Kintex-7 FPGA Connectivity Targeted Reference Design

User Guide

Vivado Design Suite 2014.3

UG927 (v7.0) December 18, 2014

XILINX®

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 Xilinx’s limited warranty, please refer to Xilinx’s Terms of Sale which can be viewed at www.xilinx.com/legal.htm#tos; 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 such critical applications, please refer to Xilinx’s Terms of Sale which can be viewed at www.xilinx.com/legal.htm#tos.

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.

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
07/18/2012	1.0	Initial Xilinx release.
11/06/2012	1.1	Added second sentence to third paragraph under Connectivity Targeted Reference Design. Updated Figure 1-1, Figure 1-2, and Figure 1-3. Changed "FIFO" to "Virtual FIFO" in third paragraph under Raw Ethernet Performance Mode. Updated resource utilization usage data in Table 1-1. Updated Figure 2-28. Deleted first two rows in Table 2-2. Deleted "Multiport Virtual Packet FIFO" section from Chapter 2, Getting Started. Updated Figure 3-10. Changed "Multiport Packet FIFO" to "AXI Virtual FIFO" in Table 3-5. Changed "DDR3 Virtual FIFO" to "AXI Virtual FIFO". Changed "virtual FIFO controller" to "AXI Virtual FIFO controller" in last sentence under AXI Virtual FIFO. Changed "Virtual FIFO" to "AXI Virtual FIFO" in section title Packet Generator/Checker Behind AXI Virtual FIFO. Updated Figure 5-1. Updated Figure A-1. Deleted "Packetized VFIFO registers" row from Table A-2. Deleted "Memory Controller Registers" section from Appendix A, Register Description.
11/07/2012	1.2	Added *Implementing the Design Using the Vivado Flow* to Chapter 2, Getting Started.
01/09/2013	2.0	Replaced references to USB stick with link to design files under Test Setup Requirements and Installing the Linux Device Drivers. Changed "ISE Design Suite Logic Edition v14.1" to "Vivado® Design Suite" under Test Setup Requirements and Rebuilding the Design. Added note preceding Hardware Demonstration Setup. Deleted Figure 2-18 "MIG Core Operation", "Implementing the Design Using Command Line Options" and "Implementing the Design Using the PlanAhead Design Tool" sections from Chapter 2, Getting Started. Added MCS file generation to *Implementing the Design Using the Vivado Flow*. Changed "8,192" to "a configurable number of" under Initialization Phase. Changed "Completed Byte Count (0x001D)" to "Completed Byte Count (0x001C)" above Table A-6. Replaced ISE Design Suite user guide reference with Vivado Design Suite user guide references under Xilinx Resources. Added reference to Faster Technology FM-S14 User Manual under Further Resources.
04/17/2013	3.0	Updated title page for Vivado Design Suite 2013.1. Changed "32-bit" to "32- and 64-bit" in last paragraph under Connectivity Targeted Reference Design. Added Notes 2 and 3 following Figure 1-1. Changed "ModelSim simulator v10.1a" to "QuestaSim simulator v10.1b" throughout document. Updated the cores and netlist directories and added the DMA netlist and AXILite Interconnect IP directories under Rebuilding the Design. Deleted last paragraph and "Generating the MIG IP Core through CORE generator" subsection under Rebuilding the Design. Updated "UG477" to "PG054", "UG773" to "PG072", and "UG692" to "PG068" user guide references throughout document. Revised Simulating the Design subsection. Added last sentence under Overview. Revised paragraph following Table 2-3. Added 64-Bit Driver Compilation subsection. Updated references in Appendix G, Additional Resources.
07/10/2013	4.0	Updated title page for Vivado Design Suite 2013.2. Updated contents of Table A-20. Replaced references to the UCF file with references to XDC file in *Different Quad Selection for 10GBASE-R IP*. Added note in Resource Utilization. Updated references in Appendix G, Additional Resources.

Date	Version	Revision
11/20/2013	5.0	Updated title page for Vivado Design Suite 2013.3 and changed references to earlier Vivado Design Suite versions to 2013.3. Added Installing the Windows Device Driver, page 22 through page 29. Added note on page 37. Revised the file name paths and instructions under *Implementing the Design Using the Vivado Flow* and Simulating the Design, page 42. Strengthened the description in the power connection caution note on page 16. Added Windows Device Driver and Application, page 74 through page 81. Revised the file name paths and instructions under 64-Bit Driver Compilation, page 98, Jumbo Frames, page 98, and Driver Queue Depth, page 99. Revised Directory Structure shown in Figure B-1, page 113 and path names under File Descriptions, page 114. Revised all links and references in Appendix G, Additional Resources and revised links to web pages and documents throughout document to conform to latest linking style convention.
12/18/2013	5.0.1	Tech Pubs edit.
05/14/2014	6.0	Updated for Vivado Design Suite 2014.1. Chapter 1: Usage numbers changed in Resource Utilization, page 11. Chapter 2: Version 10.1b was dropped from Simulation Requirements, page 13. Revised the path to the LogiCORE™ IP blocks in Rebuilding the Design, page 39. Added section Implementing the Design Using Vivado IP Integrator (IPI) Flow, page 39. Figure 2-10 InstallShield Wizard First Window was replaced. Updated Reprogramming the KC705 Board. Commands and paths changed in Simulating the Design, page 42. The USE_DIFF_QUAD macro was removed from Table 2-2 Macro Description for Design Change. Chapter 5: The *Design Top-Level Only Modifications* section was removed. Appendix B: In the `sources` folder, changed *ip_catalog/ip_cores* to *ip_catalog/ip_cores/ip_package* in text and in Figure B-1. Updated the contents of the `doc` folder. Appendix C: Corrected the path after the procedure in Compiling Traffic Generator Applications, page 115. Removed redundant Figure C-2 Private LAN Setup graphic.
12/18/2014	7.0	Updated for Vivado Design Suite 2014.3. This document is for Vivado Design Suite IP Integrator (IPI) designs only, so HDL flow information is removed or updated. *Implementing the Design Using the Vivado Flow* is removed. Updated the Resource Utilization table. Updated the Rebuilding the Design section. Commands in Simulating the Design changed. Updated the Windows Device Driver and Application and Internal Buffer Transfers sections. Revised Figure 2-17 and Figure 2-19. Updated the directory structure in Figure B-1 with IPI information.

Table of Contents

XILINX®

Chapter 1

Introduction

This chapter introduces the Kintex®-7 Connectivity Targeted Reference Design (TRD). summarizes its modes of operation, and identifies the features provided.

Connectivity Targeted Reference Design

Figure 1-1 depicts the block level overview of the Kintex-7 Connectivity TRD which delivers up to 20 Gb/s of performance per direction.

The design is a dual Network Interface Card (NIC) with a GEN2 x8 PCIe endpoint, a multi-channel packet DMA from Northwest Logic, DDR3 memory for buffering, 10G Ethernet MAC, and 10GBASE-R standard compatible physical layer interface. The PCIe-DMA together is responsible for movement of data between a PC system and FPGA (S2C implies data movement from PC system to FPGA and C2S implies data movement from FPGA to PC system).

DDR3 SDRAM (64-bit, 1,600 Mb/s or 800 MHz) is used for packet buffering – a virtual FIFO layer facilitates the use of DDR3 as multiple FIFOs. The virtual FIFO layer is built using the AXI Stream interconnect and AXI Virtual FIFO controller CoreGEN IPs

Dual NIC application is built over this by use of Ten Gigabit Ethernet MAC and Ten Gigabit PCS/PMA (10GBASE-R PHY) IPs. The 10G MAC connects to the 10G BASE-R PHY over 64-bit, SDR XGMII parallel interface. Additionally, the design provides power monitoring capability based on a PicoBlaze™ controller engine.

