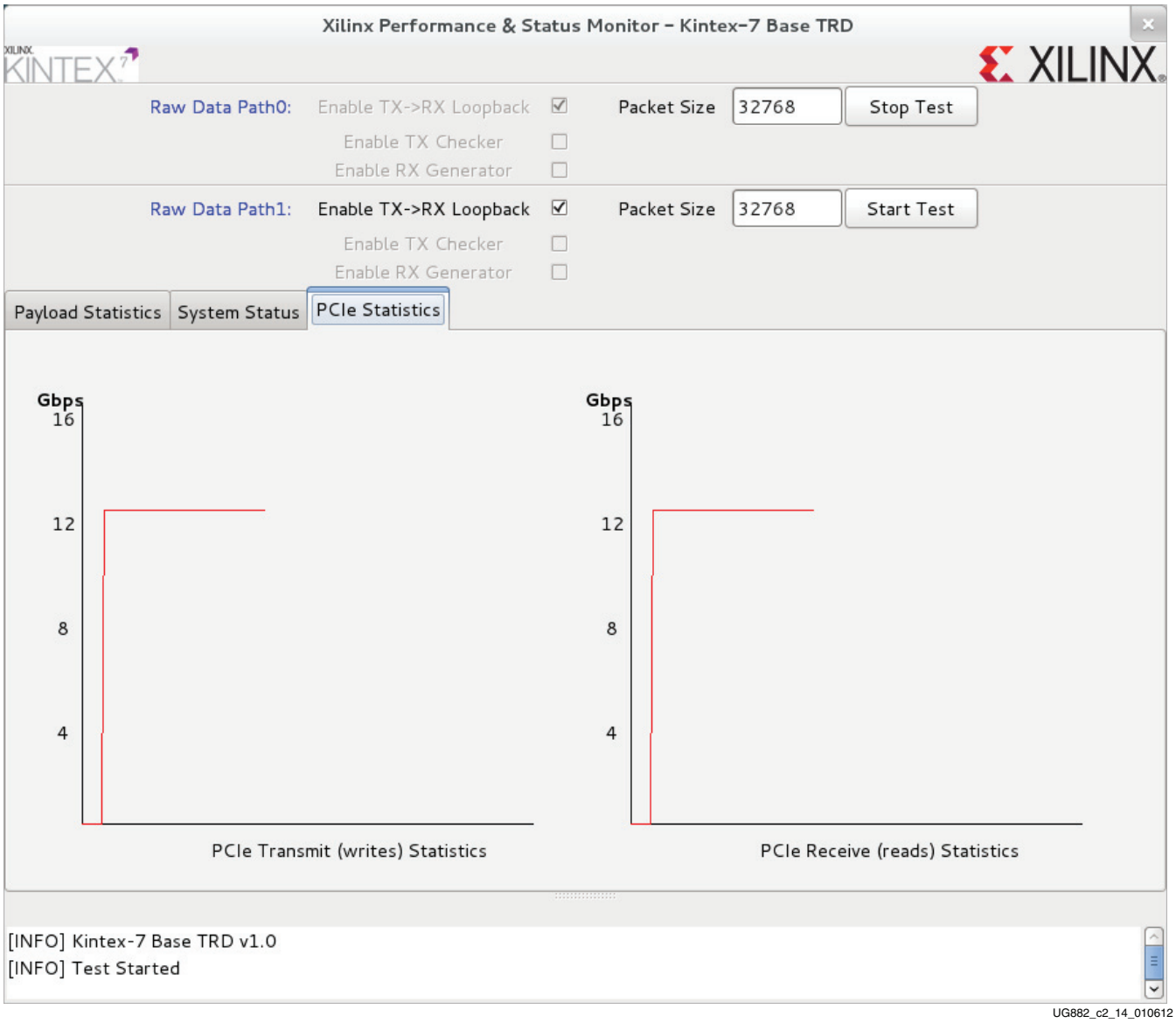


Figure 2-14: **Transaction Statistics**

Shutting Down the System

Before the PC system running Linux OS is shut down, follow these steps:

1. Unmount the USB flash drive. To unmount the drive, right-click the USB flash drive icon and select **`Safely Remove Drive`**.

   Caution! If the USB flash drive is not unmounted, files might become corrupted or the kernel might crash

2. Hold down the ALT key and select **`Live System User > Power off`** option to shut down the system. If the ALT key is not held down, only the Suspend option is available. The system slowly shuts down all processes.

Note: Any files copied or icons created will not be present after the next Fedora 16 Live DVD boot.

Rebuilding the Base TRD

The `configuring_kc705` folder provides the BIT and MCS files for the Base TRD with the PCIe link configured as x4 at a 5 Gb/s link rate (Gen2) and x8 at a 2.5 Gb/s link rate (Gen1). They can be used to reprogram the KC705 board. Programming the KC705 board with the design, where the PCIe link is configured as x8 at a 2.5 Gb/s link rate requires driver changes for the Base TRD to run successfully. Refer to Hardware and Software Modifications, page 69 for details.

The designs can also be re-implemented using the ISE or Vivado™ tools. Before running any command line scripts, refer to the "Platform-Specific Installation Instructions" section in UG798, *Xilinx Design Tools: Installation and Licensing Guide* [Ref 2] to learn how to set the appropriate environment variables for the operating system. All scripts mentioned in this user guide assume the XILINX environment variables have been set.

Note: The development machine does not have to be the hardware test machine with the PCIe slots used to run the Base TRD.

Copy the `k7_pcie_dma_ddr3_base` files to the PC with the ISE or Vivado tools installed.

The LogiCORE™ IP blocks required for the Base TRD are shipped as a part of the package. These cores and netlists are located in the `k7_pcie_dma_ddr3_base/design/ip_cores` directory:

- pcie
- fifo
- axi_ic

The MIG IP core cannot be delivered as a part of the Base TRD source, because customers have to accept a license agreement for the Micron simulation models. These models are used when simulating the Base TRD. Users must generate the MIG IP core using the ISE CORE Generator tool before trying to simulate or implement the Base TRD with the ISE tools.

For users of the Vivado tools, the IP Catalog project files are in the `k7_pcie_dma_ddr3_base/design/ip_catalog` directory, and the IP cores will be generated automatically when the synthesis step is initiated.

Generating the MIG IP Core through CORE Generator Tool

1. Open a terminal window (Linux) or an ISE Design Suite Command Prompt (Windows).
2. Navigate to `k7_pcie_dma_ddr3_base/design/ip_cores/mig` (this directory has `mig.xco`, `mig.prj`, `golden_kc705_base_trd.ucf` and `coregen.cgp` files).
3. Invoke the CORE Generator tool:

   ```
   $ coregen
   ```

4. In the CORE Generator tool, click **`File > Open project`**, and select coregen project file **`coregen.cgp`**.
5. Double click Instance Name **`mig_7x`** (Figure 2-15). This will pop up the Memory Interface Generator GUI with the configuration defined by `mig.xco` and `mig.prj` files.
6. Click **`Next`** until the Pin Selection for Controller 0 page is showing. Here, the pin assignments for the KC705 board will be automatically loaded. Hit the **`Validate`**

button and then **OK** for the pop-up window that appears afterward. Then click **Next** to go on.

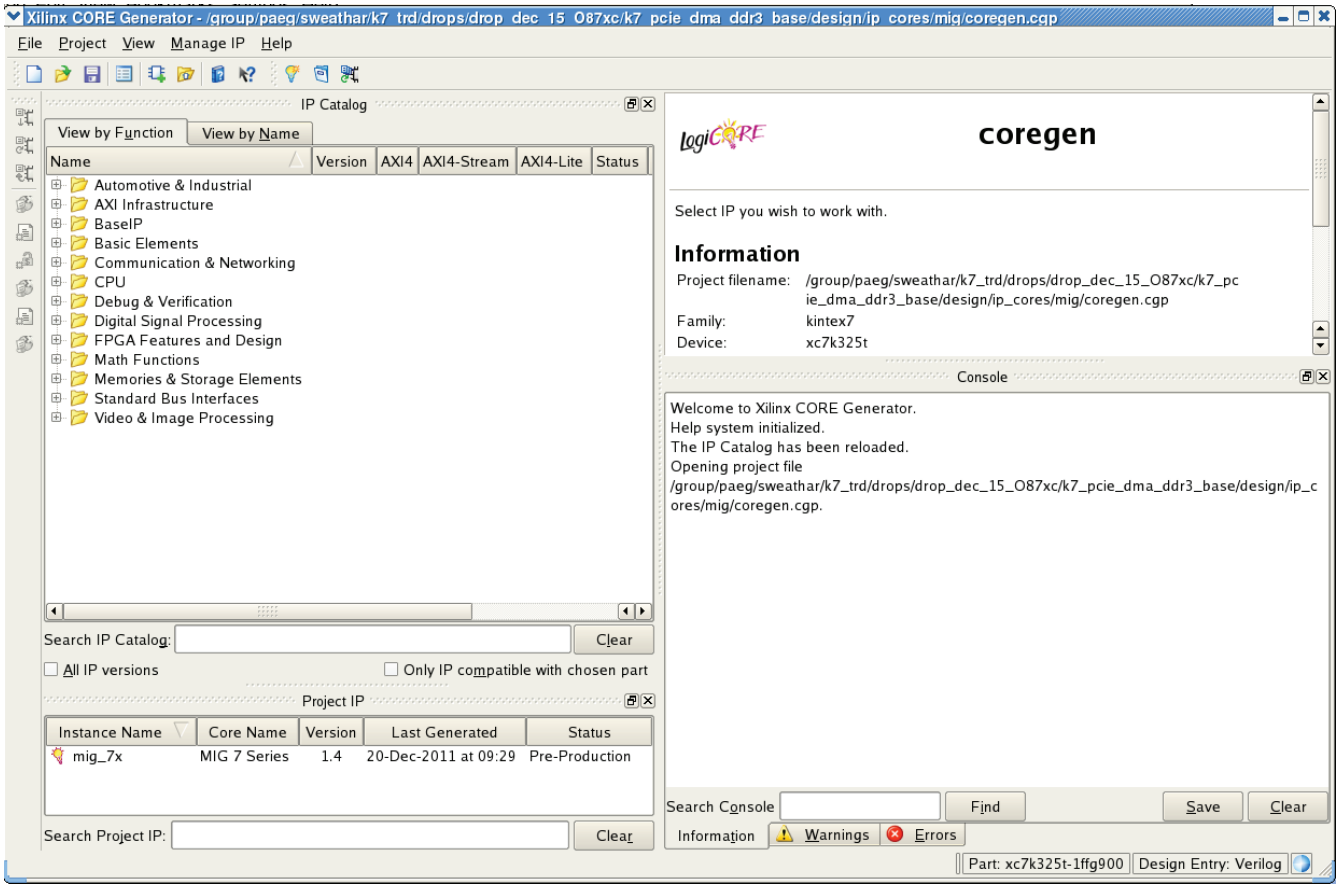


Figure 2-15: **CORE Generator Tool GUI to Generate the MIG IP Core**

Note: The version you see in Figure 2-15 might not be the version on your screen.

7. Click **Next** until the Micron Tech Inc Simulation Model License Agreement page. Select **Accept** and Click **Next**. This selection will generate the memory models required for simulation.
8. In the following page click **Next**. Then click **Generate** to create the MIG IP core.
9. Close the Readme Notes Window and then the CORE Generator tool GUI.

Additionally, a golden set of XCO files are also provided under the `k7_pcie_dma_ddr3_base/design/ip_cores/reference` directory so that the cores can be regenerated, if desired.

To regenerate the core, copy `mig.xco` and `mig.prj` from the `design/ip_cores/reference` directory.

Implementing the Design Using Command Line Options

1. Open a terminal window (Linux) or an ISE Design Suite Command Prompt (Windows).
2. Navigate to `k7_pcie_dma_ddr3_base/design/implement` directory.

3. At the command line of a terminal window (Linux) or ISE Design Suite Command Prompt (Windows), use one of these commands to invoke the ISE software tools and produce a BIT file and an MCS file in the results folder for downloading to the KC705 board:

   $ **source implement.sh x4 gen2** (for Linux)

   $ **source implement.sh x8 gen1** (for Linux)

   $ **implement.bat -lanemode x4gen2** (for Windows)

   $ **implement.bat -lanemode x8gen1** (for Windows)

   To view other options available through the implement script, run these commands:

   $ **source implement.sh -help** (for Linux)

   $ **implement.bat -help** (for Windows)

Implementing the Design Using the PlanAhead Design Tool

Base TRD with PCIe Configured as x4 at a 5 Gb/s Link Rate

1. Open a terminal window (Linux) or an ISE Design Suite Command Prompt (Windows).
2. For PlanAhead™ design tool flow for Windows and Linux, navigate to `design/implement/planahead_flow_x4gen2`.
3. Run the following command to invoke the PlanAhead design tool GUI. The design with x4 gen2 PCIe configuration is loaded:

   $ **launch_pa_x4gen2.bat**

4. Click **Synthesize** in the Project Manager window. A window with message Synthesis Completed Successfully appears after XST generates a design netlist. Close the message window.
5. Click **Implement** in the Project Manager window. A window with the message Implementation Completed Successfully appears after translate, map and par processes are done. Close the message window.
6. Click **Program & Debug**. Click **Generate Bitstream**. An options window appear.s In the column next to the `-f` field, browse to directory `design/implement` and select **bitgen_options.ut**. Click **OK** to generate bitstream. A window with the message `Generate Bitstream Completed Successfully` appears at the end of this process and a design bit file is available in `design/implement/planahead_flow_x4gen2/planAhead_run_1/k7_pcie_dma_ddr3_base_x4gen2.runs/impl_1`.
7. Close the PlanAhead design tool GUI.
8. Run the following command to generate an MCS file:

   $ **genprom.bat** (for Windows)

   $ **./genprom.sh** (for Linux)

   A promgen file is available in `design/implement/planahead_flow_x4gen2.`

Base TRD with PCIe Configured as x8 at a 2.5 Gb/s Link Rate

1. Open a terminal window (Linux) or an ISE Design Suite Command Prompt (Windows).
2. Navigate to `design/implement/planahead_flow_x8gen1`.
3. Run the following command to invoke the PlanAhead design tool GUI. The design with x8gen1 PCIe configuration is loaded:

   `$ launch_pa_x8gen1.bat`

4. Click **Synthesize** in the Project Manager window. A window with message Synthesis Completed Successfully appears after XST generates a design netlist. Close the message window.
5. Click **Implement** in the Project Manager window. A window with message Implementation Completed Successfully appears after translate, map and par processes are done. Close the message window.
6. Click **Program & Debug**. Click **Generate Bitstream**. An options window appears. In the column next to the `-f` field, browse to directory `design/implement` and select **bitgen_options.ut**. Click **OK** to generate bitstream. A window with message `Generate Bitstream Completed Successfully` appears at the end of this process and a design bit file will be available in `design/implement/planahead_flow_x8gen1/planAhead_run_1/k7_pcie_dma_ddr3_base_x8gen1.runs/impl_1`.
7. Close the PlanAhead design tool GUI.
8. Run the following command to generate an MCS file:

   `$ genprom.bat` (for Windows)

   `$ ./genprom.sh` (for Linux)

   A promgen file will be available in `design/implement/planahead_flow_x8gen1`.

   Note: If the configuration width selected is x8 gen1, driver changes are required for the Base TRD to run successfully. Refer to Hardware and Software Modifications, page 69 for details.

Implementing the Design Using the Vivado Tools

Base TRD with PCIe Configured as x4 at a 5 Gb/s Link Rate

1. Open a terminal window (Linux) or an ISE Design Suite Command Prompt (Windows).
2. For the Vivado™ design tools flow for Windows and Linux, navigate to `design/implement/vivado_flow_x4gen2`.
3. Run the following command to invoke the Vivado design tool GUI. The design with x4 gen2 PCIe configuration is loaded:

   `$ launch_x4gen2.bat`

4. Click **Run Synthesis** in the Project Manager window. A window with the message Synthesis Completed Successfully appears after the Vivado tools generate a design netlist. Close the message window.
5. Click **Run Implementation** in the Project Manager window. A window with the message Implementation Completed Successfully appears after the mapping and placement and routing processes are done. Close the message window.

6. Click **Generate Bitstream**. A window with the message Generate Bitstream Completed Successfully appears at the end of this process and a design bit file is available in `design/implement/vivado_flow_x4gen2/vivado_proj_1/k7_pcie_dma_ddr3_base_x4gen2.runs/impl_1`.
7. Close the Vivado design tools GUI.
8. Run the following command to generate an MCS file:

   `$` **`genprom.bat`** (for Windows)

   `$` **`./genprom.sh`** (for Linux)

A promgen file will be available in `design/implement/vivado_flow_x4gen2`.

Base TRD with PCIe Configured as x8 at a 2.5 Gb/s Link Rate

1. Open a terminal window (Linux) or an ISE Design Suite Command Prompt (Windows).
2. Navigate to `design/implement/vivado_flow_x8gen1`.
3. Run the following command to invoke the Vivado design tools GUI. The design with x8gen1 PCIe configuration is loaded:

   `$` **`launch_x8gen1.bat`**

4. Click **Run Synthesis** in the Project Manager window. A window with the message Synthesis Completed Successfully appears after the Vivado tools generate a design netlist. Close the message window.
5. Click **Run Implementation** in the Project Manager window. A window with the message Implementation Completed Successfully appears after mapping and placement and routing processes are done. Close the message window.
6. Click **Generate Bitstream**. A window with the message Generate Bitstream Completed Successfully appears at the end of this process and a design bit file is available in `design/implement/vivado_flow_x8gen1/vivado_proj_1/k7_pcie_dma_ddr3_base_x8gen1.runs/impl_1`.
7. Close the Vivado design tools GUI.
8. Run the following command to generate an MCS file:

   `$` **`genprom.bat`** (for Windows)

   `$` **`./genprom.sh`** (for Linux)

A promgen file will be available in design/implement/vivado_flow_x8gen1.

Note: If the configuration width selected is x8 gen1, driver changes are required for the Base TRD to run successfully. Refer to Hardware and Software Modifications, page 69 for details.

Reprogramming the Base TRD

The KC705 board is shipped preprogrammed with the Base TRD, where the PCIe link is configured as x4 at a 5 Gb/s link rate. This procedure shows how to return the KC705 board to its original condition after another user has programmed it for a different operation or as a training aid for users to program their boards. The PCIe operation requires the use of the BPI Linear Flash mode of the KC705 board. This is the only configuration option that meets the strict programming time of PCI Express. Refer to the *7 Series FPGAs Integrated Block for PCI Express User Guide* for more information on PCIe [Ref 4]

Configuration Requirements

1. Check the KC705 board switch and jumper settings as shown in Table 2-1 and Figure 2-1. Connect the micro USB cable and use the wall power adapter to provide 12V power to the 6-pin connector as shown in Figure 2-15.



Figure 2-16: **Cable Installation for KC705 Board Programming**

2. Copy the `k7_pcie_dma_ddr3_base` files to the PC with Xilinx programming tools or the ISE or Vivado Design Suites installed.
3. Open a terminal window (Linux) or an ISE Design Suite Command Prompt (Windows).
4. Navigate to the `k7_pcie_dma_ddr3_base/configuring_kc705` directory.
5. Execute the FPGA programming script at the command prompt. This operation takes approximately 500 to 800 seconds to complete.

   $ **`kc705program.bat`** (for Windows)

   $ **`impact -batch kc705program.cmd`** (for Linux)

   When complete, the "Programmed Successfully" message is displayed as shown in Figure 2-17. Remove the power connector and carefully remove the micro USB cable. The Kintex-7 Base TRD is now programmed into the BPI Linear Flash and will automatically configure at power up.