For software, the design provides 32- and 64-bit Linux drivers for all modes of operation listed below and a graphical user interface (GUI) which controls the tests and monitors the status.

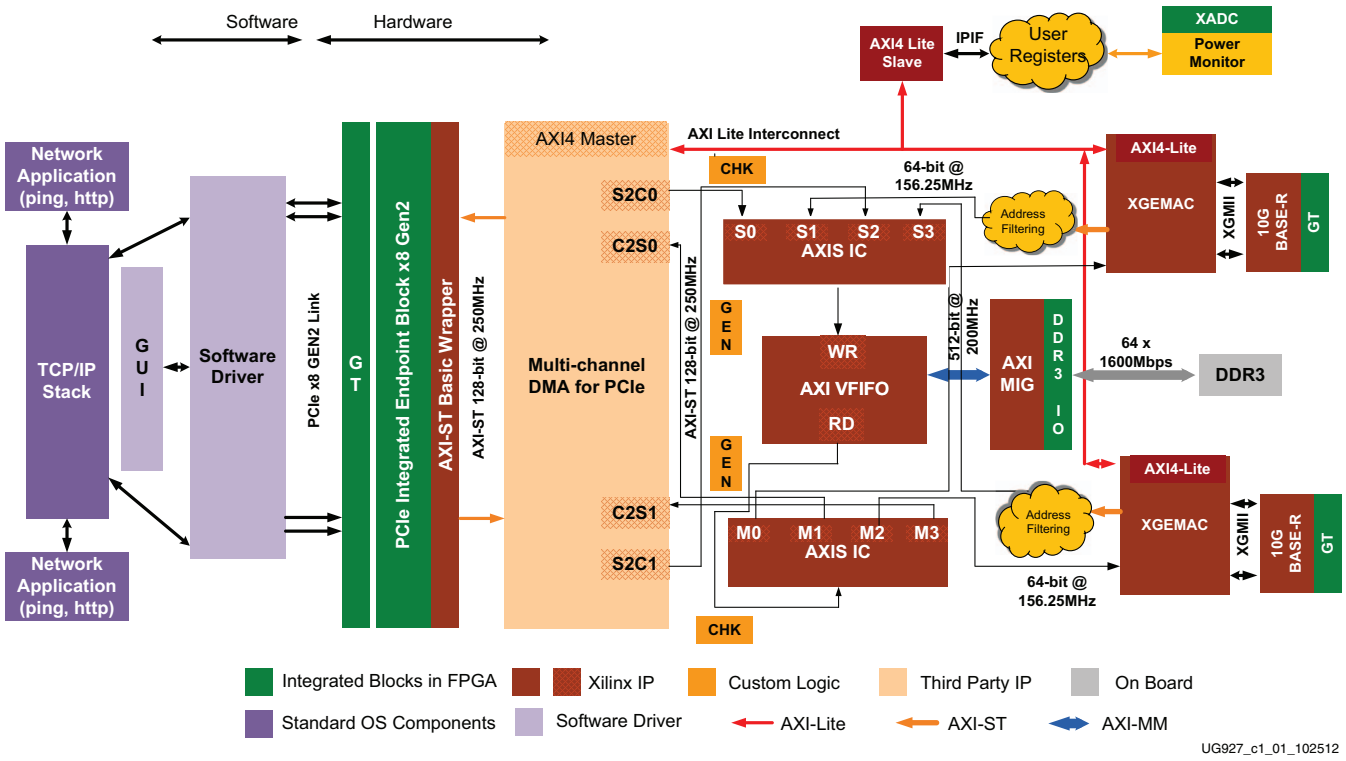


Figure 1-1: **Kintex-7 FPGA Connectivity TRD Block Diagram**

Notes:

1. The arrows in Figure 1-1 indicate AXI interface directions from master to slave. They do not indicate data flow directions.
2. The DMA netlist shipped along with this core is an evaluation netlist. For full version netlist, contact Northwest Logic.
3. The DMA used in the TRD is configured for x8Gen2 PCIe interface. For other supported configurations, contact Northwest Logic.

Operation Modes

The design offers the following modes of operation – all available within a single design bitstream.

PCIe-DMA Performance Mode

This mode demonstrates performance of the GEN2 x8 PCIe-DMA followed by a packet generator and a checker in hardware (see Figure 1-2).

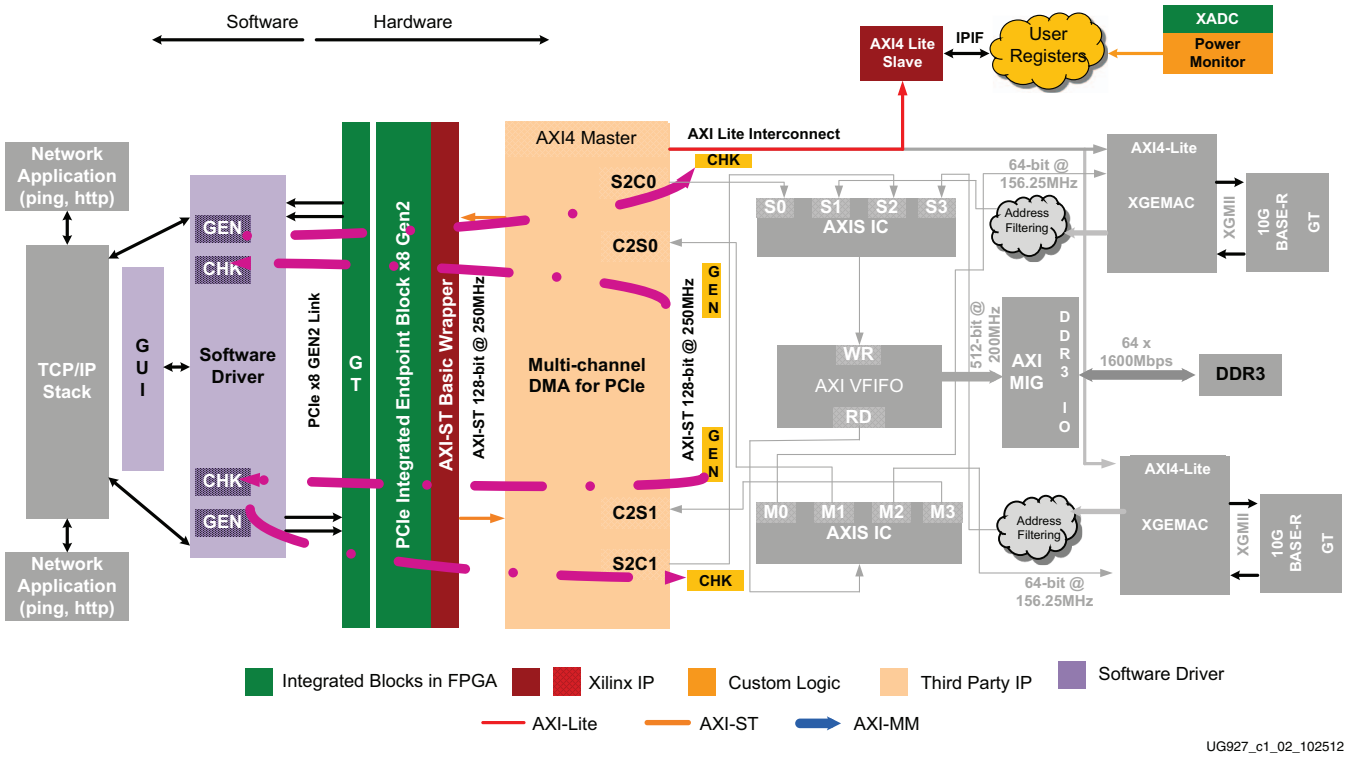


Figure 1-2: **PCIe-DMA Performance Mode**

Capability of the PCIe-DMA system standalone is performed without involvement of any further design blocks.

PCIe-DMA Performance mode supports:

1. **Loopback Mode**: Software generates packets in user space. These packets are sent to hardware over PCIe-DMA, returned back to the software driver, and are tested for integrity.
2. **Generator Mode**: Hardware generates packets and the software driver checks them for integrity.
3. **Checker Mode**: The software driver generates packets in user space. These packets are sent to hardware and then checked for integrity.

All the above modes of operation are user configurable through register programming.

Raw Ethernet Performance Mode

This mode demonstrates performance of the 10G Ethernet path showcasing hardware design capability for high performance (see Figure 1-3).

The software driver generates raw broadcast Ethernet frames with no connection to the networking stack.

The packet originates at the user space and moves to the FPGA through PCIe-DMA, traverses through DDR3 based Virtual FIFO, XGEMAC and 10GBASE-R PHY, where it is looped back through the other network channel and sent back to the software driver.

This only supports the Loopback mode of operation.

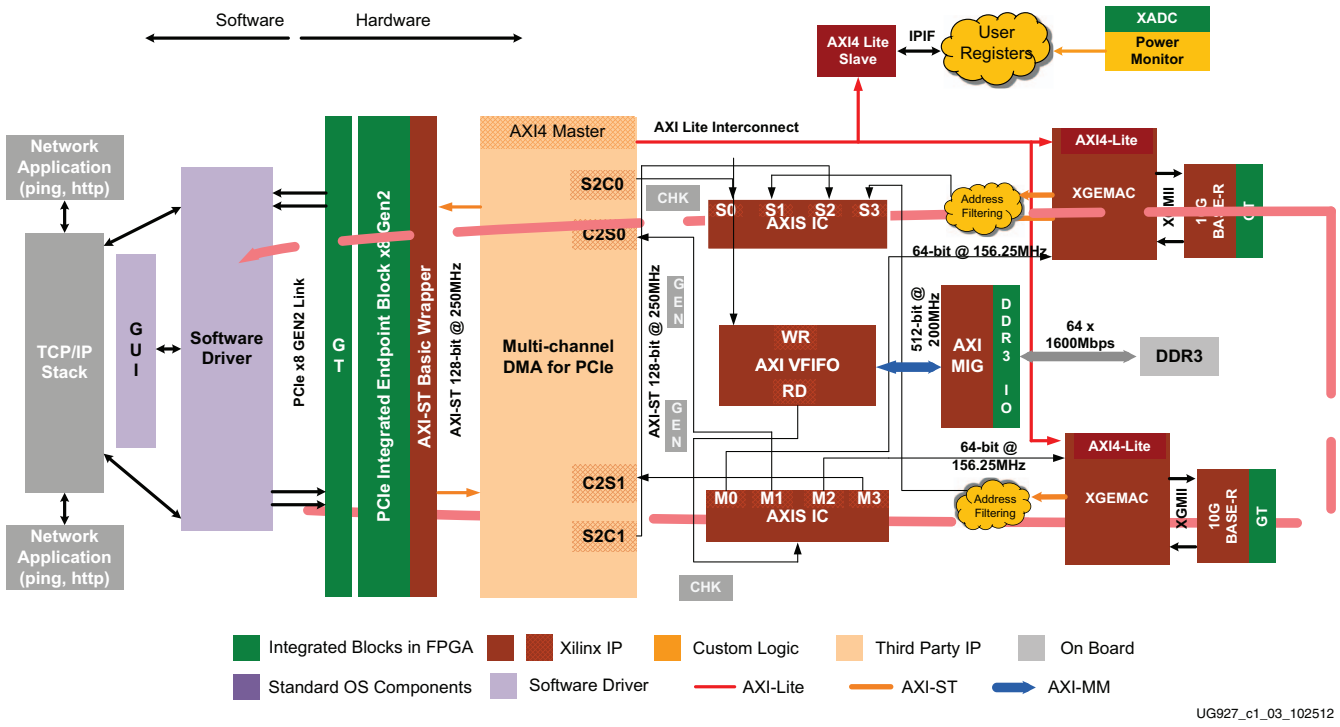


Figure 1-3: **Raw Ethernet Performance Demo**

Application Mode

This mode demonstrates end-to-end application like a dual 10G NIC. The software driver hooks up to the networking stack and standard networking applications can be used. However, due to lack of offload engine in hardware, the performance remains low.

The packets traverse through TCP/IP stack due to invocation of various standard networking applications.

Features

The Kintex-7 Connectivity TRD features are divided into base features and application features.

Base Features

This section lists the features of the PCIe and DMA, which form the backbone of the design:

- PCI Express® v2.1 compliant x8 Endpoint operating at 5 Gb/s/lane/direction
- PCIe transaction interface utilization engine
- MSI and legacy interrupt support
- Bus mastering scatter-gather DMA
- Multi-channel DMA
- AXI4 streaming interface for data
- AXI4 interface for register space access

- DMA performance engine
- Full duplex operation
  - Independent transmit and receive channels

Application Features

This section lists the features of the developed application:

- 10 Gigabit Ethernet MAC with 10G BASE-R PHY
  - Address filtering
  - Inter-frame gap control
  - Jumbo frame support up to 16,383 bytes in size
  - Ethernet statistics engine
  - Management interface for configuration (MDIO)
- Picoblaze based PVT monitoring
  - Engine in hardware to monitor power by reading TI's UCD9248 power controller chip on-board KC705
  - Engine in hardware to monitor die temperature and voltage rails via Xilinx Analog-to-Digital Converter
- Application demand driven power management
  - Option to change PCIe link width and link speed for reduced power consumption in lean traffic scenarios

Resource Utilization

Resource utilization is shown in Table 1-1.

Table 1-1: **Resource Utilization**

Resource	Total Available	Usage
Slice Registers	407,600	96,794 (23.74%)
Slice LUT	203,800	75,173 (36.88%)
RAMB36E1	445	132 (29.66)
MMCME2_ADV	10	4 (40%)
PLLE2_ADV	10	1 (10%)
BUFG/BUFGCTRL	32	10 (31.25%)
XADC	1	1 (100%)
IOB	500	134 (26%)
GTXE2_CHANNEL	16	10 (62%)
GTXE2_COMMON	4	3 (75%)

Note: The resource utilization number refers to the build corresponding to Vivado® Design Suite 2014.3 release. The utilization number might vary for a newer Vivado design suite build.

XILINX

Chapter 2

Getting Started

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

Requirements

Simulation Requirements

TRD simulation requires:

1. QuestaSim Simulator
2. Xilinx simulation libraries compiled for QuestaSim

Test Setup Requirements

Testing the design in hardware requires:

1. KC705 Evaluation board with XC7K325T-2FFG900 FPGA
2. Design files
   a. Design source files
   b. Device driver files
   c. FPGA programming files

   Design files are available at the Kintex-7 FPGA Connectivity Kit Documentation website.
3. Vivado® Design Suite
4. Micro USB cable
5. FM-S14 quad SFP+ FMC
6. Two SFP+ connectors with Fiber Optic cable
7. Fedora 16 LiveDVD
8. PC with PCIe v2.0 slot. Recommended PCI Express® Gen2 PC system motherboards are ASUS P5E (Intel X38), ASUS Rampage II Gene (Intel X58) and Intel DX58SO (Intel X58). Note the Intel X58 chipsets tend to show higher performance. This PC could also have Fedora Core 16 Linux OS installed on it.

Note: This document refers to the initially released TRD version (v1_0). For subsequent releases, the design version will be upgraded but the change will not be reflected in this document.

Hardware Demonstration Setup

This section details the hardware setup and use of provided application and control GUI to help the user get started quickly with the hardware. It provides a step-by-step explanation on hardware bring-up, software bring-up, and use of the application GUI.