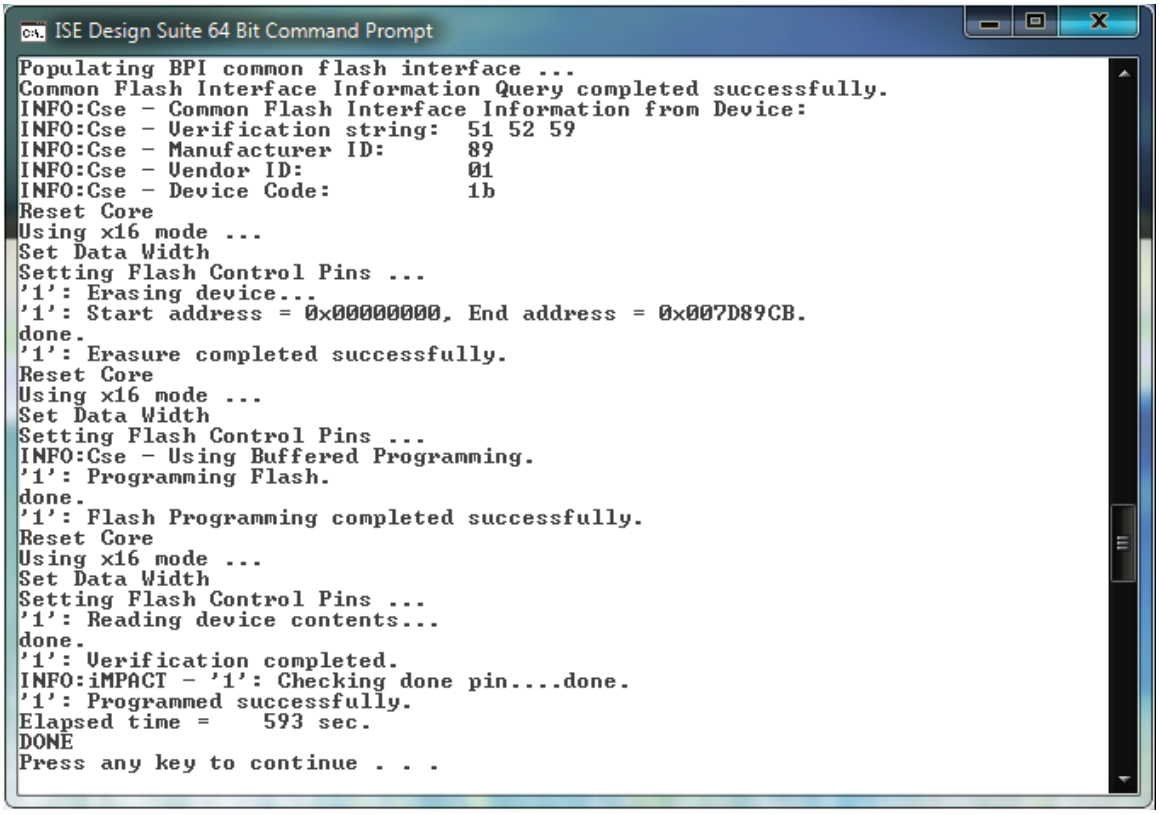


Figure 2-17: **Programming the KC05 Board Flash (Windows 7 OS)**

If the design has been rebuilt according to the instructions in Rebuilding the Base TRD, page 26, navigate to the `k7_pcie_dma_ddr3_base/design/implement` directory. The BIT and MCS files generated during implementation and the scripts to program the KC705 board are located in the `results` directory.

Navigate to the `results` directory and run the FPGA programming script at the command prompt to configure the KC705 board with the design built in the implement folder.

For the designs rebuilt using the PlanAhead design tool, the MCS files and the FPGA programming scripts are available at `k7_pcie_dma_ddr3_base/design/implement/planahead_flow_x4gen2` and `k7_pcie_dma_ddr3_base/design/implement/planahead_flow_x8gen1`.

For the designs rebuilt using the Vivado design tools, the MCS files and the FPGA programming scripts are available at `k7_pcie_dma_ddr3_base/design/implement/vivado_flow_x4gen2` and `k7_pcie_dma_ddr3_base/design/implement/vivado_flow_x8gen1`.

Simulation

The out-of-box simulation environment consists of the design under test (DUT) connected to the Kintex-7 FPGA Root Port Model for PCI Express. This simulation environment demonstrates the basic functionality of the Base TRD through various test cases. The out-of-box simulation environment covers these traffic flows:

- **Raw Data Transmit:** Raw data traffic from the Root Port Model through the Endpoint PCIe, Packet DMA, and DDR3 memory to the Loopback module
- **Raw Data Receive:** Raw data traffic from the Loopback module through the DDR3 memory, Packet DMA, and Endpoint PCIe to the Root Port Model

The Root Port Model for PCI Express is a limited test bench environment that provides a test program interface. The purpose of the Root Port Model is to provide a source mechanism for generating downstream PCI Express traffic to simulate the DUT and a destination mechanism for receiving upstream PCI Express traffic from the DUT in a simulation environment.

The out-of-box simulation environment (see Figure 2-18) consists of:

- Root Port Model for PCI Express connected to the DUT
- Transaction Layer Packet (TLP) generation tasks for various programming operations
- Test cases to generate different traffic scenarios

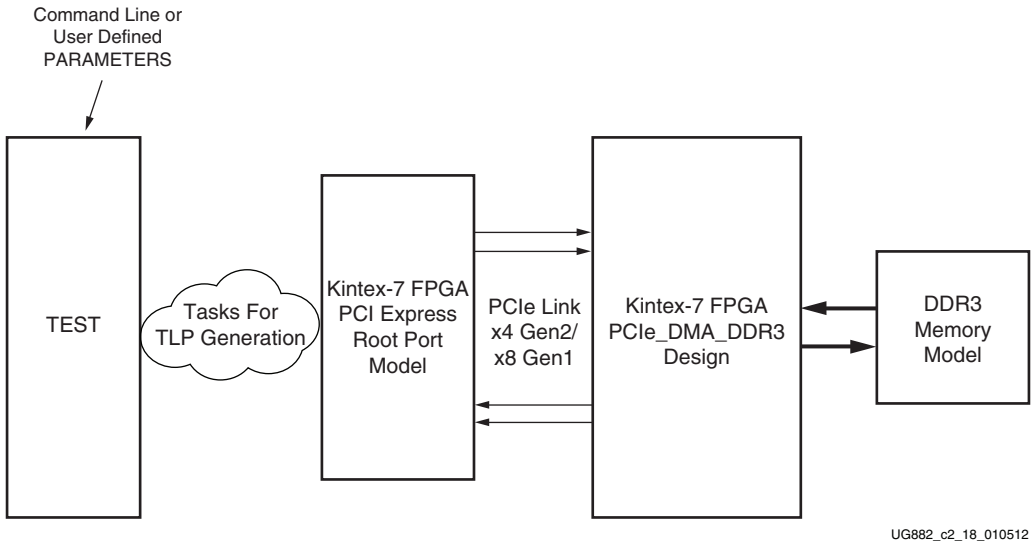


Figure 2-18: **Out-of-Box Simulation Overview**

The simulation environment creates log files during simulation. These log files contain a detailed record of every TLP that was received and transmitted by the Root Port Model.

Simulating the Design

The out-of-box simulation environment is built for the ModelSim simulator. To run the simulation, execute one of the listed scripts at the command prompt. Make sure to compile the required libraries and set the environment variables as per the ModelSim simulator before running the script. Refer to the UG626, *Synthesis and Simulation Design Guide*, which provides information on how to run simulations with different simulators [Ref 5].

- Base TRD with the PCIe link configured as x4 at 5 Gb/s: `simulate_mti_x4gen2` found in the `k7_pcie_dma_ddr3_base/design/sim/mti` directory
- Base TRD with the PCIe link configured as x8 at 2.5 Gb/s: `simulate_mti_x8gen1` found in the `k7_pcie_dma_ddr3_base/design/sim/mti directory`

Note: Before running the simulation script, make sure to generate the MIG core through the CORE Generator tool, as described in Rebuilding the Base TRD, page 26.

User-Controlled Macros

The simulation environment allows the user to define macros that control DUT configuration. These values can be changed in the `user_defines.v` file.

Table 2-2: **User-Controlled Macro Descriptions**

Macro Name	Default Value	Description
CH0	Defined	Enables raw data Path0 initialization and traffic flow.
CH1	Defined	Enables raw data Path1 initialization and traffic flow.
DETAILED_LOG	Not defined	Enables a detailed log of each transaction.

Test Selection

For the raw data path, fixed length packets of 1024 bytes are generated.

Table 2-3 describes the various tests provided by the out-of-box simulation environment.

Table 2-3: **Test Description**

Test Name	Description
basic_test	Basic test. This test runs six packets for each DMA channel. One buffer descriptor defines one full packet in this test.
packet_spanning	Packet spanning multiple descriptors. This test spans a packet across two buffer descriptors. It runs six packets for each DMA channel.
test_interrupts	Interrupt test. This test sets the interrupt bit in the descriptor and enables the interrupt registers. This test also shows interrupt handling by acknowledging relevant registers. Note: Only one channel should be enabled for this test.
dma_disable	DMA disable test. This test shows the DMA disable operation sequence on a DMA channel.
break_loop	Enable checker and generator in hardware and disable loopback. This test shows the receive path running independent of the transmit path. The data source for the receive path is the generator, not the looped back transmit data.

The name of the test to be run can be specified on the command line while invoking relevant simulators in the provided scripts. By default, the simulation script file specifies the basic test to be run using this syntax:

```
+TESTNAME=basic_test
```

The test selection can be changed by specifying a different test case as specified in Table 2-3.

Chapter 3

Functional Description

This chapter describes the hardware design and software driver components. It also describes how the data and control information flow through the various connected IPs.

Hardware Architecture

Figure 3-1 provides a detailed block level overview of the TRD. The base system components and the applications components enable data flow to/from the host memory at high data rates.

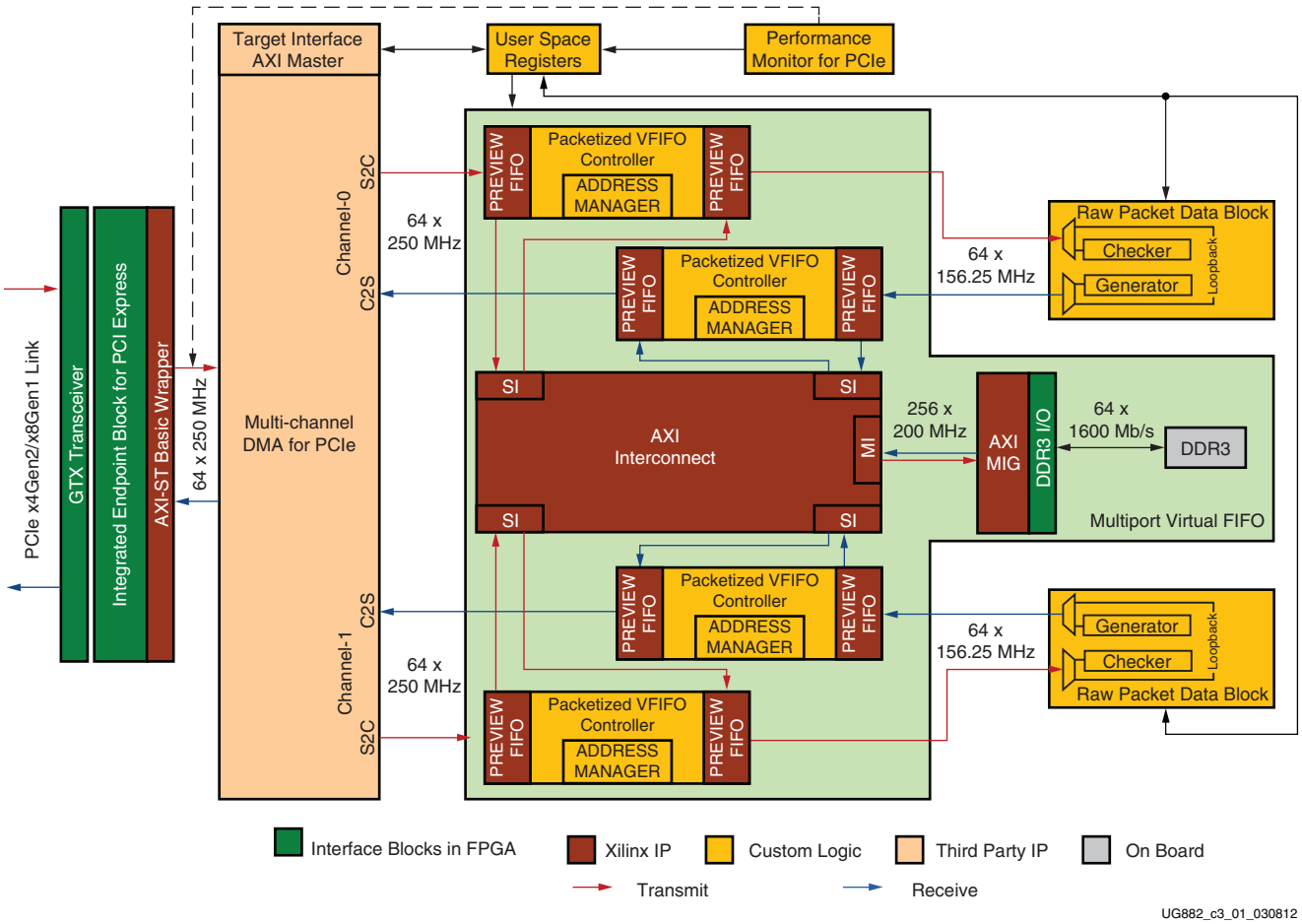


Figure 3-1: **Detailed Design Block Diagram**

The hardware architecture is detailed under these sections:

- Base System Components describes the Kintex-7 FPGA integrated Endpoint block for PCI Express, DMA, and Multiport Virtual FIFO
- Application Components describes a simple packet data generator.

Base System Components

PCI Express is a high-speed serial protocol that allows transfer of data between host systems and Endpoint cards. To efficiently use the processor bandwidth, a bus mastering scatter-gather DMA controller is used to push and pull data from the system memory. All data to and from the system is stored in the DDR3 memory through a Multiport Virtual FIFO abstraction layer before interacting with the user application.

PCI Express

The Kintex-7 FPGA Integrated Block for PCI Express provides a wrapper around the integrated block in the FPGA. The integrated block is compliant with the PCI Express v2.0 specification. It supports x1, x2, x4, x8 lane widths operating at 2.5 Gb/s (Gen1) or 5 Gb/s (Gen2) line rate per direction. The wrapper combines the Kintex-7 FPGA Integrated Block for PCI Express with transceivers, clocking, and reset logic to provide an industry standard AXI4-Stream interface as the user interface.

For details on the Kintex-7 FPGA integrated Endpoint block for PCI Express, refer to UG477, *7 Series FPGAs Integrated Block for PCI Express User Guide* [Ref 4].

Performance Monitor for PCI Express

The monitor block snoops for PCIe® transactions on the AXI4-Stream interface ports and keeps track of utilization. A timer within the block counts out the clocks until one second has elapsed, during which time several counters have collected data about the usage of the transaction layer. Table 3-1 shows the ports on the monitor.

Table 3-1: **Monitor Ports for PCI Express**

Port Name	Type	Description
reset	Input	Synchronous reset
clk	Input	250 MHz clock
Transmit Ports on the AXI4-Stream Interface		
s_axis_tx_tdata[63:0]	Input	Data to be transmitted via PCIe link

Table 3-1: **Monitor Ports for PCI Express *(Cont'd)***

Port Name	Type	Description
s_axis_tx_tkeep[7:0]		The transmit data strobe is used to determine which data bytes are valid on s_axis_tx_tdata during a given beat (this signal is valid only if s_axis_tx_tvalid and s_axis_tx_tready are both asserted). Bit 0 corresponds to the least significant byte on s_axis_tx_tdata and bit 7 corresponds to the most significant byte, for example: • s_axis_tx_tkeep[0] == 1b, • s_axis_tx_tdata[7:0] is valid. • s_axis_tx_tkeep[7] == 0b, • s_axis_tx_tdata[63:56] is not valid. • When s_axis tx_tlast is not asserted, the only valid value is `0xFF`. • When s_axis_tx_tlast is asserted, valid values are `0x0F` and `0xFF`.
s_axis_tx_tlast	Input	End of frame indicator on transmit packets. Valid only along with assertion of s_axis_tx_tvalid.
s_axis_tx_tvalid	Input	Source ready to provide transmit data. Indicates that the DMA is presenting valid data on s_axis_tx_tdata.
s_axis_tx_tuser[3] (src_dsc)	Input	Source discontinue on a transmit packet. Can be asserted any time starting on the first cycle after SOF. s_axis_tx_tlast should be asserted along with s_axis_tx_tuser[3] assertion.
s_axis_tx_tready	Input	Destination ready for transmit. Indicates that the core is ready to accept data on s_axis_tx_tdata. The simultaneous assertion of s_axis_tx_tvalid and s_axis_tx_tready marks the successful transfer of one data beat on s_axis_tx_tdata.
Receive Ports on the AXI4-Stream Interface		
m_axis_rx_tdata[63:0]	Input	Data received on the PCIe link. Valid only if m_axis_rx_tvalid is also asserted.
m_axis_rx_tkeep[7:0]	Input	The receive data keep signal is used to determine which data bytes are valid on m_axis_rx_tdata[63:0] during a given beat (this signal is valid only when m_axis_rx_tvalid and m_axis_rx_tready are both asserted). Bit 0 corresponds to the least significant byte on m_axis_rx_tdata and bit 7 corresponds to the most significant byte. When m_axis rx_tlast is not asserted, this signal can be ignored. When m_axis_rx_tlast is asserted, valid values are `0x0F` and `0xFF`.
m_axis_rx_tlast	Input	End of frame indicator for received packet. Valid only if m_axis_rx_tvalid is also asserted.

Table 3-1: **Monitor Ports for PCI Express *(Cont'd)***

Port Name	Type	Description
m_axis_rx_tvalid	Input	Source ready to provide receive data. Indicates that the core is presenting valid data on m_axis_rx_tdata.
m_axis_rx_tready	Input	Destination ready for receive. Indicates that the DMA is ready to accept data on m_axis_rx_tdata. The simultaneous assertion of m_axis_rx_tvalid and m_axis_rx_tready marks the successful transfer of one data beat on m_axis_rx_tdata.
Byte Count Ports		
tx_byte_count[31:0]	Output	Raw transmit byte count
rx_byte_count[31:0]	Output	Raw receive byte count
tx_payload_count[31:0]	Output	Transmit payload byte count
rx_payload_count[31:0]	Output	Receive payload byte count

Note: Start of packet is derived based on the signal values of source valid, destination ready and end of packet indicator. The clock cycle after end of packet is deasserted and source valid is asserted indicates start of a new packet.

Four counters collect information on the transactions on the AXI4-Stream interface:

- TX Byte Count. This counter counts bytes transferred when the s_axis_tx_tvalid and s_axis_tx_tready signals are asserted between the Packet DMA and the Kintex-7 FPGA Integrated Block for PCI Express. This value indicates the raw utilization of the PCIe transaction layer in the transmit direction, including overhead such as headers and non-payload data such as register access.
- RX Byte Count. This counter counts bytes transferred when the m_axis_rx_tvalid and m_axis_rx_tready signals are asserted between the Packet DMA and the Kintex-7 FPGA Integrated Block for PCI Express. This value indicates the raw utilization of the PCIe transaction layer in the receive direction, including overhead such as headers and non-payload data such as register access.
- TX Payload Count. This counter counts all memory writes and completions in the transmit direction from the Packet DMA to the host. This value indicates how much traffic on the PCIe transaction layer is from data, which includes the DMA buffer descriptor updates, completions for register reads, and the packet data moving from the user application to the host.
- RX Payload Count. This counter counts all memory writes and completions in the receive direction from the host to the DMA. This value indicates how much traffic on the PCIe transaction layer is from data, which includes the host writing to internal registers in the hardware design, completions for buffer description fetches, and the packet data moving from the host to user application.

The actual packet payload by itself is not reported by the performance monitor. This value can be read from the DMA register space. The method of taking performance snapshots is similar to the Northwest Logic DMA performance monitor (refer to the *Northwest Logic DMA Back-End Core User Guide* and *Northwest Logic DMA AXI DMA Back-End Core User Guide*, available in the `k7_pcie_dma_ddr3_base/design/ipcores/dma/doc` directory). The byte counts are truncated to a four-byte resolution, and the last two bits of the register indicate the sampling period. The last two bits transition every second from `00` to `01` to `10` to `11`. The software polls the performance register every second. If the

sampling bits are the same as the previous read, then the software needs to discard the second read and try again. When the one-second timer expires, the new byte counts are loaded into the registers, overwriting the previous values.

Scatter Gather Packet DMA

The scatter-gather Packet DMA IP is provided by Northwest Logic, a Xilinx third-party alliance partner. The Packet DMA is configured to support simultaneous operation of two user applications. This involves four DMA channels: two system-to-card (S2C) or transmit channels and two card-to-system (C2S) or receive channels. The DMA controller requires a 64 KB register space mapped to BAR0. All DMA registers are mapped to BAR0 from `0x0000` to `0x7FFF`. The address range from `0x8000` to `0xFFFF` is available to the user via this interface. Each DMA channel has its own set of independent registers. Registers specific to this TRD are described in Appendix B, Register Description. Further details of various registers can be obtained from the *Northwest Logic DMA Back-End Core User Guide*, available in the `k7_pcie_dma_ddr3_base/design/ipcores/dma/doc` directory.

The front end of DMA interfaces to the AXI4-Stream interface. The back end of the DMA provides an AXI4-Stream interface as well which connects to the ports on Virtual FIFO. Further details of the signal definitions can be obtained from the *Northwest Logic AXI DMA Back-End Core User Guide*, available in the `k7_pcie_dma_ddr3_base/design/ipcores/dma/doc` directory.

Scatter Gather Operation

The term scatter gather refers to the ability to write packet data segments into different memory locations and gather data segments from different memory locations to build a packet. This allows for efficient memory utilization because a packet does not need to be stored in physically contiguous locations. Scatter gather requires a common memory resident data structure that holds the list of DMA operations to be performed. DMA operations are organized as a linked list of buffer descriptors. A buffer descriptor describes a data buffer. Each buffer descriptor is eight doublewords in size (a doubleword is 4 bytes), which is a total of 32 bytes. The DMA operation implements buffer descriptor chaining, which allows a packet to be described by more than one buffer descriptor.

Figure 3-2 shows the buffer descriptor layout for S2C and C2S directions.

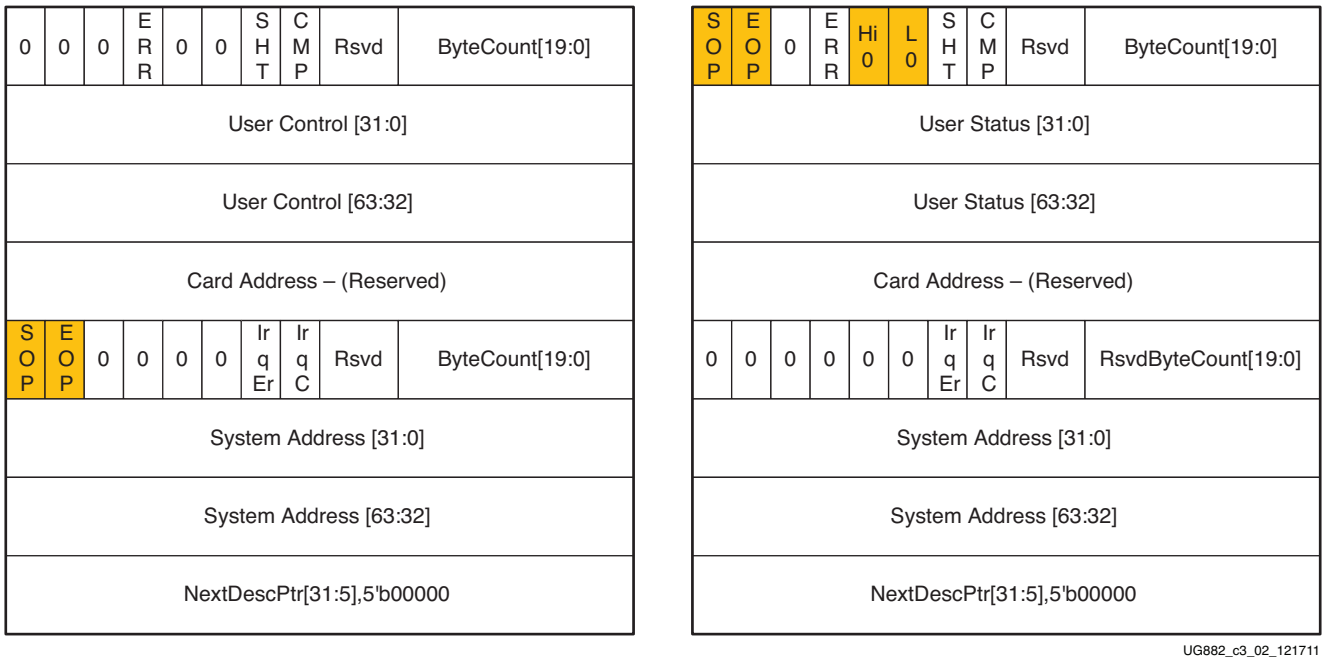


Figure 3-2: **S2C Buffer Descriptor and C2S Buffer Descriptor Layout**

The descriptor fields are described in Table 3-2.

Table 3-2: **Buffer Descriptor Fields**

Descriptor Fields	Functional Description
SOP	Start of packet. In S2C direction, indicates to the DMA the start of a new packet. In C2S, DMA updates this field to indicate to software start of a new packet.
EOP	End of packet In S2C direction, indicates to the DMA the end of current packet. In C2S, DMA updates this field to indicate to software end of the current packet.
ERR	Error This is set by DMA on descriptor update to indicate error while executing that descriptor
SHT	Short Set when the descriptor completed with a byte count less than the requested byte count. This is common for C2S descriptors having EOP status set but should be analyzed when set for S2C descriptors.
CMP	Complete This field is updated by the DMA to indicate to the software completion of operation associated with that descriptor.
Hi 0	User Status High is zero Applicable only to C2S descriptors - this is set to indicate Users Status [63:32] = 0

Table 3-2: **Buffer Descriptor Fields *(Cont'd)***

Descriptor Fields	Functional Description
L 0	User Status Low is zero Applicable only to C2S descriptors - this is set to indicate User Status [31:0] = 0
Irq Er	Interrupt On Error This bit indicates DMA to issue an interrupt when the descriptor results in error
Irq C	Interrupt on Completion This bit indicates DMA to issue an interrupt when operation associated with the descriptor is completed
ByteCount[19:0]	Byte Count In S2C direction, this indicates DMA the byte count queued up for transmission. In C2S direction, DMA updates this field to indicate the byte count updated in system memory.
RsvdByteCount[19:0]	Reserved Byte Count In S2C direction, this is equivalent to the byte count queued up for transmission. In C2S direction, this indicates the data buffer size allocated - the DMA might or might not utilize the entire buffer depending on the packet size.
User Control/User Status	User Control or Status Field (The use of this field is optional.) In S2C direction, this is used to transport application specific data to DMA. Setting of this field is not required by this reference design. In C2S direction, DMA can update application specific data in this field.
Card Address	Card Address Field This is a reserved for Packet DMA
System Address	System Address This defines the system memory address where the buffer is to be fetched from or written to.
NextDescPtr	Next Descriptor Pointer This field points to the next descriptor in the linked list. All descriptors are 32-byte aligned.

This field points to the next descriptor in the linked list. All descriptors are 32-byte aligned.

Packet Transmission

The software driver prepares a ring of descriptors in system memory and writes the start and end addresses of the ring to the relevant S2C channel registers of the DMA. When enabled, the DMA fetches the descriptor followed by the data buffer it points to. Data is fetched from the host memory and made available to the user application through the DMA S2C streaming interface.

The packet interface signals (for example, user control and the end of packet) are built from the control fields in the descriptor. The information present in the user control field is made available during the start of packet. The reference design does not use the user control field.

To indicate data fetch completion corresponding to a particular descriptor, the DMA engine updates the first doubleword of the descriptor by setting the complete bit of the 'Status and Byte Count field to 1. The software driver analyzes the complete bit field to free up the buffer memory and reuse it for later transmit operations.

Figure 3-3 shows the system to card data transfer.

Note: Start of Packet is derived based on the signal values of source valid (s2c_tvalid), destination ready (s2c_tready) and end of packet (s2c_tlast) indicator. The clock cycle after end of packet is deasserted and source valid is asserted indicates start of a new frame.

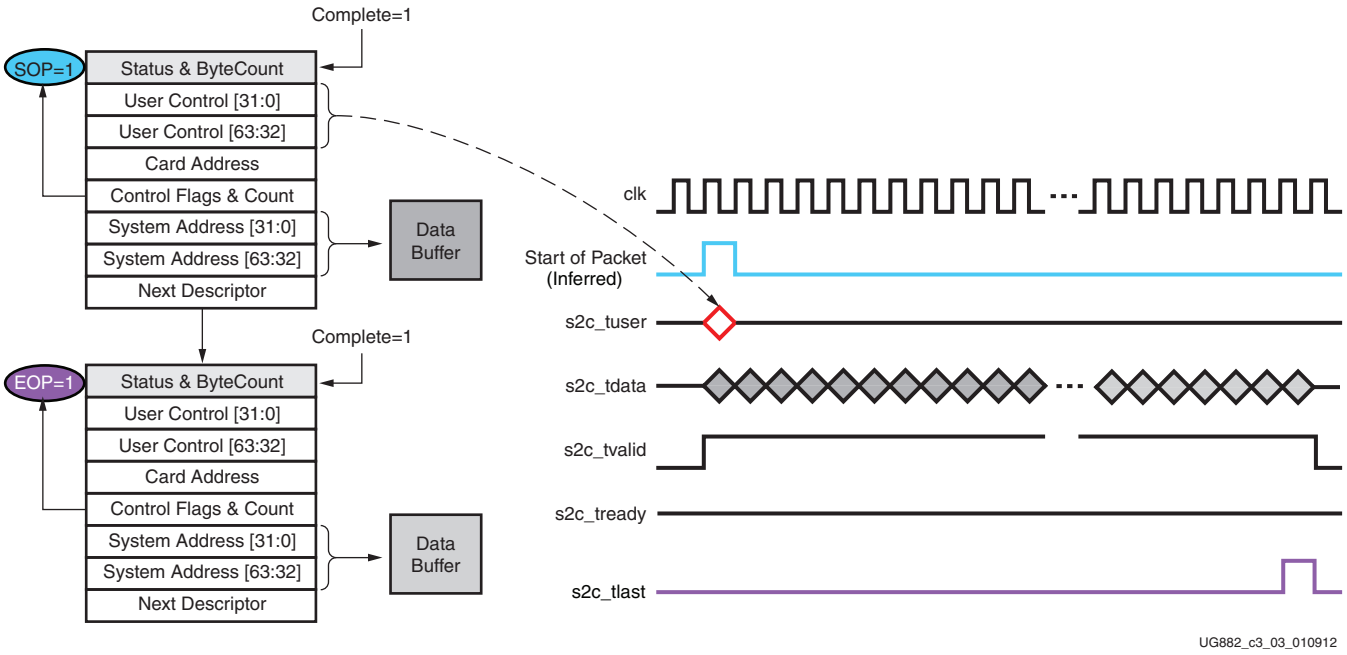


Figure 3-3: **Data Transfer from System to Card**

Packet Reception

The software driver prepares a ring of descriptors with each descriptor pointing to an empty buffer. It then programs the start and end addresses of the ring in the relevant C2S DMA channel registers. The DMA reads the descriptors and waits for the user application to provide data on the C2S streaming interface. When the user application provides data, the DMA writes the data into one or more empty data buffers pointed to by the prefetched descriptors. When a packet fragment is written to host memory, the DMA updates the status fields of the descriptor. The c2s_tuser signal on the C2S interface is valid only during c2s_tlast. Hence, when updating the EOP field, the DMA engine also needs to update the User Status fields of the descriptor. In all other cases, the DMA updates only the Status and Byte Count field. The completed bit in the updated status field indicates to the software driver that data was received from the user application. When the software driver processes the data, it frees the buffer and reuses it for later receive operations.

Figure 3-4 shows the card to system data transfer.

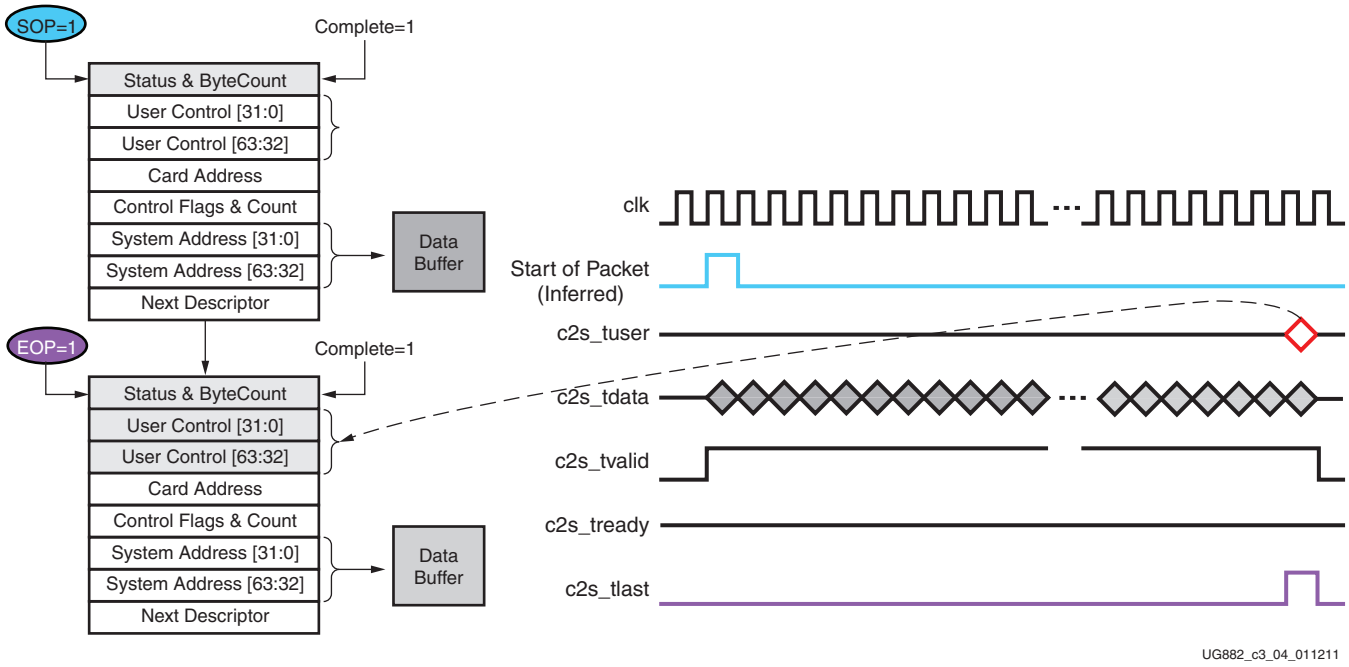


Figure 3-4: **Data Transfer from Card to System**

Note: Start of Packet is derived based on the signal values of source valid (c2s_tvalid), destination ready (c2s_tready) and end of packet (c2s_tlast) indicator. The clock cycle after end of packet is deasserted and source valid is asserted indicates start of a new frame.

The software periodically updates the end address register on the Transmit and Receive DMA channels to ensure uninterrupted data flow to and from the DMA.

Multiport Virtual Packet FIFO

The Multiport Virtual Packet FIFO is built using LogiCORE™ IP MIG (Memory Interface Controller) and LogiCORE IP AXI Interconnect. The Packetized VFIFO controller logic around the IPs converts the external DDR3 memory into a multiport FIFO. Figure 3-5 is the block level representation of Multiport Virtual Packet FIFO.

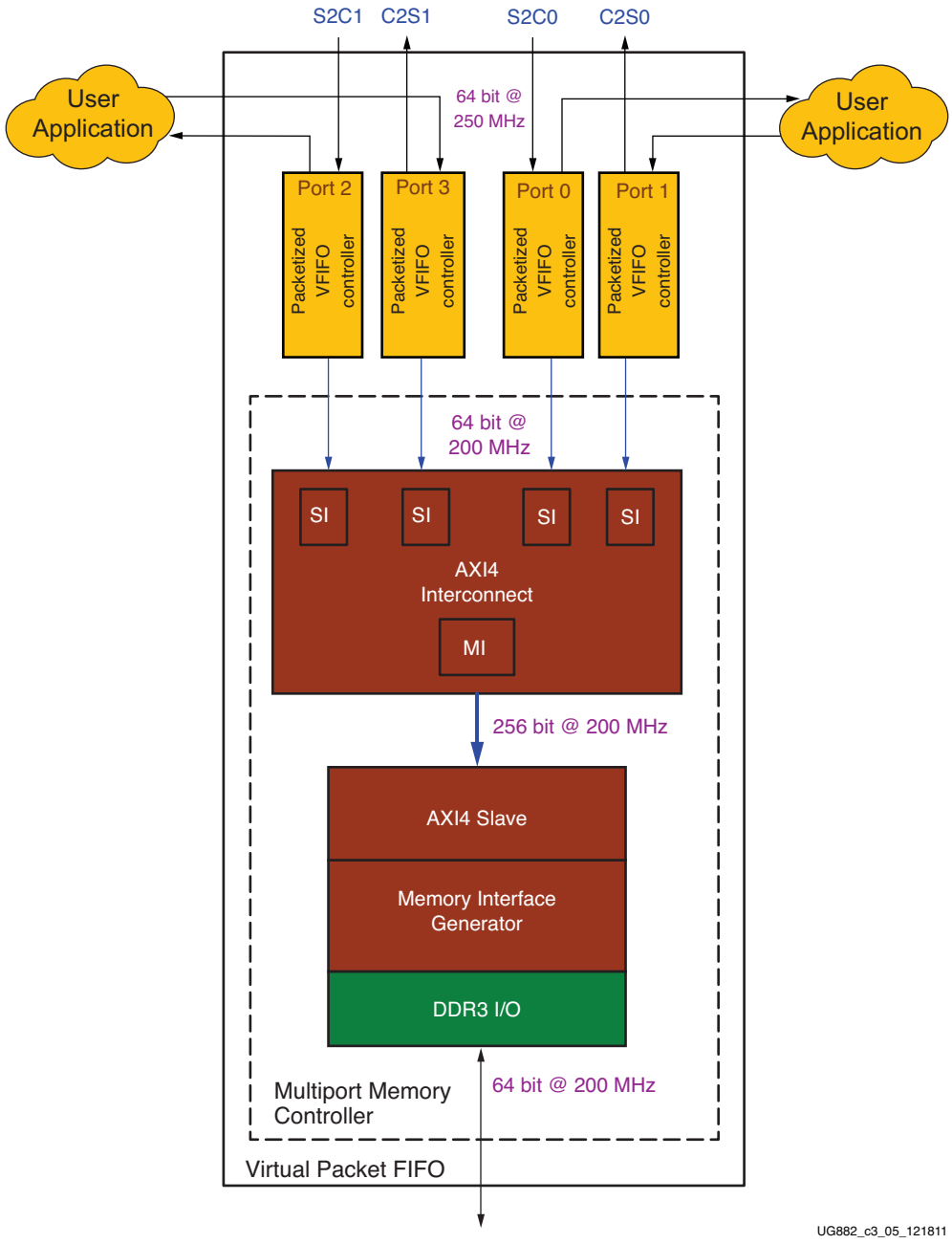


Figure 3-5: **Virtual Packet FIFO**

Multiport Memory Controller

The Memory Interface controller and the AXI Interconnect constitute the Multiport Memory Controller. The LogiCORE IP MIG provides a single port with AXI4 interface. Because this reference design supports two user applications, four ports are required on Memory Controller—two ports for DMA to push data and user applications to pull data and two ports for user application to push data and DMA to pull data. Using the AXI4 interconnect a single port Memory Controller can be converted into a multiport Memory Controller. In this design AXI4 interconnect is used in a 4x1 configuration. The DMA engines and the user applications are 4 masters to drive 4 slave interfaces (SI). The master

interface (MI) on the Interconnect drives the single port Memory Controller Interface which is a slave.