All procedures listed in the following sections require super user access on a Linux machine. When using Fedora 16 LiveDVD provided with the kit, super user access is granted by default due to the way the kernel image is built; if LiveDVD is not used contact the system administrator for super user access.

1. With the power supply turned off, ensure that switches P1 and P2 on the FM-S14 FMC card are in the ON position, as shown in Figure 2-1.

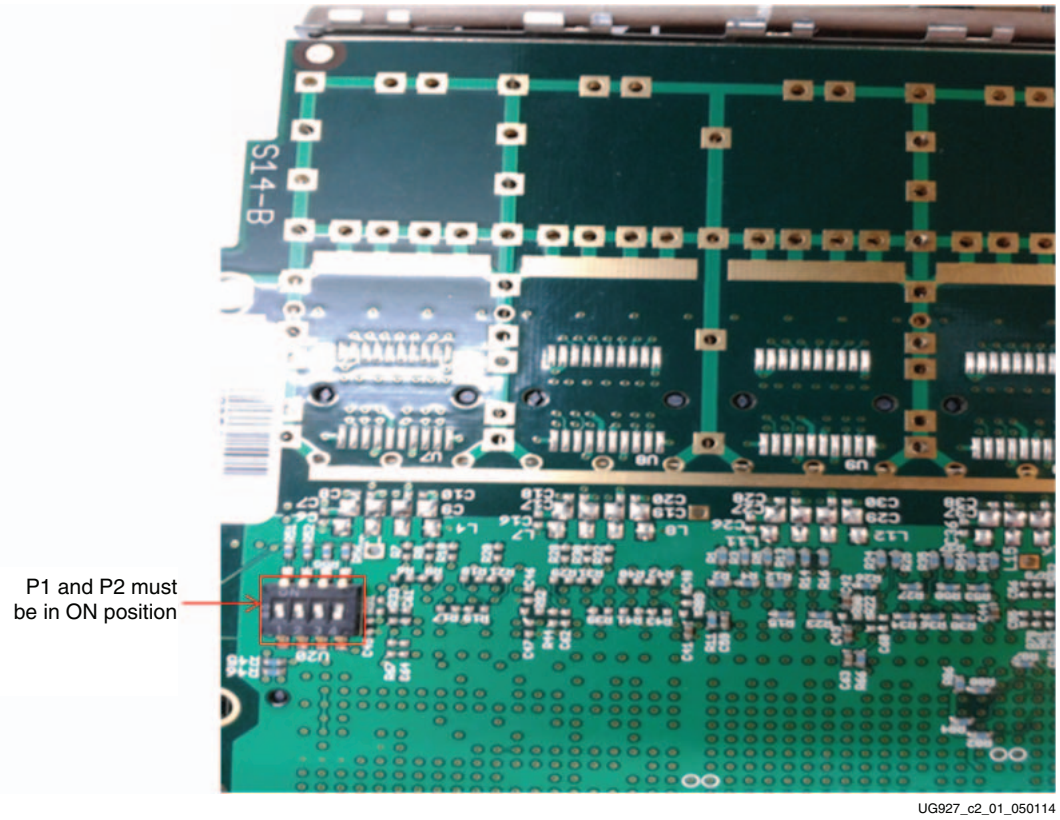
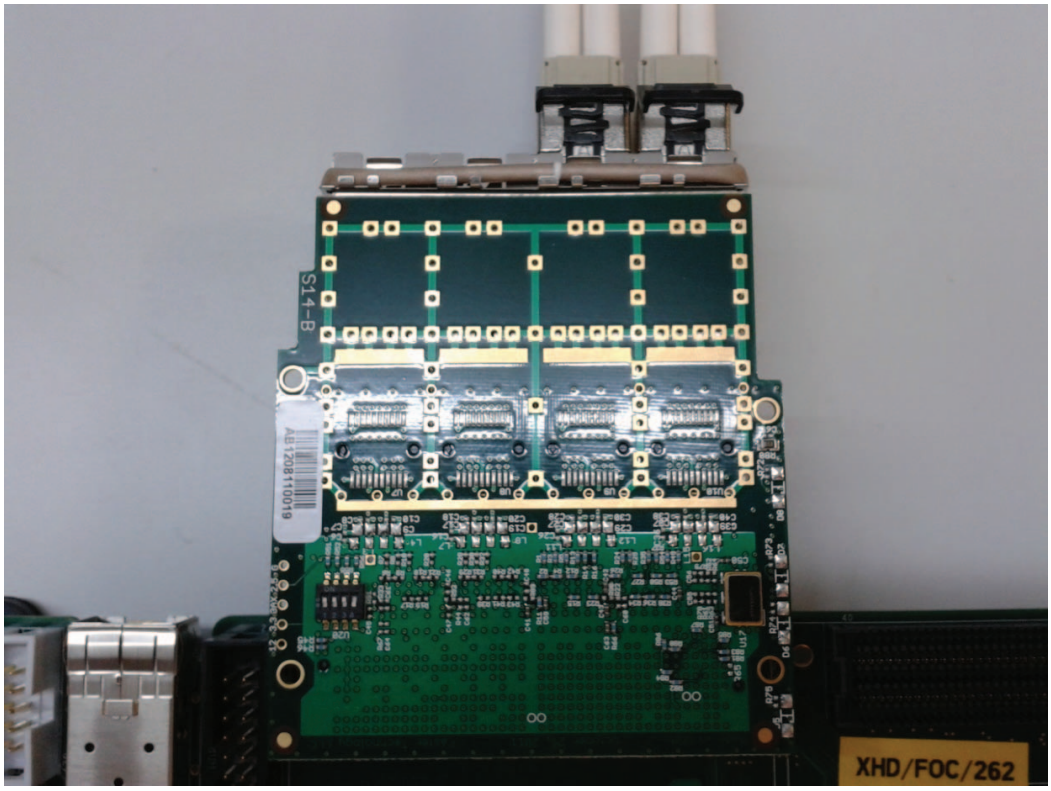


Figure 2-1: **DIP Switch Position on FMC Card**

2. Insert SFP+ connectors to channel 2 and channel 3 positions as shown in Figure 2-2.



UG927_c2_02_050114

Figure 2-2: **SFP+ Connector Position on FMC Card**

3. Insert the FM-S14 FMC card to the HPC slot of KC705 as shown in Figure 2-3. Remove the cap from the fiber optic cables and connect the fiber optic cables in a loopback fashion as shown in the figure.