Packetized VFIFO Controller

Figure 3-6 provides a block level overview of the Packetized VFIFO controller.

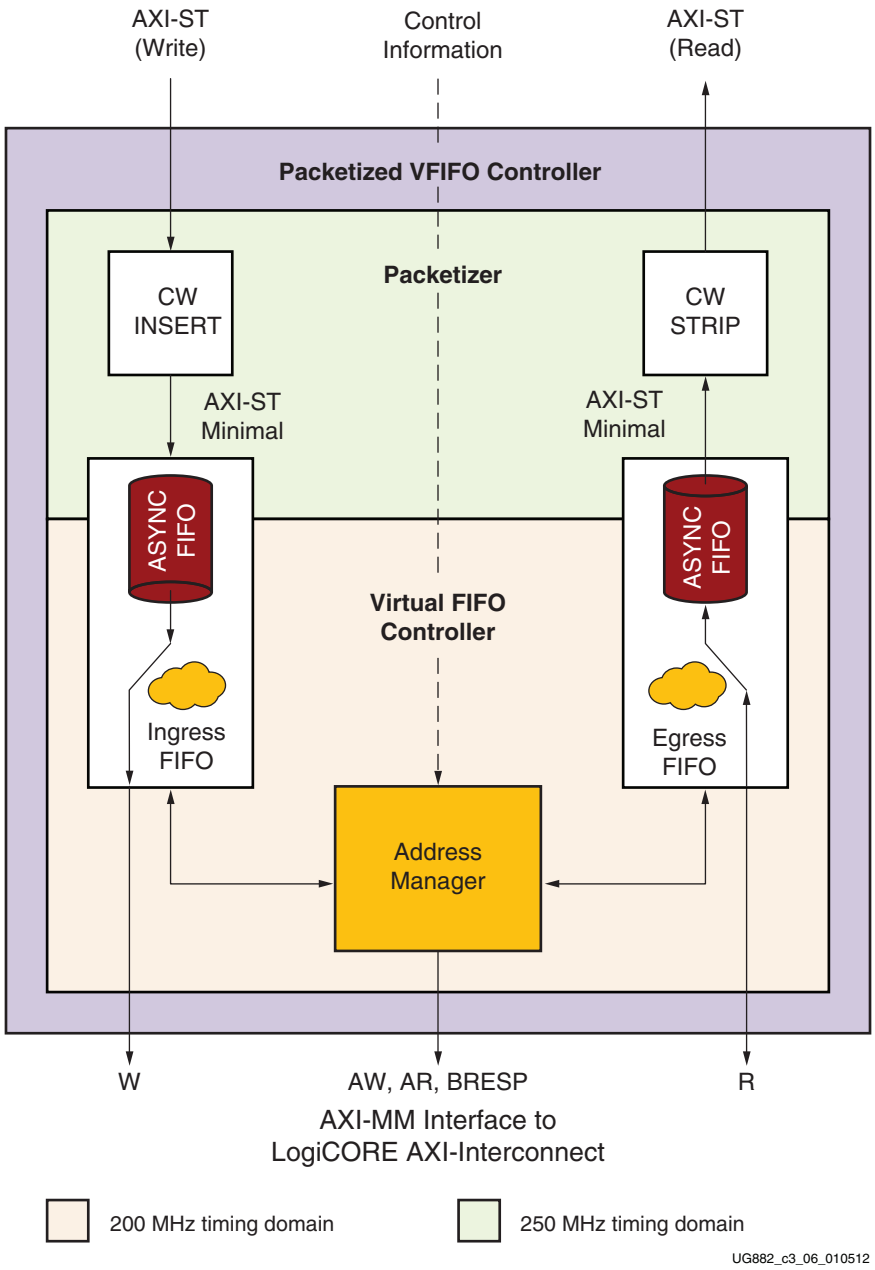


Figure 3-6: **Packetized VFIFO Controller**

The DDR3 memory is required to store packets (size ranging from 64B to 32 KB). Because the interface width on DDR3 is 64-bits, there are no extra bits to store control information. Therefore, the reference design needs a way to find the start and end of the packet and valid data bytes when data is read out of the DDR3. Inserting a control word in a data packet before it is written into the DDR3 and using the control word when data is read out

of the DDR3 is a simple scheme to determine packet delineations. Using the Packetizer logic does not have a large performance impact and avoids use of a store and forward scheme.

The input and output interfaces on Packetized VFIFO controller are AXI-Stream compliant. The DMA engines and the user application interface with this module. On the write port, the packet length should be available with the first data beat to enable control word insertion. This control word is discarded before the packet is made available on the read port.

The Virtual FIFO Controller comprises of three modules Ingress FIFO, Egress FIFO and address manager.

The address manager implements the addressing scheme to manage DDR3 as FIFO. Users have control to set the DDR3 start and end address boundary to be used as FIFO. The users can also set the burst size to be used on write and read AXI-MM interfaces. These values are guidance to what the maximum burst size could be. For example, say burst size is programmed as 256 for both read and write interfaces. Effort is made to operate at this burst size, but a sub-optimal burst (lesser than 256) can be issued based on timeout in lean traffic scenarios

(Refer to Memory Controller Registers in Appendix B to set the start and end addresses and burst size.)

The Ingress and Egress FIFO blocks communicate to the AXI Interconnect block based on the control signals from the address manager.

The Ingress FIFO block handles the write data and is responsible for driving the write interface of AXI-MM. The asynchronous preview FIFO in this block helps with clock domain crossing. It also allows storing up enough data to create a transaction of write burst size and then sending it to the AXI Interconnect.

The Egress FIFO block handles the read data and is responsible for draining the read data interface of AXI-MM after a read command is issued and when read data is available. The asynchronous preview FIFO in this block helps with clock domain crossing. It also allows storing up enough data to store a transaction of read burst size received from AXI Interconnect.

The preview FIFOs in the Ingress FIFO block and Egress FIFO block are generated using the LogiCORE FIFO Generator IP with a data width of 64-bit and depth of 1024. The FIFOs internally use Block RAMs.

Table 3-3 shows the signals on the Multiport Virtual Packet FIFO. The read and writes interface signals and the user-specified register widths scale with the number of ports.

Table 3-3: **Ports on the Packetized VFIFO Controller**

Port Name	Type	Description
Write Interface		
axi_str_wr_tlast	Input	Indicates end of a packet
axi_str_wr_tdata	Input	Data packets received from the DMA engines and User Applications
axi_str_wr_tvalid	Input	Source is ready, and data is valid on the data bus
axi_str_wr_tready	Output	Destination is ready, and can receive data
axi_str_wr_tkeep	Input	Byte enables on data, qualified by tvalid. Can be deasserted only when tlast is asserted

Table 3-3: **Ports on the Packetized VFIFO Controller *(Cont'd)***

Port Name	Type	Description
axi_str_wr_aclk	Input	Clock for write transactions
axi_str_wr_tuser	Input	Length of the data packet is defined by this signal. This information should be available on the first data beat of a packet.
wr_reset_n	Input	User reset to the write port
Read Interface		
axi_str_rd_tlast	Output	Indicates end of a packet
axi_str_rd_tdata	Output	Data transmitted to the DMA engines and User Applications
axi_str_rd_tvalid	Output	Source is ready, and data is valid on the data bus
axi_str_rd_tready	Input	Destination is ready, and can receive data
axi_str_rd_tkeep	Output	Byte enables on data, qualified by tvalid. Can be deasserted only when tlast is asserted
axi_str_rd_aclk	Input	Clock for read transactions
axi_str_rd_tuser	Output	Length of the data packet is defined by this signal. This information is available on the first data beat of a packet.
rd_reset_n	Input	User reset to the read port
User-Specified Registers		
start_addr	Input	DDR3 start address boundary
end_addr	Input	DDR3 end address boundary
wrburst_size	Input	Write burst size on the AXI-MM interface to the LogiCOREIP AXI Interconnect
rdburst_size	Input	Read burst size on the AXI-MM interface to the LogiCORE IP AXI Interconnect
DDR3 Memory Interface		
ddr_addr	Output	PHY signal
ddr_ba	Output	PHY signal
ddr_cas_n	Output	PHY signal
ddr_ck_p	Output	PHY signal
ddr_ck_n	Output	PHY signal
ddr_cke	Output	PHY signal
ddr_cs_n	Output	PHY signal
ddr_dm	Output	PHY signal
ddr_odt	Output	PHY signal
ddr_ras_n	Output	PHY signal
ddr_reset_n	Output	PHY signal

Table 3-3: **Ports on the Packetized VFIFO Controller *(Cont'd)***

Port Name	Type	Description
ddr_we_n	Output	PHY signal
ddr_dq	Inout	PHY signal
ddr_dqs_p	Inout	PHY signal
ddr_dqs_n	Inout	PHY signal
sda	Inout	PHY signal
scl	Output	PHY signal
Other		
calib_done	Output	DDR3 Memory calibration is complete
clk_ref_p	Input	200 MHz differential clock
clk_ref_n	Input	200 MHz differential clock
mcb_clk	Output	User interface 200 MHz clock for the LogiCORE IP MIG
mcb_rst	Output	User interface reset for the LogiCORE IP MIG
ddr3_fifo_empty	Output	Indicates the DDR3 FIFOs, Egress and Ingress preview FIFOs are empty. Width of the signals is equal to the number of ports on the Virtual Packet FIFO.
axi_ic_shim_rst_n	Input	This signal resets the AXI interconnect IP and AXI interface of the MIG IP. The software verifies the DDR3 FIFO and preview FIFOs are empty before issuing this reset.
user_reset	Input	This signal resets the MIG IP core. It is connected PCIe Endpoint card reset (perst_n) driven by the PC motherboard.

Application Components

This section describes the block that interfaces with the base components to support Raw Packet Data flow. It is a simple application which can be replaced with any other application protocol like XAUI or Aurora.

Raw Data Packet Path

The Raw Data application is an example of a data streaming protocol. Because the DMA provides a packetized interface on its back end, a fixed length packet is defined on this path, though the data itself does not have any packet annotations in the user space. The fixed length is configurable through a register write (refer to Packet Length (0x9104) in Appendix B and Packet Length (0x9204) in Appendix B for details).

The Raw Data Packet module implements a loopback function, a data checker function, and a data generator function. The module enables specific functions depending on the GUI configuration options selected by the user. On the transmit path, the data checker verifies the data transmitted from the host system via the Packet DMA. On the receive path, data can be sourced either by the data generator or transmit data can be looped back and sent to the host system.

Based on user inputs, the driver programs user space registers to enable checker, enable generator, or enable loopback (see Enable Generator (0x9100), Enable Loopback/Checker

(0x9108), Enable Generator (0x9200), Enable Loopback/Checker (0x9208) in Appendix B, Register Description).

If the Enable Loopback bit is set, as soon as Virtual FIFO receive is ready, the block is ready to accept data from the Virtual FIFO transmit. The data on the Virtual FIFO transmit is then passed on to Virtual FIFO receive. This cycles the data from one port of the Virtual FIFO back into another port (Port 0 to Port 1 and Port 2 to Port 3 as shown in Figure 3-5) with no change to the data. In the loopback mode, data is not verified by the checker; the software driver on the receive end checks for data integrity.

If the Enable Checker bit is set, as soon as data is valid on the Virtual FIFO transmit (Port 0 and 2 in Figure 3-5) each data byte received is checked against a fixed data pattern. If there is a mismatch during a comparison, the data_mismatch signal is asserted. This signal can be accessed through the register space (see Checker Status (0x910C) and Checker Status (0x920C) in Appendix B).

If the Enable Generator bit is set and the Virtual FIFO receive is ready to accept data, the data produced by the generator is passed to the Virtual FIFO (Port 1 and 3 in Figure 3-5). The data from the generator also follows the same data pattern as the checker. The data received and transmitted by the module is divided into packets. The first two bytes of each packet define the length of packet. All other bytes carry the tag/sequence number of the packet. The tag number increases by one per packet. Table 3-4 shows the packet format used in the Raw Packet Data module.

Table 3-4: **Packet Format**

[63:56] [48:55]	[47:40] [39:32]	[31:24] [23:16]	[15:8] [7:0]
TAG	TAG	TAG	PACKET LEN
TAG	TAG	TAG	TAG
-	-	-	-
-	-	-	-
TAG	TAG	TAG	TAG

Note: The data uses a fixed pattern to enable data checking. The data could be any random data otherwise.

Table 3-5 shows the ports on the Raw Packet Data module.

Table 3-5: **Ports on Raw Packet Data Module**

Port Name	Type	Description
clk	Input	250 MHz clock
reset	Input	Synchronous reset
Read Interface		
axi_str_tx_tdata	Input	Data available from VFIFO transmit
axi_str_tx_tkeep	Input	Number of bytes valid per data beat on axi_str_tx_tdata
axi_str_tx_tvalid	Input	Indicates data on axi_str_tx_tdata is valid
axi_str_tx_tlast	Input	Indicates the end of packet on axi_str_tx_tdata
axi_str_tx_tuser	Input	VFIFO passes the length of the packet being transmitted

Table 3-5: **Ports on Raw Packet Data Module *(Cont'd)***

Port Name	Type	Description
axi_str_tx_tready	Output	Indicates the Raw Packet Data module is ready to accept data from VFIFO transmit
Write Interface		
axi_str_tx_tdata	Output	Data available for VFIFO receive
axi_str_tx_tkeep	Output	Number of bytes valid per data beat on axi_str_rx_tdata
axi_str_tx_tvalid	Output	Indicates data on axi_str_rx_tdata is valid
axi_str_tx_tlast	Output	Indicates the end of packet on axi_str_tx_tdata
axi_str_tx_tuser	Output	Raw Packet Data module passes the length of the packet to VFIFO receive
axi_str_tx_tready	Input	Indicates VFIFO receive is ready to accept data from Raw Packet Data module
Register Interface		
enable_loopback	Input	Enables looping back of transmit data
enable_generator	Input	Enables data generator
pkt_len	Input	Length of packets produced by the generator
enable_checker	Input	Enable data checker
data_mismatch	Output	Indicates the transmit data is incorrect

Clocking

This section describes the clocking requirements for this Kintex-7 FPGA Base TRD.

Two differential clocks are needed in this TRD:

- 200 MHz clock and 800 MHz clock for the Memory Controller
- 100 MHz clock for PCI Express integrated Endpoint block

The KC705 board used for this reference design has a 100 MHz differential clock coming from the PCIe edge connector is passed on to the PCIe wrapper. The 200 MHz differential clock for the DDR3 Memory Controller comes from an oscillator on the KC705 board, and an 800 MHz clock is generated inside the Memory Controller by adjusting the MMCM multipliers and dividers.

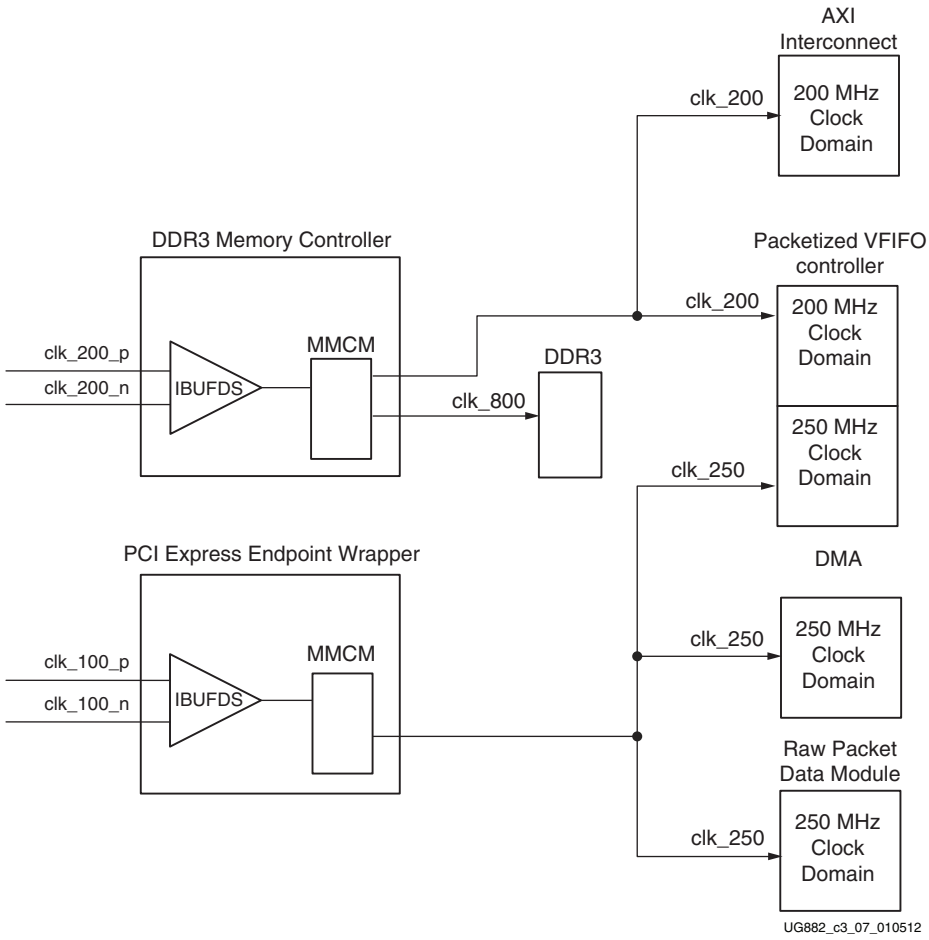


Figure 3-7: **Clocking Diagram**

Figure 3-7 shows the clocking connections. The wrapper for PCI Express generates a 250 MHz single-ended clock that goes to the DMA, Packetized Virtual FIFO Controller, and Raw Packet Data modules. The DDR3 Memory Controller generates a single-ended 200 MHz clock for the Packetized Virtual FIFO Controller and AXI Interconnect, and an 800 MHz single/differential clock for various parts of the Memory Controller and the external DDR3 device.

Resets

This section describes the reset requirements for Kintex-7 FPGA Base TRD.

Table 3-6: **Resets by Function**

Modules	PERSTn Asserted	PCIe Link Down	Software Requests DMA Abort
PCIe Wrapper	X		
DMA IP	X	X	
DDR3 Memory Controller IP	X		
AXI Interconnect	X		

Table 3-6: **Resets by Function *(Cont'd)***

Modules	PERSTn Asserted	PCIe Link Down	Software Requests DMA Abort
Packetized VFIFO controller	X	X	X
Raw Packet Data	X	X	X

Table 3-6 shows how the different blocks get reset depending on the events that can happen. The primary reset for the Kintex-7 FPGA Base TRD is driven from the PERSTn pin of the PCIe edge connector. When this asynchronous pin is active (Low), the Kintex-7 FPGA Integrated Block for PCI Express, GT transceivers for PCIe and DDR3 Memory Controller IP are held in reset. When PERSTn is released, the initialization sequences start on these blocks. The initialization sequence for each of these blocks takes a long time, which is why they get the PERSTn pin directly. Each of these blocks has an output that reflects the status of its initialization sequence. PCIe asserts user_lnk_up, and the Memory Controller asserts init_calib_complete when the respective initialization is complete. These status signals are combined to generate the user logic resets. Figure 3-8 shows the connections for the resets used in the design.

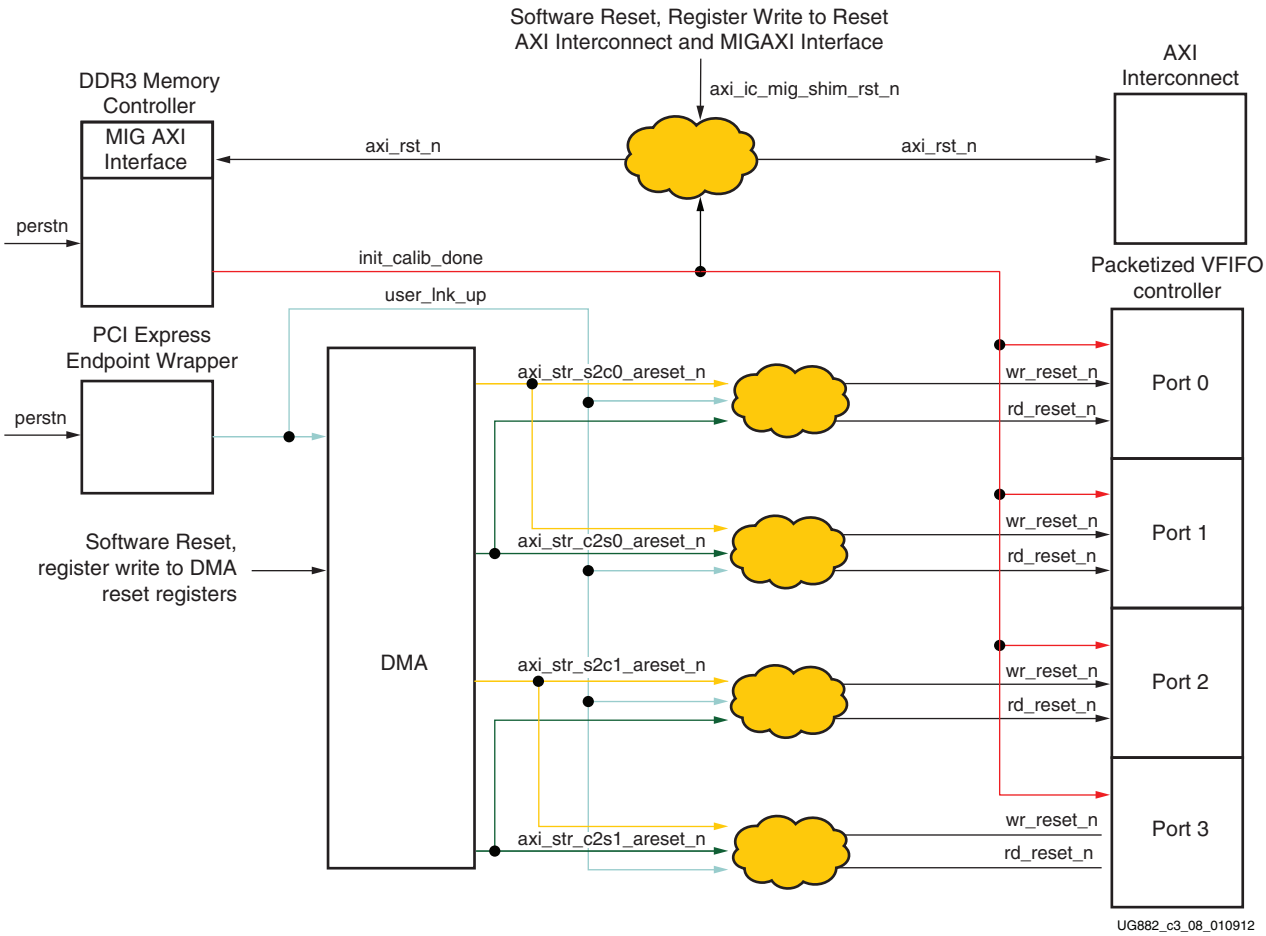


Figure 3-8: **Reset Diagram**

In addition, a software reset is implemented through the Packet DMA ports axi_str_s2c_areset_n and axi_str_c2s_areset_n, for each DMA channel. It is used when the software wants to reset the entire design without bringing the PCIe link down. This is done when unloading the driver. The DMA reset signals, resets the Packetized VFIFO controller and the Raw Packet Data modules. Another reset axi_ic_mig_shim_rst_n is issued by the software when unloading the driver. The software makes sure that DDR3 FIFO and the preview FIFOs are empty before resetting the AXI Interconnect IP and the MIG AXI interface. This is to ensure that all packets were received by the software and there are no partial packets left in the FIFOs.

Software Architecture

Figure 3-9 shows the software components of the Kintex-7 FPGA Base TRD. The software comprises several Linux kernel-space drivers and a user-space application.

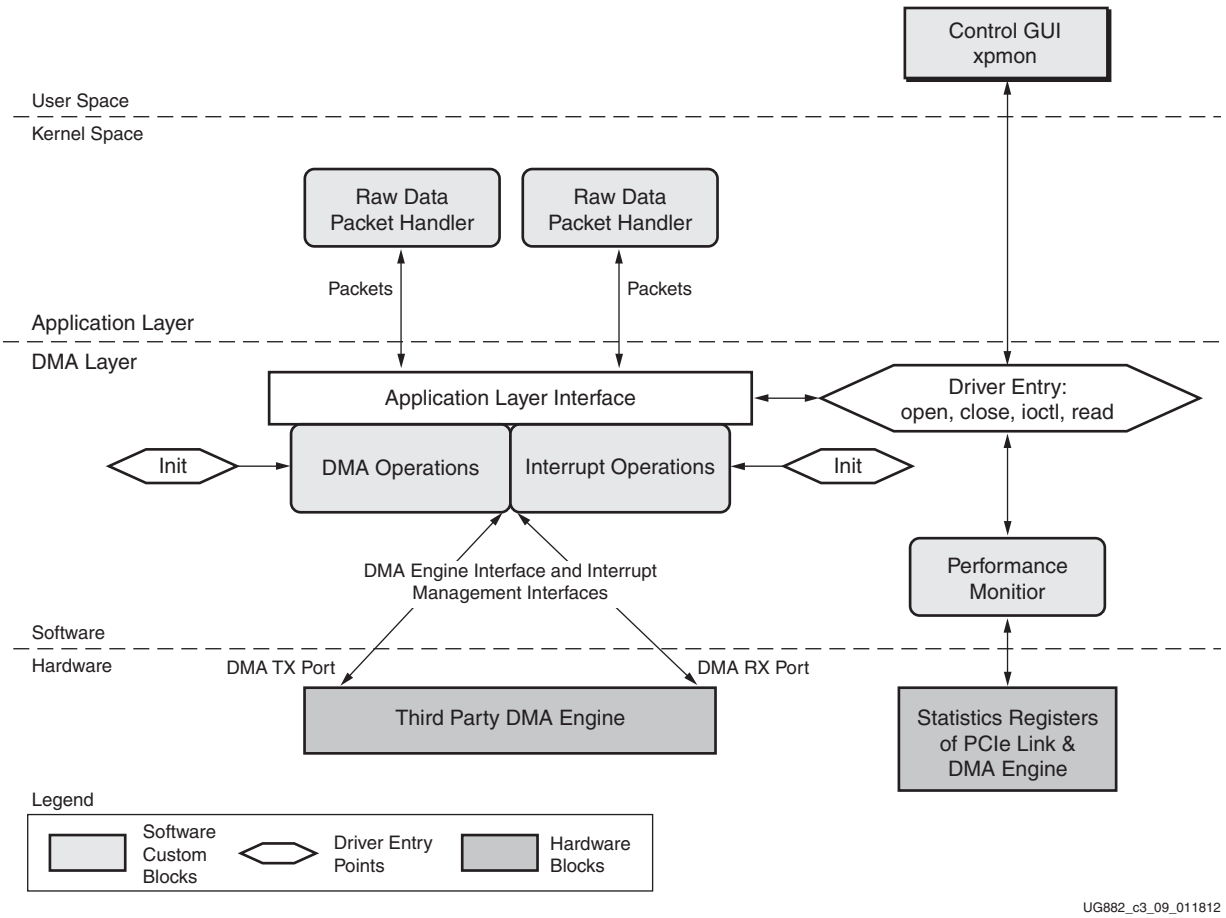


Figure 3-9: **Software Architecture Overview**

Kernel-space Drivers are responsible for:

- Configuration of the DMA engine to enable data transfer between the hardware design and main system memory

- Generation and transfer of raw data streams from host memory to hardware (Transmit). Transfer of the looped or generated streaming data back to the host memory (Receive).

User-space Application (xpmon) is a graphical user interface (GUI) used to:

- Manage the driver and device - for example setting configuration controls for packet generation and display options
- Display of performance statistics reported by PCIe performance monitor and DMA performance monitor

The software developed:

- Can generate adequate data to enable the hardware design to operate at throughput rates of up to 10 Gb/s end to end.
- Showcases the ability of the multi-channel DMA to transfer large amounts of data.
- Provides a user interface that is easy to use and intuitive.
- Is modular and allows for reuse in similar designs.

Kernel Components

Driver Entry Points

The driver has several entry points, some of which are described here. The system invokes the driver entry function when a hardware match is detected after driver insertion (when the PCIe device probed by the driver is found). After reading the device's configuration space, various initialization actions are done. These are initialization of the DMA engine(s), setting up of receive and transmit buffer descriptor rings, and, finally, initialization of interrupts. The other driver entry points are when the GUI starts up and shuts down; when a new performance test is started or stopped; and to convey periodic status information and performance statistics results to the GUI.

On a Linux OS, the system invokes the probe() function when a hardware match is detected. A device node is created for xdma (the node name is fixed and the major/minor numbers are allocated by the system). The base DMA driver appears as a device table entry in Linux.

DMA Operations

For each DMA channel, the driver sets up a buffer descriptor ring. At initialization, the receive ring (associated with a C2S channel) is fully populated with buffers meant to store incoming packets, and the full receive ring is submitted for DMA. On the other hand, the transmit ring (associated with S2C channel) is empty. As packets arrive for transmission, they are added to the buffer descriptor ring, and submitted for DMA.

Raw Data Packet Handler

Data payload for raw data flow is being generated and consumed in two instances of the Raw Data Packet Handler. These are referred to as xrawdata0 and xrawdata1 drivers. When a test is started, data buffers are generated of a fixed size based on user selection and then queued for transmit DMA. The hardware design loops this data back through the Raw Packet Data module, and the data buffers arrive in the system as receive DMA. The handler does a data integrity check on the received data after which it discards the data and returns the buffer to a free pool for future use.

Interrupt Operations

If interrupts are enabled (by setting the compile-time macro TH_BH_ISR), the interrupt service routine (ISR) handles interrupts from the DMA engine and other errors from hardware, if any. The driver sets up the DMA engine to interrupt after every N descriptors that it processes. This value of N can be set by a compile-time macro. The ISR invokes the functionality in the block handler routines pertaining to handling received data and housekeeping of completed transmit and receive buffers.

Performance Monitor

The Performance Monitor is a handler that reads all the performance-related registers (PCIe link level, DMA Engine level). Each of these is read periodically at an interval of one second.

User Space Components

The Control & Monitor GUI (xpmon) is a graphical user interface tool used to monitor device status, run performance tests and display statistics. It conveys the user-configured test parameters to the DMA driver, which in turn passes this information to the Raw Data Packet driver through user registered functions. The Raw Data Packet drivers start an appropriate test according to the test value set by the user. Performance statistics gathered during the test are periodically conveyed to the GUI through the base DMA driver, where they are displayed in several graphs. For screen captures of the graphs, refer to Chapter 2, Getting Started.

The GUI uses the OS-specific methods to communicate with the driver, which results in the appropriate driver entry points being invoked.

Control

The GUI allows the user to specify these items before starting a test:

- Packet size
- Enable Loopback or Enable Generator or Enable Checker

When the user starts a test, the GUI informs the DMA driver of the parameters of the test (unidirectional or bidirectional, the fixed buffer size). The driver sets up the test parameters and informs the Raw Data Packet Handler, which then starts setting up data buffers for transmission, reception or both. Similarly, if the user were to abort a test, the GUI informs the driver, which stops the packet generation mechanism. The test is aborted by stopping the transmit side flow, and then allowing the receive side flow to drain.

Monitor

The driver always maintains information on the status of the hardware. The GUI periodically invokes an ioctl() to read this status information.

- PCIe link status, device status
- DMA Engine status
- BDs and buffer information from drivers
- Interrupt status

The driver maintains a set of arrays to hold per-second sampling points of different kinds of statistics, which are periodically collected by the performance monitor handler. The

arrays are handled in a circular fashion. The GUI periodically invokes an ioctl() to read these statistics, and then displays them.

- PCIe link statistics provided by hardware
- DMA engine statistics provided by DMA hardware
- Graph display of all of the above

Figure 3-10 shows a screen capture of the GUI with the **`System Status`** tab selected.

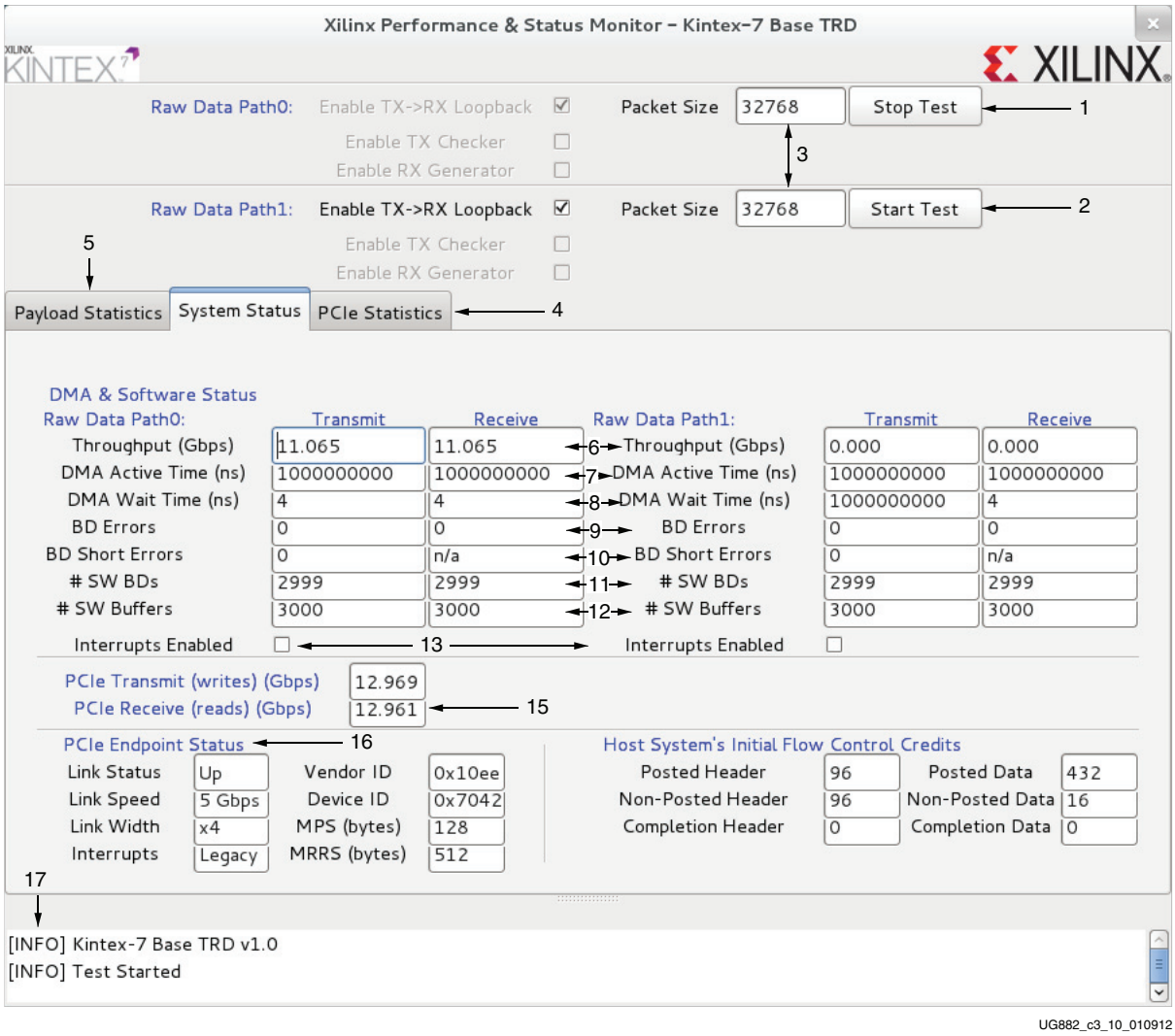


Figure 3-10: **Software Application Screen Capture**

The GUI Fields (indicated by the numbers in the Figure 3-10) are explained here.

1. **`Stop Test:`** Test start/stop control for raw data Path0.
2. **`Start Test:`** Test start/stop control for raw data Path1.
3. **`Packet Size:`** Fixed packet size selection in bytes for the raw data path.
4. **`PCIe Statistics tab:`** Plots the PCIe transactions on the AXI4-Stream interface.
5. **`Payload Statistics tab:`** Shows the payload statistics graphs based on DMA engine performance monitor.

6. **Throughput (Gb/s):** DMA payload throughput in gigabits per second for each engine.
7. **DMA Active Time (ns):** The time in nanoseconds that the DMA engine has been active in the last second.
8. **DMA Wait Time (ns):** The time in nanosecond that the DMA was waiting for the software to provide more descriptors.
9. **BD Errors:** Indicates a count of descriptors that caused a DMA error. Indicated by the error status field in the descriptor update.
10. **BD Short Errors:** Indicates a short error in descriptors in the transmit direction when the entire buffer specified by length in the descriptor could not be fetched. This field is not applicable for the receive direction.
11. **# SW BDs:** Indicates the count of total descriptors set up in the descriptor ring.
12. **# SW Buffers:** Indicates the count of total data buffers associated with the ring.
13. **Interrupts Enabled:** Indicates the interrupt enable status for that DMA engine. The driver enables interrupts on a DMA engine by writing to the DMA engine's register space. To enable interrupts, the compile-time macro TH_BH_ISR needs to be set.
14. **PCIe Transmit (writes) (Gb/s):** Reports the transmit (Endpoint card to host) utilization as obtained from the PCIe performance monitor in hardware.
15. **PCIe Receive (reads) (Gb/s):** Reports the receive (host to Endpoint card) utilization as obtained from the PCIe performance monitor in hardware.
16. **PCIe Endpoint Status:** Reports the status of various PCIe fields as reported in the Endpoint's configuration space. Host System's Initial Flow Control Credits. Initial Flow control credits advertised by the host system after link training with the Endpoint. A value of zero implies infinite flow control credits.
17. The text pane at the bottom shows informational messages, warnings, or errors.

GUI programming environment

After looking at various options, it was decided to choose GTK+ as the GUI programming environment. It has several advantages

- GTK+ libraries are native to Linux. Nothing has to be installed for basic features. This makes it easy to distribute source code and binaries for the GUI.
- It supports C/C++ programming.
- The code can be reused on Windows (where GTK+ would need to be installed).
- It is widely used and popular in the Linux community and is free.

DMA Descriptor Management

This section describes the DMA operation in terms of the descriptor management. It also describes data alignment needs of the DMA engine.