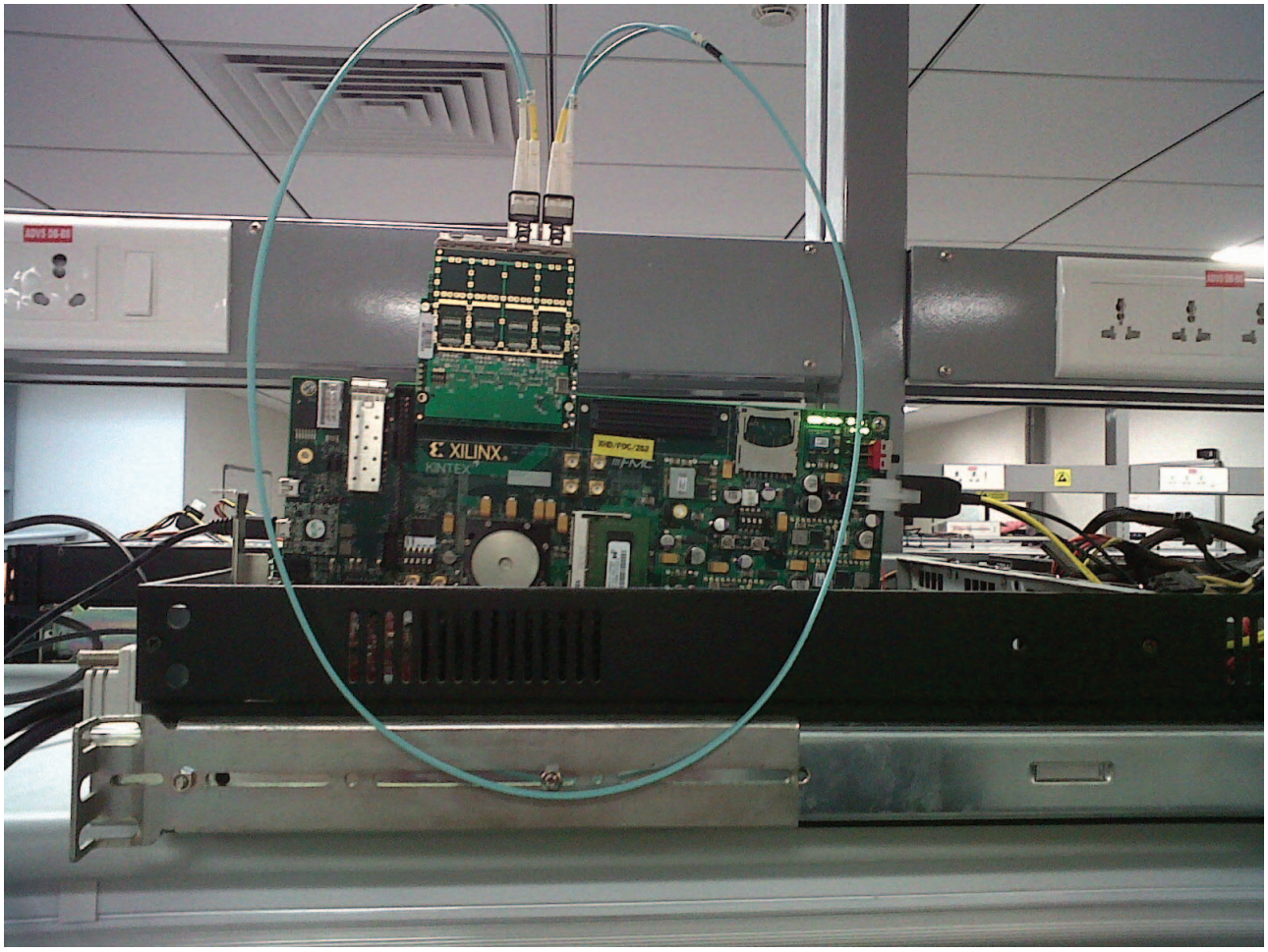


Figure 2-3: **Setup with Fiber Optic Cable**

4. Connect the 12V ATX power supply 4-pin disk drive type connector to the board. Note that the 6-pin ATX supply cannot be connected directly to the KC705 board and the 6-pin adapter is required.

   Caution! Do NOT plug a PC ATX power supply 6-pin connector into J49 on the KC705 board The ATX 6-pin connector has a different pinout than J49. Connecting an ATX 6-pin connector into J49 will damage the KC705 board and void the board warranty.

5. With the host system powered off, insert the KC705 board in the PCI Express slot through the PCI Express x8 edge connector.
6. Ensure that the connections are secure so as to avoid loose contact problems. Power on the KC705 board and then the system.

7. The GPIO LEDs are located in the top right corner of the KC705 board. These LED indicators illuminate to provide the following status (LED positions are marked from left to right):

   LED position 1 – DDR3 link up

   LED position 2 – 10GBASE-R link 1 ready

   LED position 3 – 10GBASE-R link 2 ready

   LED position 4 – 156.25 MHz clock heart beat LED

   LED position 5 – PCIe x8 link stable

   LED position 6 – PCIe 250 MHz clock

   LED position 7 – PCIe link up

   LED positions on the KC705 board are shown in Figure 2-4.

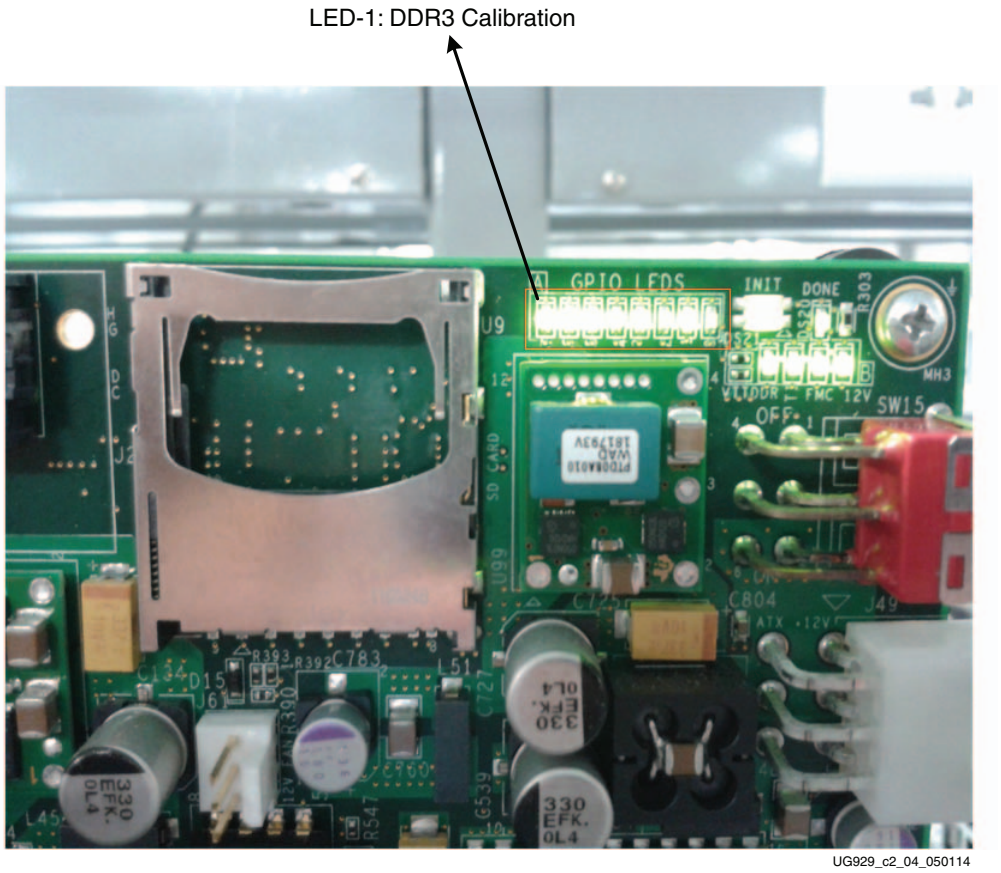


Figure 2-4: **LED Position on the FMC Card**

8. The LEDs on the FMC card (note that these are on the bottom side) indicate the following status:

   LED position top – FM-S14 is connected on the correct FMC connector on KC705 board

   LED position bottom – indicates clock generator on FMC is programmed to generate 312.5 MHz as required by the TRD

Installing the Linux Device Drivers

This sections describes the steps to install the device drivers for the Kintex-7 Connectivity TRD after completion of the above hardware setup steps.

1. If Fedora 16 is installed on the PC system's hard disk, boot as a root-privileged user, proceed to step 3. Otherwise continue with step 2.
2. To boot from the Fedora 16 LiveDVD provided in the kit, place the DVD in the PC's CD-ROM drive. The Fedora 16 Live Media is for Intel-compatible PCs. The DVD contains a complete, bootable 32-bit Fedora 16 environment with the proper packages installed for the 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: Users might have to adjust BIOS boot order settings to enure that the CD-ROM drive is the first drive in the boot order. To enter the BIOS menu to set the boot order, press 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.)

   The PC should boot from the CD-ROM drive. The images in Figure 2-5 are seen on the monitor during boot up. (Booting from Fedora 16 LiveDVD takes few minutes – wait for until Fedora 16 menu pops up on the screen as shown in Figure 2-5.)

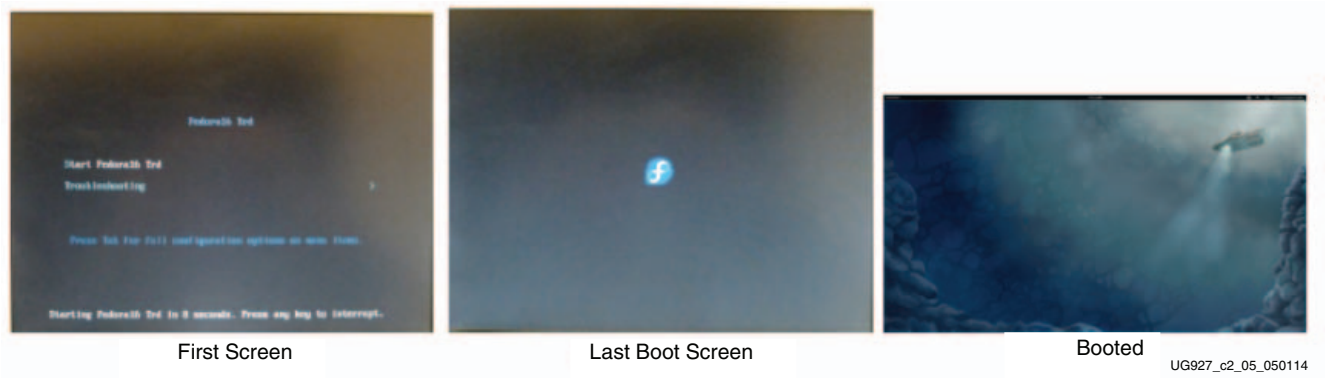


Figure 2-5: **Fedora 16 LiveDVD Boot Sequence**

3. Copy the `k7_connectvity_trd_v1_0` folder to the home directory (or a folder of choice). Note that the user must be a root-privileged user. (Connectivity kit design files are available at the Kintex-7 FPGA Connectivity Kit Documentation webpage.

   Double-click the copied `k7_connectvity_trd_v1_0` folder. The screen capture in Figure 2-6 shows the content of the `k7_connectvity_trd_v1_0` folder. The user needs to browse through the "Activities" tab after Fedora 16 boots up to access the "Home" directory.

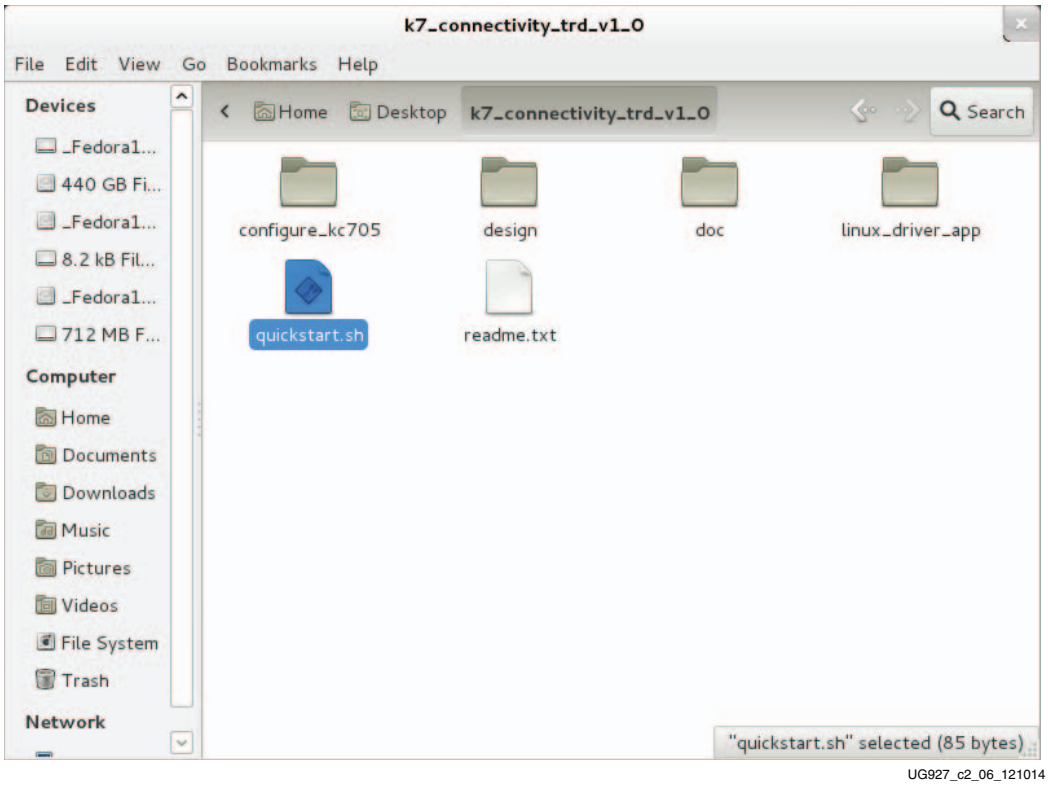


Figure 2-6: **Directory Structure of k7_connectivity_trd**

4. Ensure that the TRD package has the proper "execute" permission. Double-click `quickstart.sh` script (see Figure 2-7). This script invokes the driver installation GUI. Click **Run in Terminal**.

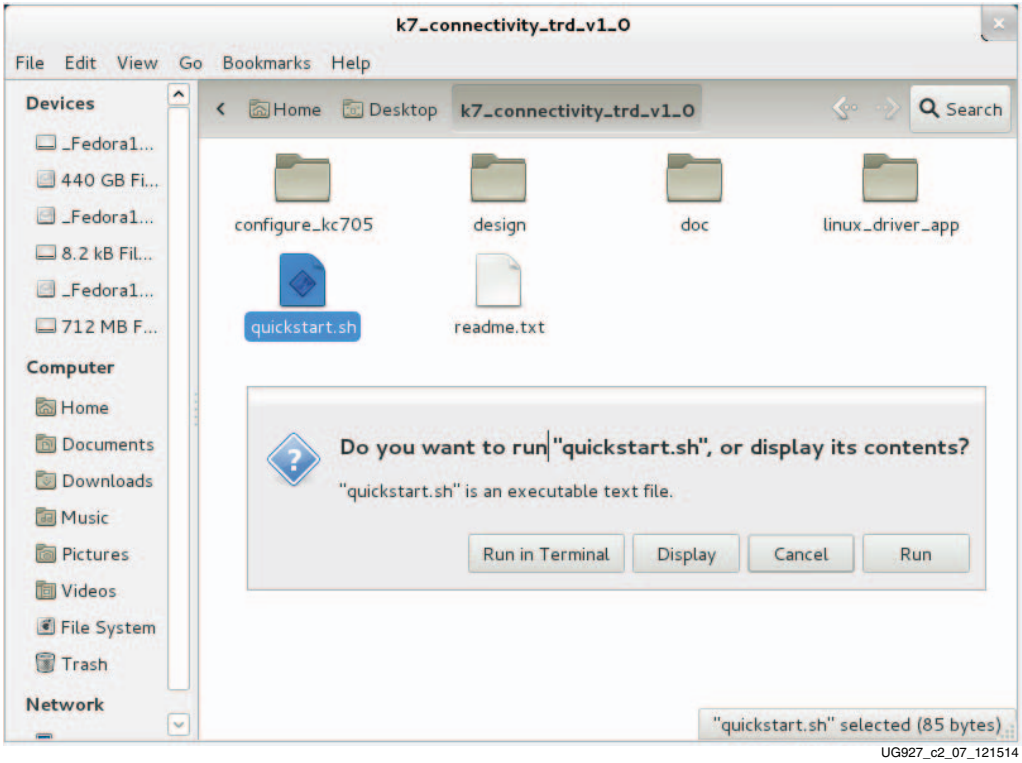


Figure 2-7: **Running the Quickstart Script**

5. The GUI with driver installation option pops up as shown in Figure 2-8. The next steps demonstrate all modes of design operation by installing and un-installing various drivers.