Traffic patterns could be bursty or sustained. To deal with different traffic scenarios, the software does not decide in advance the number of packets to be transferred, and accordingly sets up a descriptor chain for it. Packets can fit in a single descriptor, or might be required to span across multiple descriptors. Also, on the receive side, the actual packet might be smaller than the original buffer provided to accommodate it.

It is therefore required that:

- The software and hardware are each able to independently work on a set of buffer descriptors in a supplier-consumer model
- The software is informed of packets being received and transmitted as it happens
- On the receive side, the software needs a way of knowing the size of the packet

The rest of this section describes how the driver uses the features provided by DMA to achieve the above requirements. Refer to Scatter Gather Packet DMA, page 39 and the *Northwest Logic Packet DMA User Guide* to get an overview of the DMA descriptors and DMA register space [Ref 17].

Dynamic DMA Updates

This section describes how the descriptor ring is managed in the Transmit or System-to-Card (S2C) and Receive or Card-to-System (C2S) directions.

Initialization Phase

Driver prepares descriptor rings for each DMA channel, each containing a number of descriptors that can be set via a compile-time macro. In the current design, the driver thus prepares four rings.

Transmit (S2C) Descriptor Management

In Figure 3-11, the dark blocks indicate descriptors that are under hardware control, and the light blocks indicate descriptors that are under software control.

Table 3-7 presents some of the terminology used in this section.

Table 3-7: **Terminology Summary**

Term	Description
HW_Completed	Register with the address of the last descriptor that DMA engine has completed processing
HW_Next	Register with the address of the next descriptor that DMA engine processes
SW_Next	Register with the address of the next descriptor that software submits for DMA

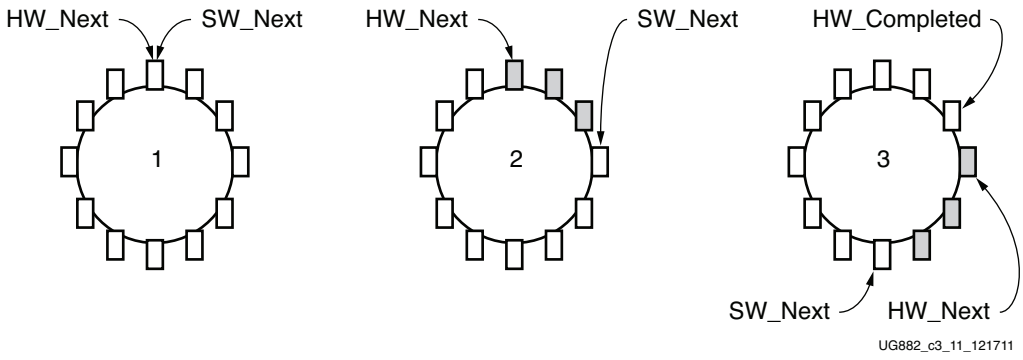


Figure 3-11: **Transmit Descriptor Ring Management**

S2C Initialization Phase

- Driver initializes HW_Next and SW_Next registers to start of ring.
- Driver resets HW_Completed register.

- Driver initializes and enables DMA engine.

Packet Transmission

- Packet is generated by the packet handler.
- Packet is attached to one or more descriptors in ring.
- Driver marks SOP, EOP and IRQ_on_completion in descriptors.
- Driver updates SW_Next register.

Post-Processing

- Driver checks for completion status in descriptor.
- Driver frees packet buffer.

This process continues as the driver keeps adding packets for transmission, and the DMA engine keeps consuming them. Because the descriptors are already arranged in a ring, post-processing of descriptors is minimal and dynamic allocation of descriptors is not required.

Receive (C2S) Descriptor Management

In Figure 3-12, the dark blocks indicate descriptors that are under hardware control, and the light blocks indicate descriptors that are under software control.

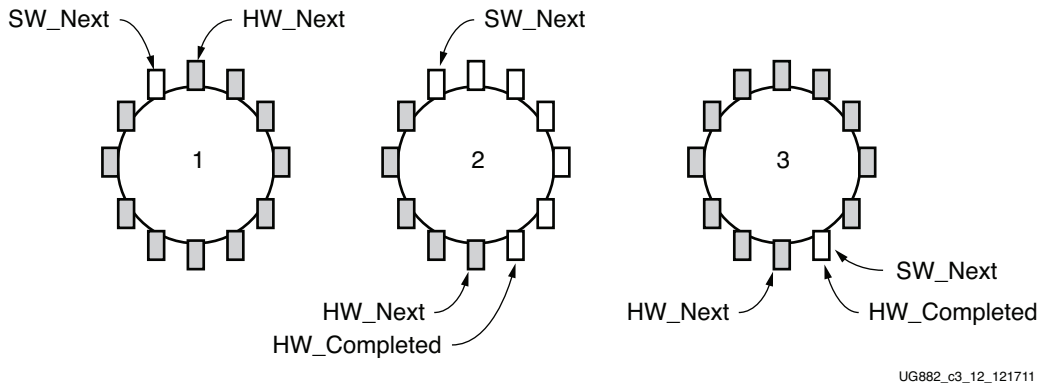


Figure 3-12: **Receive Descriptor Ring Management**

C2S Initialization Phase

- Driver initializes each receive descriptor with an appropriate Data buffer.
- Driver initializes HW_Next register to start of ring and SW_Next register to end of ring.
- Driver resets HW_Completed register.
- Driver initializes and enables DMA engine.

Post-Processing after Packet Reception

- Driver checks for completion status in descriptor.
- Driver checks for SOP, EOP and User Status information.
- Driver discards the completed packet buffer(s).
- Driver allocates new packet buffer for descriptor.
- Driver updates SW_Next register.

This process continues as the DMA engine keeps adding received packets in the ring, and the driver keeps consuming them. Because the descriptors are already arranged in a ring, post-processing of descriptors is minimal and dynamic allocation of descriptors is not required.

XILINX®

Chapter 4

Performance Estimation

This chapter presents a theoretical estimation of performance on the PCI Express interface and the Packetized Virtual FIFO. It also presents a method to measure performance.

PCI Express Performance

PCI Express is a serialized, high bandwidth and scalable point-to-point protocol which provides highly reliable data transfer operations. The maximum transfer rate for a device that is version 2.0 compliant per lane is 2.5 Gb/s at Gen1 and 5 Gb/s at Gen2. This rate is the raw bit rate per lane per direction and not the actual data transfer rate. The effective data transfer rate is lower due to protocol overheads and other system design tradeoffs. Refer to the white paper WP350, *Understanding Performance of PCI Express Systems*, for more information [Ref 6].

The PCI Express link performance together with scatter-gather DMA is estimated under these assumptions:

- Each buffer descriptor points to a 4 KB data buffer space.
- Maximum Payload Size (MPS) = 128B
- Maximum Read Request Size (MRRS) = 128B
- Read Completion Boundary (RCB) = 64B
- Transaction layer packets (TLPs) of three data words (3DWs) considered without extended cyclic redundancy check (ECRC), overhead = 20B
- One ACK packet (acknowledgment packet) assumed per TLP, DLLP (ACK is a Data link layer packet) overhead = 8B
- Update FC DLLPs are not accounted for, but they do affect the final throughput slightly.

The performance is projected by estimating overheads, then calculating the effective throughput by deducting these overheads.

These conventions are used in the calculations in Table 4-1 and Table 4-2:

Term	Description
MRD	Memory Read transaction
MWR	Memory Write transaction
CPLD	Completion with Data
C2S	Card to System
S2C	System to Card

Calculations are done considering unidirectional data traffic that is either transmit (data transfer from System to Card) or receive (data transfer from Card to System).

Traffic on the upstream (Card to System) PCIe link is bolded and traffic on the downstream (System to Card) PCIe link is italicized.

The C2S DMA engine (which deals with data reception, i.e., writing data to system memory) first does a buffer descriptor fetch. Using the buffer address in the descriptor, it issues Memory Writes to the system. After the actual payload in transferred to the system, it sends a Memory Write to update the buffer descriptor. Table 4-1 shows the overhead incurred during data transfer in the C2S direction.

Table 4-1: **PCI Express Performance Estimation with DMA in the C2S Direction**

Transaction Overhead	ACK Overhead	Comment
MRD for C2S Desc = 20/4096 = 0.625/128	*8/4096 = 0.25/128*	One descriptor fetch from C2S engine for 4 KB data (TRN-TX); 20B of TLP overhead and 8 bytes DLLP overhead
CPLD for C2S Desc = 20+32/4096 = 1.625/128	**8/4096=0.25/128**	Descriptor reception by C2S engine (TRN-RX). CPLD Header is 20 bytes, and the C2S Desc data is 32 bytes.
MWR for C2S buffer = 20/128	*8/128*	MPS = 128B; Buffer write from C2S engine (TRN-TX)
MWR for C2S Desc update = 20+12/4096 = 1/128	*8/4096 = 0.25/128*	Descriptor update from C2S engine (TRN-TX). MWR header is 20 bytes, and the C2S Desc update data is 12 bytes.

For every 128 bytes of data sent from the card to the system, the overhead on the upstream link (in bold) is 21.875 bytes.

$$\% \text{ Overhead} = 21.875 / (128 + 21.875) = 14.60\%$$

The throughput per PCIe lane is 2.5 Gb/s, but because of 8B/10B encoding, the throughput comes down to 2 Gb/s.

Maximum theoretical throughput per lane for
Receive = (100 – 14.60)/100 * 2 Gb/s = 1.70 Gb/s

Maximum theoretical throughput for a
x4 Gen2 or x8 Gen1 link for Receive = 13.6 Gb/s

The S2C DMA engine (which deals with data transmission, i.e., reading data from system memory) first does a buffer descriptor fetch. Using the buffer address in the descriptor, it issues Memory Read requests and receives data from system memory through completions. After the actual payload in transferred from the system, it sends a Memory Write to update the buffer descriptor. Table 4-2 shows the overhead incurred during data transfer in the S2C direction.

Table 4-2: **PCI Express Performance Estimation with DMA in the S2C Direction**

Transaction Overhead	ACK Overhead	Comment
MRD for S2C Desc=20/ 4096=0.625/128	*8/4096 = 0.25/ 128*	Descriptor fetch from S2C engine (TRN-TX)
CPLD for S2C Desc=20+32/ 4096=1.625/128	**8/4096 = 0.25/ 128**	Descriptor reception by S2C engine (TRN-RX). CPLD Header is 20 bytes and the S2C Desc data is 32 bytes.
MRD for S2C Buffer = 20/128	*8/128*	Buffer fetch from S2C engine (TRN-TX). MRRS=128B
CPLD for S2C buffer = 20/64 = 40/ 128	**8/64=16/128**	Buffer reception by S2C engine (TRN-RX). Because RCB=64B, 2 completions are received for every 128 byte read request
MWR for S2C Desc=20+4/ 4096=0.75/128	*8/4096=0.25/ 128*	Descriptor update from S2C engine (TRN-TX). MWR Header is 20 bytes and the S2C Desc update data is 12 bytes.

For every 128 bytes of data sent from system to card, the overhead on the downstream link (italicized) is 50.125 bytes.

% Overhead = 50.125/128 + 50.125 = 28.14%

The throughput per PCIe lane is 2.5 Gb/s, but because of 8B/10B encoding, the throughput comes down to 2 Gb/s.

Maximum theoretical throughput per lane for Transmit = (100 – 28.14)/100 * 2 = 1.43 Gb/s

Maximum theoretical throughput for a x4 Gen2 or x8 Gen1 link for Transmit = 11.44 Gb/s.

Because the TRD has two raw data paths, there are two C2S DMA engines and two S2C DMA engines. Each C2S and S2C engine should be able to theoretically operate at the 13.6 Gb/s and 11.44 Gb/s, respectively. If both data are enabled, the DMA splits the available bandwidth between the two C2S engines and two S2C engines.

The throughput numbers are theoretical and could go down further due other factors, such as:

- With an increase in lane width, PCIe credits are consumed at a faster rate, which could lead to throttling on the PCIe link reducing throughput.
- The transaction interface of PCIe is 64 bits wide. The data sent is not always 64-bit aligned, and this could cause some reduction in throughput.
- Changes in MPS, MRRS, RCB, and buffer descriptor size also have significant impact on the throughput. The MPS and MRRS values are negotiated between the host PC and all the endpoints plugged into the host PC. The RCB value is specific to the host PC.
- If bidirectional traffic is enabled, then overhead incurred further reduces throughput.
- Software overhead/latencies also contribute to reducing throughput.

Packetized Virtual FIFO Performance

For the Packetized Virtual FIFO, the theoretical maximum bandwidth to the DDR3 @ 800 MHz is 102.4 Gb/s.

Maximum IO Rate (double data rate) = 800 MHz * 2 = 1,600 Mb/s

Maximum Bandwidth = (Maximum IO rate) x (Number of IOs) = 1600 Mb/s x 64 = 102.4 Gb/s.

The data bandwidth to and from the DDR3 is a percentage of the total bandwidth on the 64-bit I/O lines. For the Virtual FIFO, data bandwidth efficiency is expected to be 80–90%.

An estimate of Memory Controller performance for burst size of 128 is calculated as follows:

With larger burst lengths, higher efficiency can be achieved.

With 64-bit port using a burst length of 128, a total of 8192 bits are transferred.

The number of bits transferred per cycle is:

64 (bit width) * 2 (double data rate) = 128 bits per cycle

The total cycles used for 8192 bits is:

8192/128 = 64 cycles per transfer

Assuming 10 cycles read to write overhead:

64/74 = 86% efficiency

Assuming 5% efficiency overhead for refresh, the total efficiency is about 81%.

Table 4-4 lists the estimated performance of the Packetized Virtual FIFO.

Table 4-3: **Projected Performance of Packetized Virtual FIFO with DDR3 Running @ 800 MHz**

Virtual FIFO	Throughput (Gb/s)	Comments
Total throughput	102.4*0.8= 81.92	80% efficiency
Total throughput	102.4*0.9= 92.16	90% efficiency

If we consider one S2C and one C2S DMA engine is enabled, the throughput required on the DDR3 interface is

[13.6 Gb/s (S2C) + 11.44 (C2S)] * 2 (Writes and Read, in and out of the DDR3) = 50.08 Gb/s

Because the maximum theoretical throughput numbers on the PCIe link with the DMA overhead is less than what the Virtual FIFO can handle, the limiting component in this Base TRD's system performance is the PCIe and DMA.

Measuring Performance

This section shows how performance is measured in the TRD.

It should be noted that PCI Express performance depends on factors like maximum payload size, maximum read request size, and read completion boundary, which are dependent on the systems used. With higher MPS values, performance improves as packet size increases.

Hardware provides the registers listed in Table 4-4 for software to aid performance measurement.

Table 4-4: **Performance Registers in Hardware**

Register	Description
DMA Completed Byte Count	DMA implements a completed byte count register per engine which counts the payload bytes delivered to the user on the streaming interface.
PCIe AXI TX Utilization	This register counts traffic on PCIe AXI TX interface including TLP headers for all transactions.
PCIe AXI RX Utilization	This register counts traffic on PCIe AXI RX interface including TLP headers for all transactions.
PCIe AXI TX Payload	This register counts payload for memory write transactions upstream which includes buffer write and descriptor updates.
PCIe AXI RX payload	This register counts payload for completion transactions downstream which includes descriptor or data buffer fetch completions.

These registers are updated once every second by hardware. Software can read them periodically at one second intervals to directly get the throughput.

The PCIe monitor registers can be read to understand PCIe transaction layer utilization. The DMA registers provide throughput measurement for actual payload transferred.

These registers give a good estimate of the TRD performance.

XILINX®

Chapter 5

Designing with the TRD Platform

The TRD platform acts as a framework for system designers to derive extensions or modify designs. This chapter outlines various ways for a user to evaluate, modify, and re-run the TRD. The suggested modifications are grouped under these categories:

- *Software-only* modifications are made by modifying software components only (drivers, demo parameters, and so on.). The design does not need to be re-implemented.
- *Design modifications* are changes made to design parameters. These changes modify hardware components only, i.e., parameters of individual IP components, custom logic, or the top level. The design must be re-implemented through the ISE® design tools.
- *Architectural changes* modify both hardware and software components. The design must be re-implemented through the ISE design tools. An example would be to add or replace IP blocks. The user needs to do some design work to ensure the new blocks can communicate with the existing interfaces in the framework. The user is also responsible to make sure that the new IP does not break the functionality of the existing framework. Relevant software changes might also be required to support the new IP.

All of these use models are fully supported by the framework, provided that the modifications do not require the supported IP components to operate outside the scope of their specified functionality.

This chapter provides examples to illustrate some of these use models. While some are simple modifications to the design, others involve replacement or addition of new IP. The new IP could come from Xilinx (and its partners) or from the customer's internal IP activities.

Software-Only Modifications

This section describes modifications to the platform done directly in the software driver. The same hardware design (BIT/MCS files) works. After any software modification, the code needs to be recompiled. The Linux driver compilation procedure is detailed in Appendix D, Compiling Linux Drivers.

Macro-Based Modifications

This section describes the modifications, which can be realized by compiling the software driver with various macro options, either in the Makefile or in the driver source code.

Descriptor Ring Size

The number of descriptors to be set up in the descriptor ring can be defined as a compile time option.

To change the size of the buffer descriptor ring used for DMA operations, modify `DMA_BD_CNT` in `linux_driver/xdma/xdma_base.c`.

Smaller rings can affect throughput adversely, which can be observed by running the performance tests.

A larger descriptor ring size uses additional memory, but improves performance, because more descriptors can be queued to hardware. Also see section Size of Block Data, page 68.

Note: The DMA_BD_CNT in the driver is set to 2999. Increasing this number might not improve performance.

Log Verbosity Level

To control the log verbosity level in Linux:

- Add DEBUG_VERBOSE in the Makefiles in the directories `linux_driver/xdma`, `linux_driver/xrawdata0`, and `linux_driver/xrawdata1` to cause the drivers to generate verbose logs.
- Add DEBUG_NORMAL in the Makefiles in the directories `linux_driver/xdma`, `linux_driver/xrawdata0`, and `linux_driver/ xrawdata1` to cause the drivers to generate informational logs.
- Remove both these macros from the Makefiles in the directories `linux_driver/xdma`, `linux_driver/xrawdata0`, and `linux_driver/ xrawdata1` to cause the drivers to only generate error logs.

Changes in the log verbosity are observed when examining the system logs. Increasing the logging level also causes a drop in throughput.

Driver Mode of Operation

The base DMA driver can be configured to run in either interrupt mode with MSI interrupts or in polled mode. Only one mode can be selected. To control the driver:

- Add TH_BH_ISR in the Makefile `linux_driver/xdma` to run the base DMA driver in interrupt mode.
- Remove the TH_BH_ISR macro to run the base DMA driver in polled mode.

Note: The interrupt mode has had only limited testing in hardware.

Size of Block Data

To modify the default amount of data being transmitted and received in the raw data drivers:

- Modify NUM_BUFS in `xrawdata0/sguser.c` or `xrawdata1/sguser.c` to change the number of buffers in the free pool available to the drivers. This modification changes the throughput observed with these drivers. If NUM_BUFS is modified, DMA_BD_CNT should also be modified to a value equal to NUM_BUFS-1.

Note: The available system memory must not be exceeded when these defaults are changed.

Software Driver Code Modifications

This section describes the modifications that can be made to software driver code to see a change in design behavior or performance.

The Block Data handler for raw data paths (`xrawdata0/sguser.c`, `xrawdata1/sguser.c`) can be modified as follows.

Data is written into DDR3 memory in a flat, unstructured manner, with known patterns. It is possible to create a packet format with some form of CRC, which can then be verified on the receive path. Packets are generated and verified within the driver and are not conveyed to or from any real user application as data. One suggested modification is to transfer this data between the driver and a user application. This requires significant changes in the driver entry points and in the driver's PutPkt() and GetPkt() routines. The data is transmitted (written) into DDR3 memory, and is looped back and received (read) from DDR3 memory.

Top-Level Design Modifications

This section describes changes to parameters in the top-level design file that can change the design behavior. Modifications to the software driver might be required based on the parameters being changed.

Hardware-Only Modifications

This section outlines the changes that require only hardware re-implementation.

Configuring the PCIe Link as x4 Lane at 2.5 Gb/s

The Kintex-7 FPGA integrated Endpoint block for PCI Express can be configured as x4 at a 2.5 Gb/s (Gen1) link rate instead of x4 at a 5 Gb/s (Gen2) link rate, taking a hit in performance. Selecting the option to configure the reference design with a x4 PCIe link at 2.5 Gb/s in the implement script automatically sets the parameters required for this change in the top-level design file. This option is enabled by this command:

```
$ source implement.sh x4 gen1 (for Linux)
```

```
$ implement.bat -lanemode x4gen1 (for Windows)
```

The implement script is available in the `k7_pcie_dma_ddr3_base/design/implement` directory of the TRD. After configuring the FPGA with the new bitstream, the user can rerun the TRD (refer to Reprogramming the Base TRD, page 30 in Chapter 2, Getting Startedto configure the FPGA). The results of the performance evaluation should be lower than the original version of the TRD.

Hardware and Software Modifications

This section outlines changes to be done to the top-level design file (`k7_pcie_dma_ddr3_base.v`) that also require software driver modifications.

PCIe Vendor and Device ID

PCIe vendor ID and device ID can be updated through these local parameters (localparam) in the top-level file:

- VENDOR_ID in the file `k7_pcie_dma_ddr3_base/design/source/k7_pcie_dma_ddr3_base.v` changes the vendor ID.
- DEVICE_ID in the file `k7_pcie_dma_ddr3_base /design/source/k7_pcie_dma_ddr3_base.v` changes the device ID.

The software then requires a corresponding change.

- Change the PCI_VENDOR_ID_DMA macro in `k7_pcie_dma_ddr3_base/linux_driver/xdma/xdma_base.c`.
- Change the PCI_DEVICE_ID_DMA macro in `k7_pcie_dma_ddr3_base/linux_driver/xdma/xdma_base.c`.

Refer to Appendix D, Compiling Linux Drivers for how to recompile drivers and install them. Refer to Appendix C, Directory Structure to navigate to the required files.

Architectural Modifications

This section describes architecture level changes to the functionality of the platform. These changes include adding or deleting IP with similar interfaces used in the framework.

Aurora IP Integration

The LogiCORE IP Aurora 8B/10B core implements the Aurora 8B/10B protocol using the high-speed Kintex-7 FPGA GTX transceivers. The core is a scalable, lightweight link layer protocol for high-speed serial communication. It is used to transfer data between two devices using transceivers. It provides an AXI4-Stream compliant user interface.

A 4-lane Aurora design with 2-byte user interface data width presents a 64-bit AXI4-Stream user interface, which matches the Raw Packet Data module's interface within the framework. Hence, a customer can accelerate the task of creating a PCIe-to-Aurora bridge design through these high-level steps:

1. Generate a four-lane (3.125 Gb/s line rate per lane) and two-byte Aurora 8B/10B LogiCORE IP from the CORE Generator tool. Remove the raw data block instance.
2. Remove the Raw Packet Data block and insert the Aurora LogiCORE IP into the framework (see Figure 5-1).
3. Add an MMCM block to generate a 156.25 MHz clock, or use an external clock source, to drive a 156.25 MHz clock into the Aurora LogiCORE IP.
4. Simulate the design with the out-of-box simulation framework with appropriate modifications to include the Aurora files.
5. Implement the design and run the design with Aurora in loopback mode with minimal changes to the implementation flow.

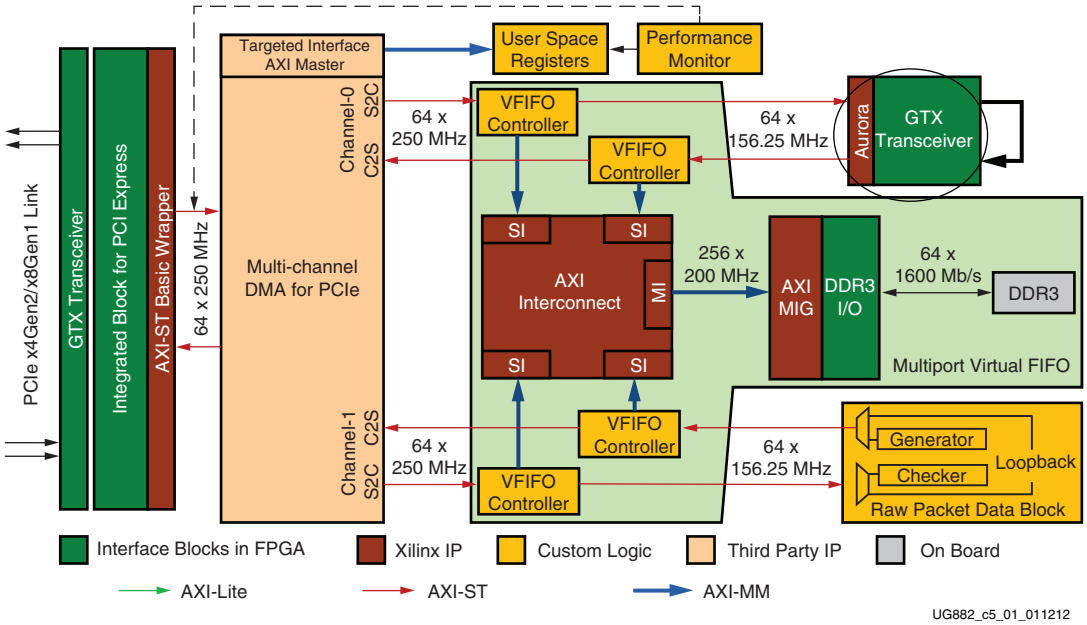


Figure 5-1: **Integrating Aurora**

Aurora IP does not support throttling in the receive direction, because the core has no internal buffers. The Multiport Virtual FIFO in the data path allows the user to drain packets at the line rate. The Native Flow Control feature of Aurora can also be used to manage flow control. As per the Aurora protocol, the round trip delay through the Aurora interfaces between the NFC request and the first pause arriving at the originating channel partner must not exceed 256 symbol times.

For 4 lanes, time taken to transmit 4 symbols with each lane running at 3.125 Gb/s
40 bits/4 lanes x 1/3.125 Gb/s = 3.2 ns (1 symbol = 10 bits because of 8B/10B encoding scheme).

For 256 symbols, time taken to transmit is 256/4 x 3.2 = 205 ns.

For a 156.25 MHz clock (8 ns period), this is 26 clock cycles (the worst case delay), amounting to a FIFO depth of 26, which is required to hold data received on the Aurora RX interface after an NFC request to pause data is initiated. The user must appropriately configure the preview FIFO thresholds for full and empty in Multiport Virtual Packet FIFO considering this value to prevent overflows.

The Raw Packet Data driver can be reused for Aurora with some modifications. The data generated by the block handler for Raw Packet Data could now drive traffic over Aurora. The Aurora serial interface needs to be looped back externally or connected to another Aurora link partner.

The maximum theoretical throughput that can be achieved on the Aurora path is 10 Gb/s (64 bit * 156.25 MHz). Refer to UG766, *LogiCORE IP Aurora 8B/10B v7.1 User Guide* (AXI) for throughput efficiency [Ref 3].

XILINX®

Appendix A

Resource Utilization

Table A-1 and Table A-2 list resource utilization obtained from the map report during the implementation phase. The XC7K325T-2FFG900C is the target FPGA.

Note: The reported utilization numbers are obtained with the specific options set for synthesis and implementation of the design. Refer to the implement script to find the options that are set. A change in the default options results in a change in the utilization numbers.

Table A-1: **Resources for the TRD with the PCIe Link Configured as x4 at a 5 Gb/s Link Rate**

Resource	Utilization	Total Available	Percentage Utilization (%)
Slice registers	55,797	407,600	13%
Slice LUTs	47,551	203,800	23%
Bonded IOB	126	500	25%
RAMB36E1	69	445	15%
RAMB18E1	4	890	1%
BUFG/BUFGCNTRL	8	32	25%
MMCM_ADV	1	10	10%
GTXE2_CHANNELS	4	16	25%
GTXE2_COMMONS	1	4	25%
PCIE_2_1	1	1	100%

Table A-2: **Resources for the TRD with the PCIe Link Configured as x8 at a 2.5 Gb/s Link Rate**

Resource	Utilization	Total Available	Percentage Utilization (%)
Slice registers	56,793	407,600	13%
Slice LUTs	47,668	203,800	23%
Bonded IOB	126	500	25%
RAMB36E1	69	445	15%
RAMB18E1	4	890	1%
BUFG/BUFGCNTRL	8	32	25%
MMCM_ADV	1	10	10%

Table A-2: **Resources for the TRD with the PCIe Link Configured as x8 at a 2.5 Gb/s Link Rate *(Cont'd)***

Resource	Utilization	Total Available	Percentage Utilization (%)
GTXE2_CHANNELS	8	16	50%
GTXE2_COMMONS	2	4	50%
PCIE_2_1	1	1	100%

XILINX®

Appendix B

Register Description

This appendix describes registers most commonly accessed by the software driver.

The registers implemented in hardware are mapped to base address register (BAR0) in the PCIe integrated Endpoint block.

Table B-1 shows the mapping of multiple DMA channel registers across the BAR.

Table B-1: **DMA Channel Register Address**

DMA Channel	Offset from BAR0
Channel-0 S2C	0x0
Channel-1 S2C	0x100
Channel-0 C2S	0x2000
Channel-1 C2S	0x2100

Registers in DMA for interrupt handling are grouped under a category called common registers, which are at an offset of 0x4000 from BAR0.

Figure B-1 shows the layout of registers.

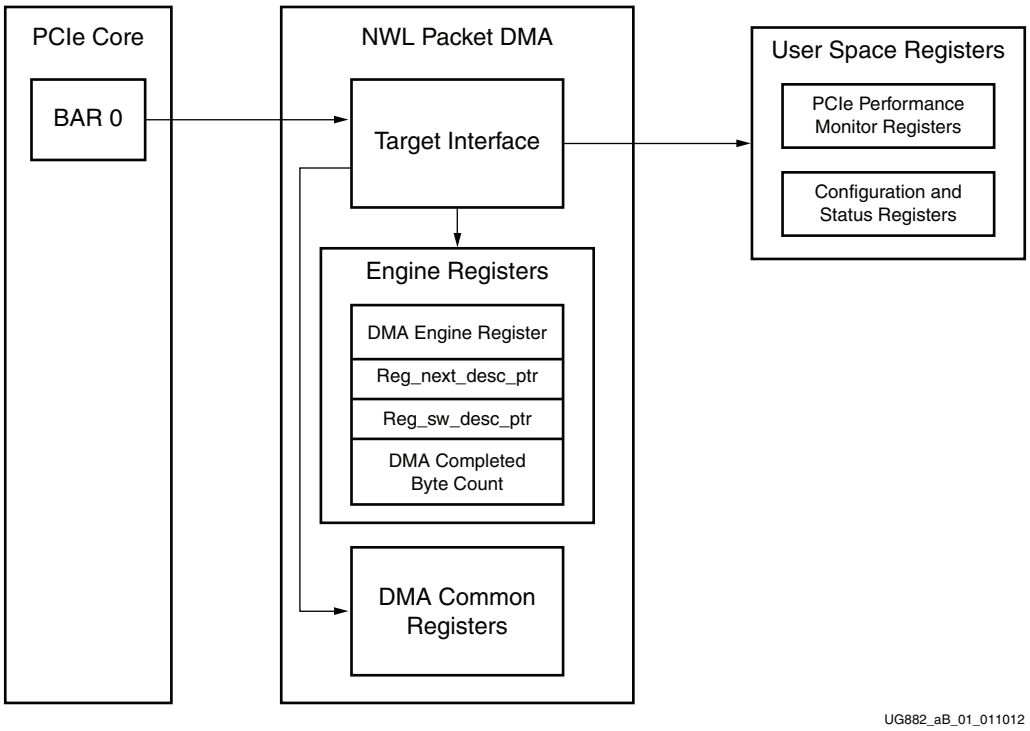


Figure B-1: **Register Map**

The user logic registers are mapped as shown in Table B-2.

Table B-2: **User Register Address Offsets**

User Logic Register Group	Range (Offset from BAR0)
PCIe performance Registers Design version and status Registers	0x9000–0x90FF
User Application 0 Registers	0x9100–0x91FF
User Application 1 Registers	0x9200–0x92FF
Packetized VFIFO Registers	0x9300–0x93FF

DMA Registers

This section describes certain prominent DMA registers used very frequently by the software driver. For a detailed description of all registers available, refer to the *Northwest Logic DMA Back-End Core User Guide* and *Northwest Logic DMA AXI DMA Back-End Core User Guide*, available in the `k7_pcie_dma_ddr3_base/design/ipcores/dma/doc` directory.

Channel Specific Registers

The registers described in this section are present in all channels. The address of the register is channel address offset from BAR0 (refer to Table B-1) + the register offset.

Engine Control (0x0004)

Table B-3: **DMA Engine Control Register**

Bit	Field	Mode	Default Value	Description
0	Interrupt Enable	RW	0	Enables interrupt generation
1	Interrupt Active	RW1C	0	Interrupt active is set whenever an interrupt event occurs. Write '1' to clear.
2	Descriptor Complete	RW1C	0	Interrupt active was asserted due to completion of descriptor. This is asserted when a descriptor with interrupt on completion bit set is seen.
3	Descriptor Alignment Error	RW1C	0	This causes interrupt when a descriptor address is unaligned, and that DMA operation is aborted.
4	Descriptor Fetch Error	RW1C	0	This causes interrupt when a descriptor fetch errors, i.e., completion status is not successful.
5	SW_Abort_Error	RW1C	0	This is asserted when a DMA operation is aborted by software.
8	DMA Enable	RW	0	Enables the DMA engine. After enabled, the engine compares the next descriptor pointer and software descriptor pointer to begin execution.
10	DMA_Running	RO	0	Indicates DMA in operation.
11	DMA_Waiting	RO	0	Indicates DMA waiting on software to provide more descriptors.
14	DMA_Reset_Request	RW	0	Issues a request to user logic connected to DMA to abort outstanding operation and prepare for reset. This is cleared when user acknowledges the reset request.
15	DMA_Reset	RW	0	Assertion of this bit resets the DMA engine and issues a reset to user logic.

Next Descriptor Pointer (0x0008)

Table B-4: **DMA Next Descriptor Pointer Register**

Bit	Field	Mode	Default Value	Description
[31:5]	Reg_Next_Desc_Ptr	RW	0	Next Descriptor Pointer is writable when DMA is not enabled. It is read only when DMA is enabled. This should be written to initialize the start of a new DMA chain.
[4:0]	Reserved	RO	5'b00000	Required for 32-byte alignment

Software Descriptor Pointer (0x000C)

Table B-5: **DMA Software Descriptor Pointer Register**

Bit	Field	Mode	Default Value	Description
[31:5]	Reg_SW_Desc_Ptr	RW	0	Software Descriptor Pointer is the location of the first descriptor in a chain that is still owned by the software
[4:0]	Reserved	RO	5'b00000	Required for 32-byte alignment

Completed Byte Count (0x001C)

Table B-6: **DMA Completed Byte Count Register**

Bit	Field	Mode	Default Value	Description
[31:2]	DMA_Completed_Byte_Count	RO	0	Completed byte count records the number of bytes that transferred in the previous one second. This has a resolution of 4 bytes.
[1:0]	Sample Count	RO	0	This sample count increments every time a sample is taken at 1 second interval

Common Registers

The registers described in this section are common to all engines. The register addresses are located at the given offsets from BAR0.

Common Control & Status (0x4000)

Table B-7: **DMA Common Control & Status Register**

Bit	Field	Mode	Default Value	Description
0	Global Interrupt Enable	RW	0	Global DMA Interrupt Enable This bit globally enables or disables interrupts for all DMA engines
1	Interrupt Active	RO	0	Reflects the state of the DMA interrupt hardware output considering the state is global interrupt enable
2	Interrupt Pending	RO	0	Reflects the state of the DMA interrupt output *without* considering the state of global interrupt enable
3	Interrupt Mode	RO	0	0 - MSI mode 1 - Legacy interrupt mode
4	User Interrupt Enable	RW	0	Enables generation of user interrupts
5	User Interrupt Active	RW1C	0	Indicates active user interrupt
23:16	S2C Interrupt Status	RO	0	Bit[i] indicates interrupt status of S2C DMA engine[i] If S2C engine is not present, then this bit is read as zero.
31:24	C2S Interrupt Status	RO	0	Bit[i] indicates interrupt status of C2S DMA engine[i] If C2S engine is not present, then this bit is read as zero.

User Space Registers

This section describes the custom registers implemented in the user space. All registers are 32-bit wide. Register bits positions are to be read from 31 to 0 from left to right. All bits undefined in this section are reserved, and return zero on read. All registers would return default values on reset. Address holes return a value of zero on being read.

All registers are mapped to BAR0 and relevant offsets are provided.

Design Version and Status Registers

Design Version (0x9000)

Table B-8: **Design Version Register**

Bit Position	Mode	Default Value	Description
3:0	RO	0001	Subversion number: 0000: Non-AXI design 0001: AXI design
11:4	RO	0001_0000	Version of the design (e.g., for v1.0, the version is 0001_0000)

Table B-8: **Design Version Register *(Cont'd)***

Bit Position	Mode	Default Value	Description
15:12	RO	0000	Reserved
27:16	RO	0001_0000_1000	NWL DMA Version
31:28	RO	0001	Device— 0000: Artix-7 FPGA 0001: Kintex-7 FPGA 0010: Virtex-7 FPGA 0011: Virtex-7 XT FPGA

Design Status (0x9008)

Table B-9: **Design Status Register**

Bit Position	Mode	Default Value	Description
0	RO	0	DDR3 Memory Controller initialization/calibration done (design operational status from hardware)
1	RW	1	axi_ic_mig_shim_rst_n - Resets the AXI Interconnect IP and MIG AXI Interface. When software writes to this register it self clears after 9 clock cycles
5:2	RO	f	ddr3_fifo_empty - indicates the DDR3 FIFO and the preview FIFOs per port are empty.

Transmit Utilization Byte Count (0x900C)

Table B-10: **PCIe Performance Monitor - Transmit Utilization Byte Count Register**

Bit Position	Mode	Default Value	Description
1:0	RO	00	Sample count - increments every second
31:2	RO	0	**Transmit utilization byte count**—This field contains the interface utilization count for active beats on PCIe AXI4-Stream interface for transmit. It has a resolution of 4 bytes.

Receive Utilization Byte Count (0x9010)

Table B-11: **Performance Monitor - Receive Utilization Byte Count Register**

Bit Position	Mode	Default Value	Description
1:0	RO	00	Sample count - increments every second
31:2	RO	0	**Receive utilization payload byte count**—This field contains the interface utilization count for active beats on PCIe AXI4-Stream interface for receive. It has a resolution of 4 bytes.

Upstream Memory Write Byte Count (0x9014)

Table B-12: **PCIe Performance Monitor - Upstream Memory Write Byte Count Register**

Bit Position	Mode	Default Value	Description
1:0	RO	00	Sample count—Increments every second
31:2	RO	0	**Upstream memory write byte count**—This field contains the payload byte count for upstream PCIe memory write transactions. It has a resolution of 4 bytes.