   Select **GEN/CHK** Performance mode driver mode as shown in Figure 2-8 and click **Install**.

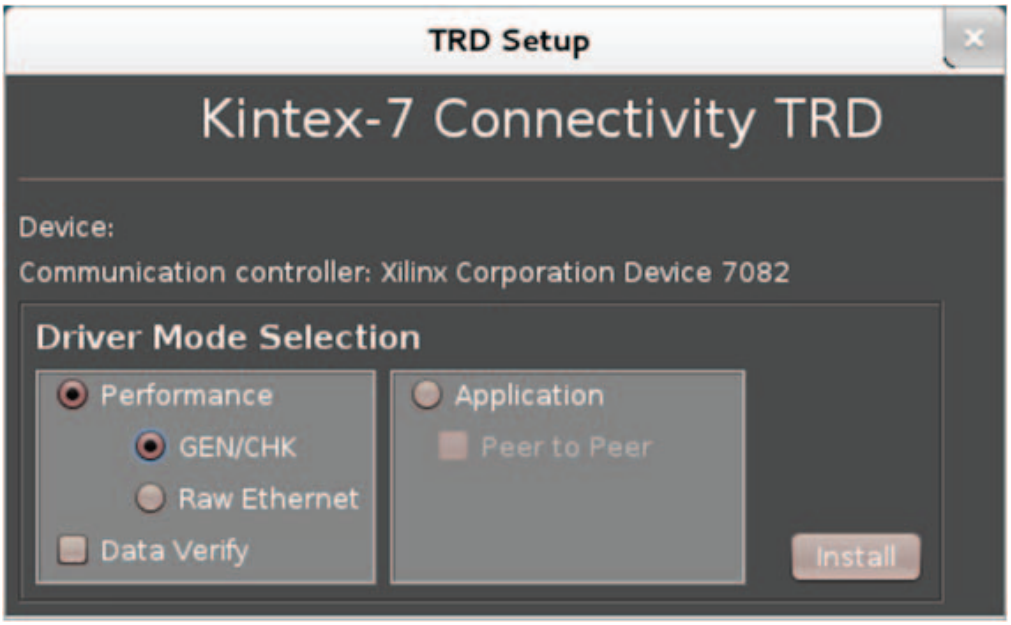


Figure 2-8: **Landing Page of Kintex-7 TRD**

Installing the Windows Device Driver

PC requirements for installing the Windows device drivers:

- Operating System: Windows 7 (32 bit or 64 bit)
- Java installation: Java SE Development Kit 7u5 and Java SE Runtime Environment 7u5
- Hardware Demonstration Setup, page 14 has been completed

To install the drivers:

1. Restart the computer. During bootup, select **Windows 7** from the boot menu, and press the **F8** key to go to the Advanced Boot Options (Figure 2-9).

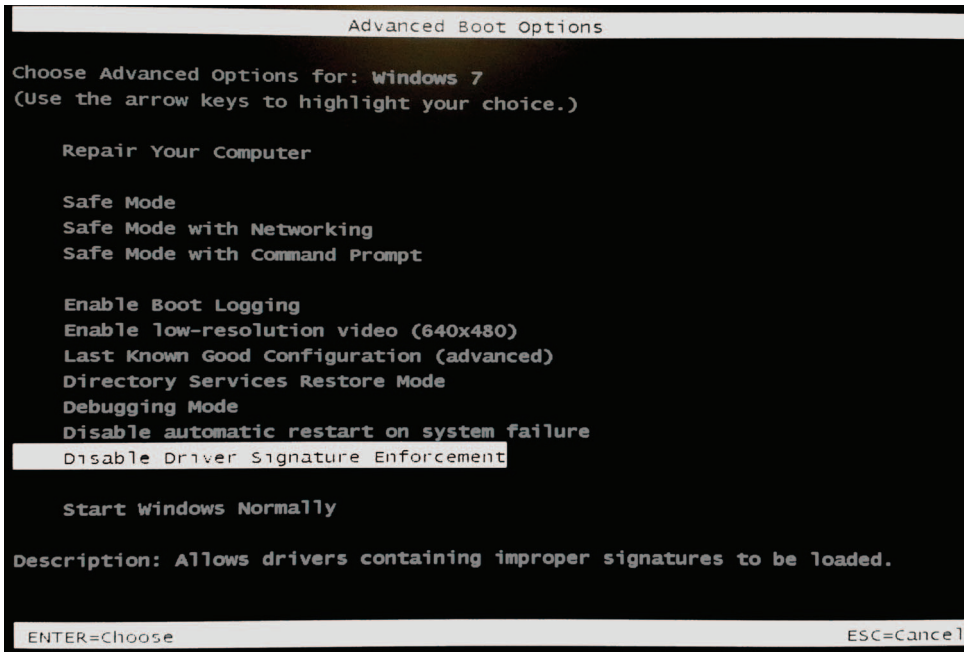


Figure 2-9: **Windows 7 Advanced Boot Menu**

2. **Select Disable Driver Signature Enforcement** (Figure 2-9) and press the **Enter** key to boot Windows.
3. Download `rdf0282-k7-connectivity-trd-2014-1.zip` from the Kintex-7 FPGA Connectivity Kit Documentation webpage to the desktop (or a folder of choice).
4. Double-click on the `rdf0282-k7-connectivity-trd-2014-1.zip` file and navigate to the `k7_connectivity_trd` folder.
5. Execute the `quickstart.bat` file using administrative privileges by selecting **Run as Administrator** in the right-click menu.
6. When the User Account Control window opens, select **YES** to invoke the InstallShield wizard shown in Figure 2-10).

7. Click Next (Figure 2-10) to open the Customer Information window (Figure 2-11).

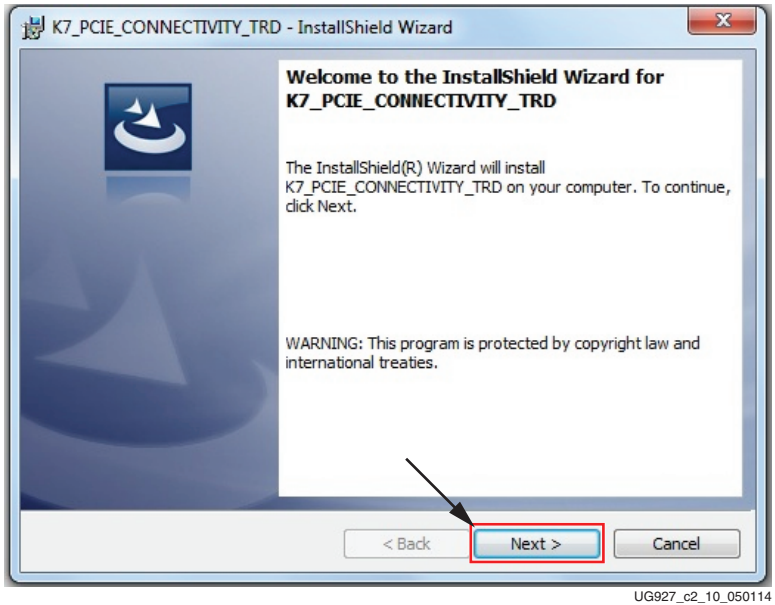


Figure 2-10: **InstallShield Wizard First Window**

8. Enter your user name and organization. Click **Next** (Figure 2-11) to open the Destination Folder window (Figure 2-12).