Downstream Completion Byte Count (0x9018)

Table B-13: **PCIe Performance Monitor - Downstream Completion Byte**

Bit Position	Mode	Default Value	Description
1:0	RO	00	Sample count—Increments every second
31:2	RO	0	**Downstream completion byte count**—This field contains the payload byte count for downstream PCIe completion with data transactions. It has a resolution of 4 bytes.

Initial Completion Data Credits for Downstream Port (0x901C)

Table B-14: **PCIe Performance Monitor - Initial Completion Data Credits Register**

Bit Position	Mode	Default Value	Description
11:0	RO	00	**INIT_FC_CD** captures initial flow control credits for completion data for host system

Initial Completion Header Credits for Downstream Port (0x9020)

Table B-15: **PCIe Performance Monitor - Initial Completion Header Credits Register**

Bit Position	Mode	Default Value	Description
7:0	RO	00	**INIT_FC_CH** captures initial flow control credits for completion header for host system

PCIe Credits Status - Initial Non Posted Data Credits for Downstream Port (0x9024)

Table B-16: **PCIe Performance Monitor - Initial NPD Credits Register**

Bit Position	Mode	Default Value	Description
11:0	RO	00	**INIT_FC_NPD** captures initial flow control credits for non-posted data for host system

PCIe Credits Status - Initial Non Posted Header Credits for Downstream Port (0x9028)

Table B-17: **PCIe Performance Monitor - Initial NPH Credits Register**

Bit Position	Mode	Default Value	Description
7:0	RO	00	**INIT_FC_NPH** captures initial flow control credits for non-posted header for host system

PCIe Credits Status - Initial Posted Data Credits for Downstream Port (0x902C)

Table B-18: **PCIe Performance Monitor - Initial PD Credits Register**

Bit Position	Mode	Default Value	Description
11:0	RO	00	**INIT_FC_PD** captures initial flow control credits for posted data for host system

PCIe Credits Status - Initial Posted Header Credits for Downstream Port (0x9030)

Table B-19: **PCIe Performance Monitor - Initial PH Credits Register**

Bit Position	Mode	Default Value	Description
7:0	RO	00	**INIT_FC_PH** captures initial flow control credits for posted header for host system

Generator/Checker/Loopback Registers for User APP 0

This section describes registers that can be configured on the raw data Path0 in the user application space.

Enable Generator (0x9100)

Table B-20: **User App 0 - Enable Generator Register**

Bit Position	Mode	Default Value	Description
0	RW	0	Enable traffic generator - C2S0

Packet Length (0x9104)

Table B-21: **User App 0 - Packet Length Register**

Bit Position	Mode	Default Value	Description
15:0	RW	16'd4096	**Packet Length** to be generated. Maximum supported is 64KB size packets. (C2S0)

Enable Loopback/Checker (0x9108)

Table B-22: **User App 0 - Enable Loopback/Checker Register**

Bit Position	Mode	Default Value	Description
0	RW	0	Enable traffic checker - S2C0
1	RW	0	Enable Loopback - S2C0 <—> C2S0

Checker Status (0x910C)

Table B-23: **User App 0 - Checker Status Register**

Bit Position	Mode	Default Value	Description
0	RW1C	0	**Checker error**—Indicates data mismatch when set (S2C0)

Generator/Checker/Loopback Registers for User APP 1

This section describes registers that can be configured on the raw data Path1 in the user application space.

Enable Generator (0x9200)

Table B-24: **User App 1 - Enable Generator Register**

Bit Position	Mode	Default Value	Description
0	RW	0	Enable traffic generator - C2S1

Packet Length (0x9204)

Table B-25: **User App 1 - Packet Length Register**

Bit Position	Mode	Default Value	Description
15:0	RW	16'd4096	Packet Length to be generated. Maximum supported is 64KB size packets. (C2S1)

Enable Loopback/Checker (0x9208)

Table B-26: **User App 1 - Enable Loopback/Checker Register**

Bit Position	Mode	Default Value	Description
0	RW	0	Enable traffic checker - S2C1
1	RW	0	Enable Loopback - S2C1 <—> C2S1

Checker Status (0x920C)

Table B-27: **User App 1 - Checker Status Register**

Bit Position	Mode	Default Value	Description
0	RW1C	0	**Checker error**—Indicates data mismatch when set (S2C1)

Memory Controller Registers

Registers in Table B-28 through Table B-31 are repeated for each of the four ports of the Virtual Packet FIFO. The addresses given for each register are in the ascending order (Port 1, Port 2, Port 3, Port 4). Refer to Multiport Virtual Packet FIFO, page 43, for details.

Start Address (0x9300, 0x9310, 0x9320, 0x9330)

Table B-28: **Start Address Register**

Bit Position	Mode	Default Value	Description
31:0	RW	0	Start address of the DDR3 memory for that channel

End Address (0x9304, 0x9314, 0x9324, 0x9334)

Table B-29: **End Address Register**

Bit Position	Mode	Default Value	Description
31:0	RW	32'h0100_0000	**End address** of the DDR3 memory for that channel. The implemented FIFO logic wraps around at this address.

Write Burst Size (0x9308, 0x9318, 0x9328, 0x9338)

Table B-30: **Write Burst Size Register**

Bit Position	Mode	Default Value	Description
8:0	RW	9'd256	**Write Burst Size** for AXI-MM write transactions issued to AXI interconnect SI slot

Read Burst Size (0x930C, 0x931C, 0x932C, 0x933C)

Table B-31: **Read Burst Size Register**

Bit Position	Mode	Default Value	Description
8:0	RW	9'd256	**Read Burst Size** for AXI-MM read transactions issued to AXI interconnect SI slot

Appendix C

Directory Structure

This appendix describes the directory structure and explains the organization of various files and folders.

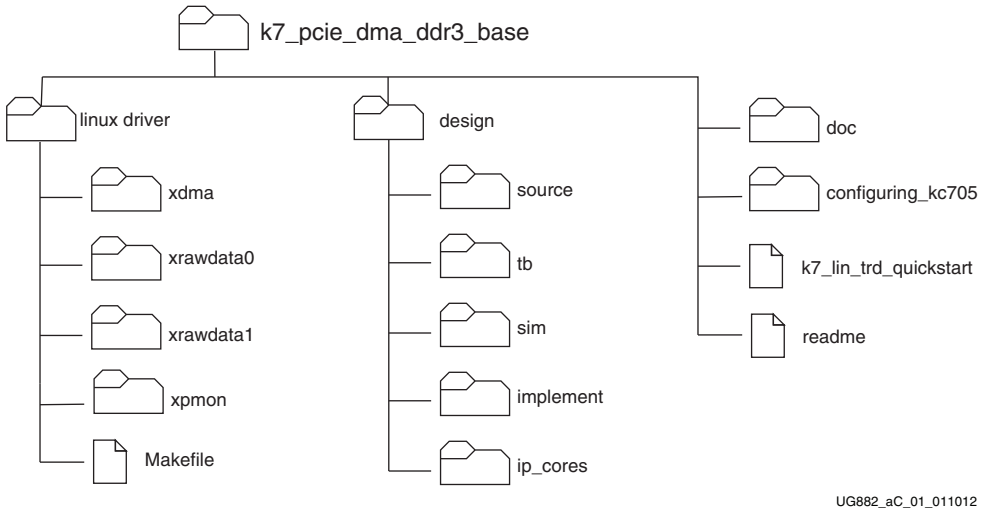


Figure C-1: **Directory Structure**

The `design` folder contains all the hardware design deliverables:

- The `source` subfolder contains source code deliverable files.
- The `tb` subfolder contains test bench related files for simulation.
- The `sim` subfolder contains simulation scripts for supported simulators for both Microsoft Windows and Linux operating systems.
- The `implement` subfolder contains implementation scripts for the design for both Microsoft Windows and Linux operating systems.
- The `ip_cores` subfolder contains IP cores required for this design and the DMA design files.

The `doc` folder contains the TRD documentation (the user guide and Doxygen generated html for software driver details).

The `configuring_kc705` folder contains programming files and scripts to configure the KC705 board.

These files are in the top-level directory:

- The `readme` file provides details on the use of simulation and implementation scripts.

- The `k7_trd_lin_quickstart` script is used to build and insert driver and GUI modules, invoke the GUI, and remove the driver modules when the user closes the GUI window.

XILINX®

Appendix D

Compiling Linux Drivers

This section provides steps on Linux driver compilation. The Linux driver source code for the design is available under the directory `k7_pcie_dma_ddr3_base/linux_driver`.

If the software is modified, rerun `k7_trd_lin_quickstart` to recompile and load the driver. This also launches the Application GUI. (Refer to Driver Installation, page 17.)

The user can also run the individual steps detailed in this appendix to get a better understanding of the driver.

1. Compile the Linux drivers and insert the kernel modules.

   Open a terminal window. Navigate to the `k7_pcie_dma_ddr3_base/linux_driver` folder. To compile and insert the driver, follow these steps at the command line in the terminal in the `linux_driver` folder:

   a. To clean the area, type:
      `$ make clean`

   b. To compile the files and build the kernel objects, type:
      `$ make`

   c. To insert the kernel object files, type:
      `$ make insert`

2. To check the status of the device drivers, at the terminal command line, type:
   `$ lsmod | more`

   Look for the drives that are loaded. The xrawdata0 and xrawdata1 modules depend on the base xdma driver. The **`lsmod`** command displays *Used by* on the xdma_k7 entry as shown in Figure D-1.

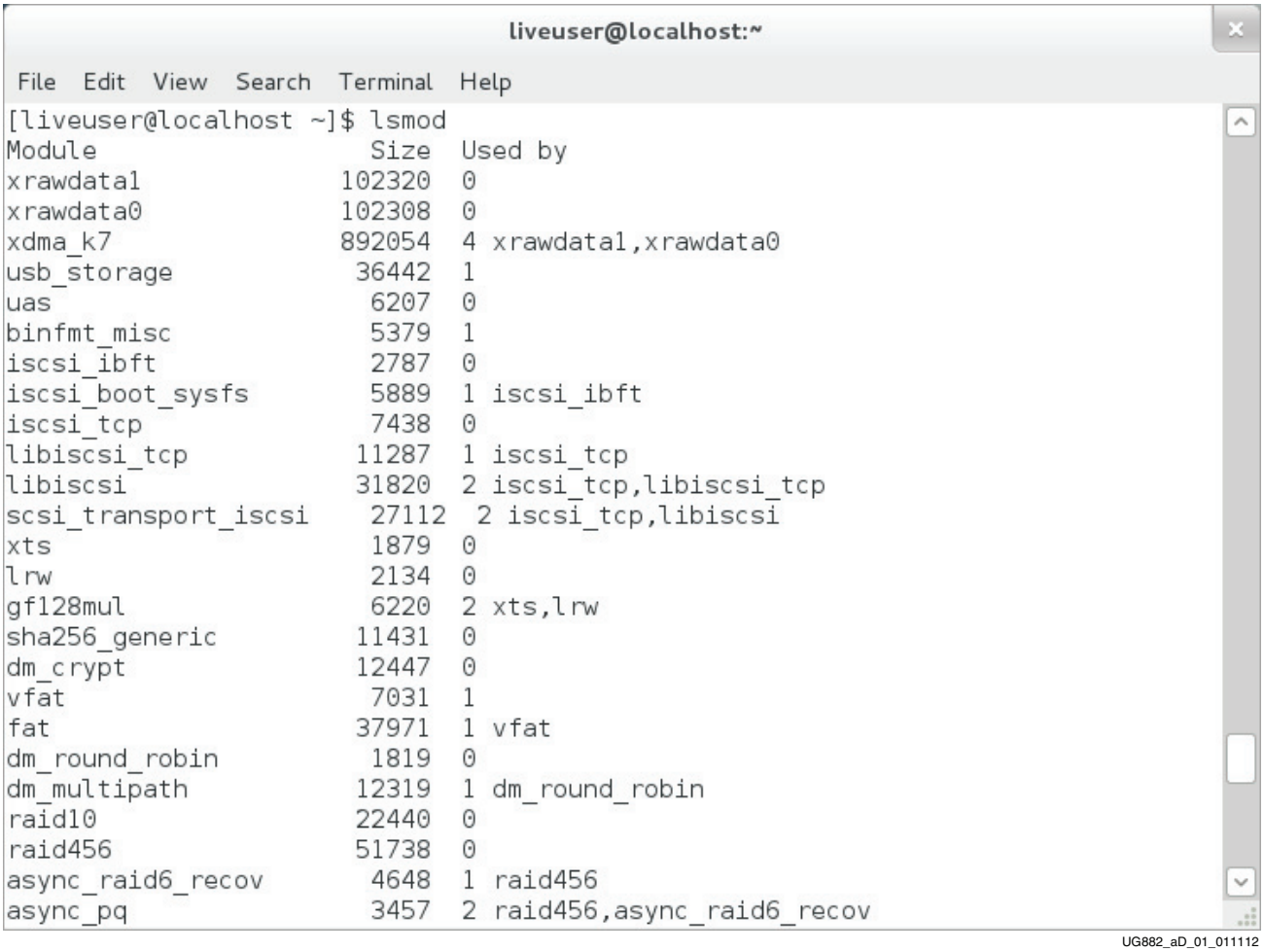


Figure D-1: **Loaded Drivers xdma_k7, xrawdata0, xrawdata1**

3. GUI compilation: Steps are provided for compiling and invoking the GUI.

   To compile and invoke the GUI, navigate to the `k7_pcie_dma_ddr3_base/linux_driver/xpmon` folder and follow these steps:

   a. To clean the area, type:
      `$ make clean`

   b. To compile the files, type:
      `$ make`

   c. To invoke the GUI, type:
      `$. /xpmon`

   To run the application GUI, go to Using the Application GUI, page 19.

4. Remove the device drivers. Steps are provided for unloading the driver.

   To unload the driver modules, navigate to the `k7_pcie_dma_ddr3_base/linux_driver` folder and execute this command at the command line in the terminal:
   `$ make remove`

This step takes a few seconds to free the allocated buffers and remove the three device drivers. To check that the drivers have been successfully removed, use the **`lsmod`** command in the terminal window again (see Figure D-2).

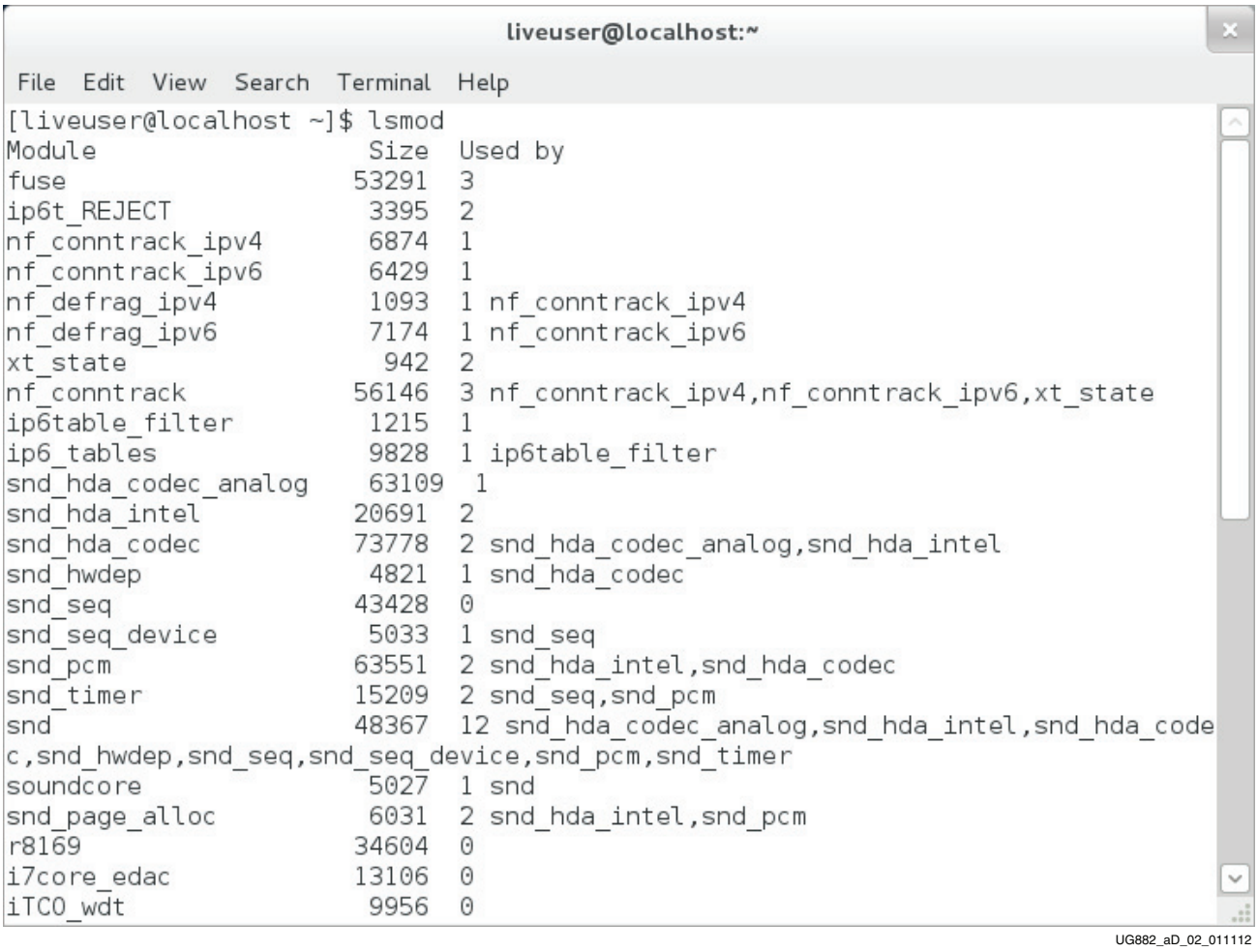


Figure D-2: **Removed Drivers xdma_k7, xrawdata0, and xrawdata1**

XILINX®

Appendix E

Additional Resources

Xilinx Resources

To search the Answer database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

References

These documents provide supplemental material useful with this user guide.

1. UG882, *Kintex-7 FPGA Base Targeted Reference Design User Guide* (this guide)
2. UG798, *Xilinx Design Tools: Installation and Licensing Guide*
3. UG766, *LogiCORE IP Aurora 8B/10B v7.1 User Guide*
4. UG477, *7 Series FPGAs Integrated Block for PCI Express User Guide*
5. UG626, *Synthesis and Simulation Design Guide*
6. WP350, *Understanding Performance of PCI Express Systems*
7. UG476, *7 Series FPGAs GTX Transceivers User Guide*
8. UG810, *KC705 Evaluation Board for the Kintex-7 FPGA User Guide*
9. UG586, *7 Series FPGAs Memory Interface Solutions User Guide*
10. UG883, *Kintex-7 FPGA Base Targeted Reference Design Getting Started Guide*
11. AXI Interconnect IP:
    http://www.xilinx.com/products/intellectual-property/axi_interconnect.htm

Additional Useful Sites for Boards and Kits

12. Design advisories by software release for Kintex-7 FPGA KC705 Evaluation Kit
    http://www.xilinx.com/support/#nav=sd-nav-link-179661&tab=tab-bk
13. Updated information about the Kintex-7 FPGA Base TRD and Kintex-7 FPGA KC705 Evaluation Kit
    www.xilinx.com/kc705
14. KC705 support page
    http://www.xilinx.com/products/boards-and-kits/EK-K7-KC705-G.htm

Third Party Resources

Documents associated with other software, tool, and IP used by the base TRD are available at these vendor websites:

15. Northwest Logic DMA back-end core:
    http://www.nwlogic.com/packetdma/
16. Fedora project:
    http://fedoraproject.org

    Fedora is a Linux-based operating system used in the development of this TRD.
17. The GTK+ project API documentation:
    http://www.gtk.org/documentation.php
    GTK+ is a toolkit for creating graphical user interfaces (GUI).