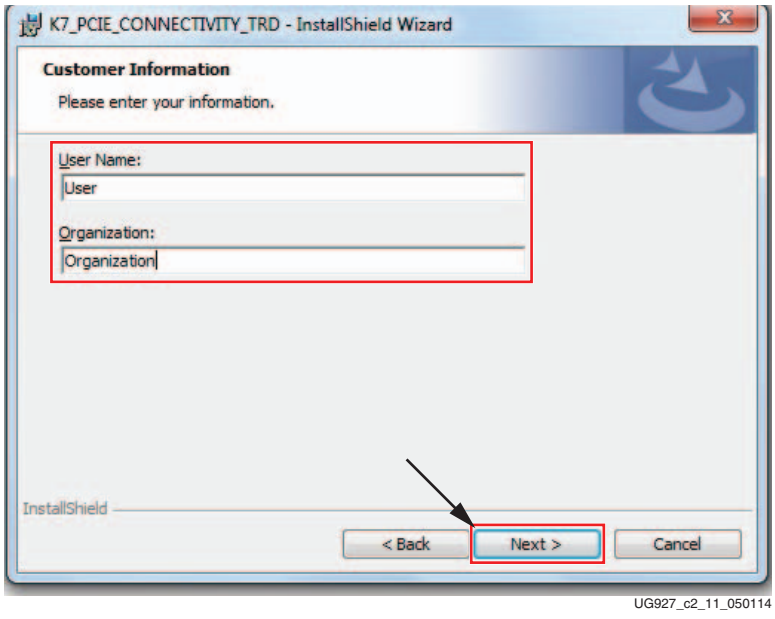


Figure 2-11: **Customer Information Window**

9. Do one of the following:
   a. Click **Next** (Figure 2-12) to copy the driver files, Java GUI, user guides and source files to their default installation locations:
      - `C:\ProgramFiles(x86)\Xilinx\K7_PCIE_CONNECTIVITY_TRD` (for 64-bit machines)

- `C:\ProgramFiles\Xilinx\K7_PCIE_CONNECTIVITY_TRD` (for x86 machines)

b. Click **Change...** (Figure 2-12) to copy the driver files, Java GUI, user guides and source files to a custom installation directory.

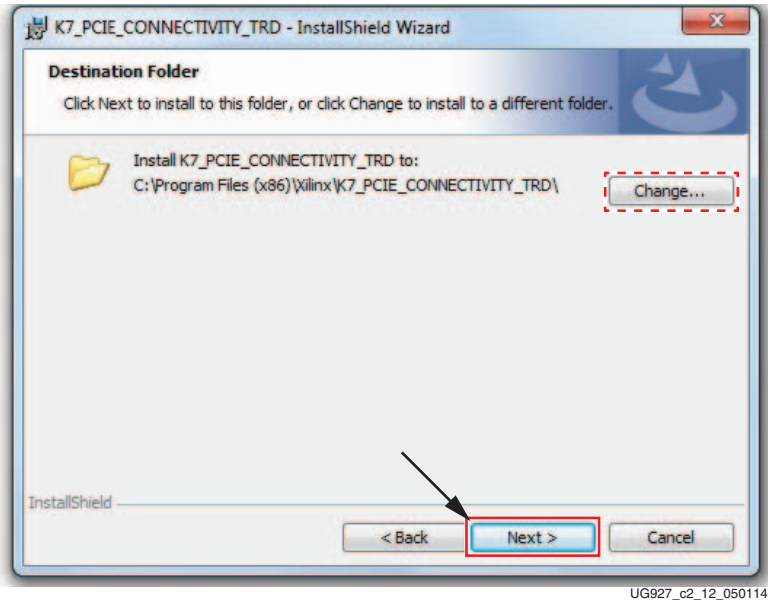


Figure 2-12: **Destination Folder Window**

10. Click **Install** (Figure 2-13) to begin installation.

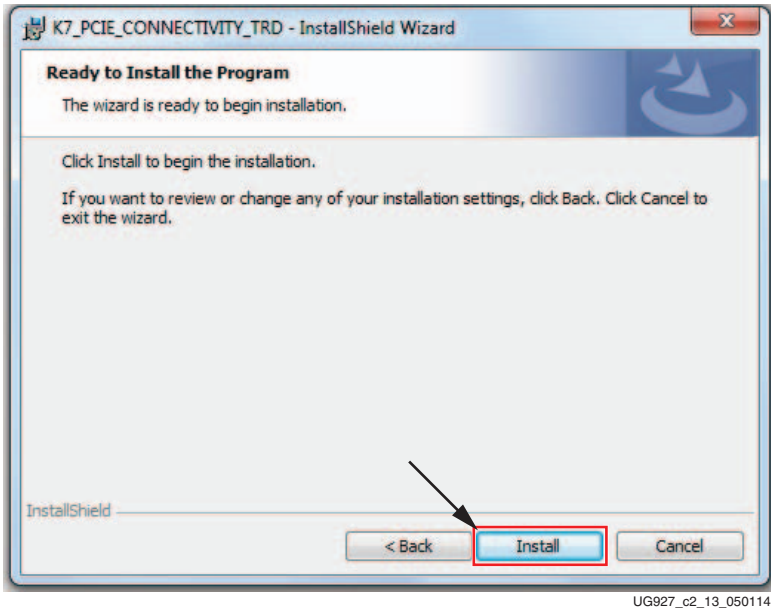


Figure 2-13: **Install Button**

11. When the Windows Security Window opens, Select **Install this driver software anyway** (Figure 2-14). This warning is seen because the drivers are unsigned.

    Note: This step is repeated two more times because this driver package contains three drivers.

Figure 2-14: **Windows Security Window**

12. When the installation is complete, Figure 2-15 is displayed. Click **Finish** to close the wizard.

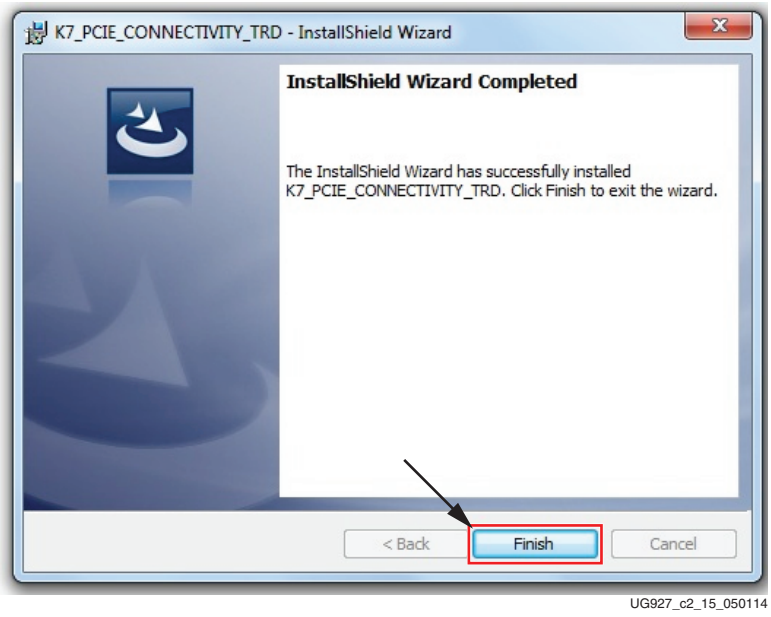


Figure 2-15: **Finish Button**

Verify Installation

After the installation is successful, verify the installed drivers are properly mapped as described here:

1. Open Device Manager (Figure 2-16). Click **Start**, click **Control Panel**, and then click **Device Manager**.
2. Right click on the computer name and select **Scan for hardware changes** (Figure 2-16).

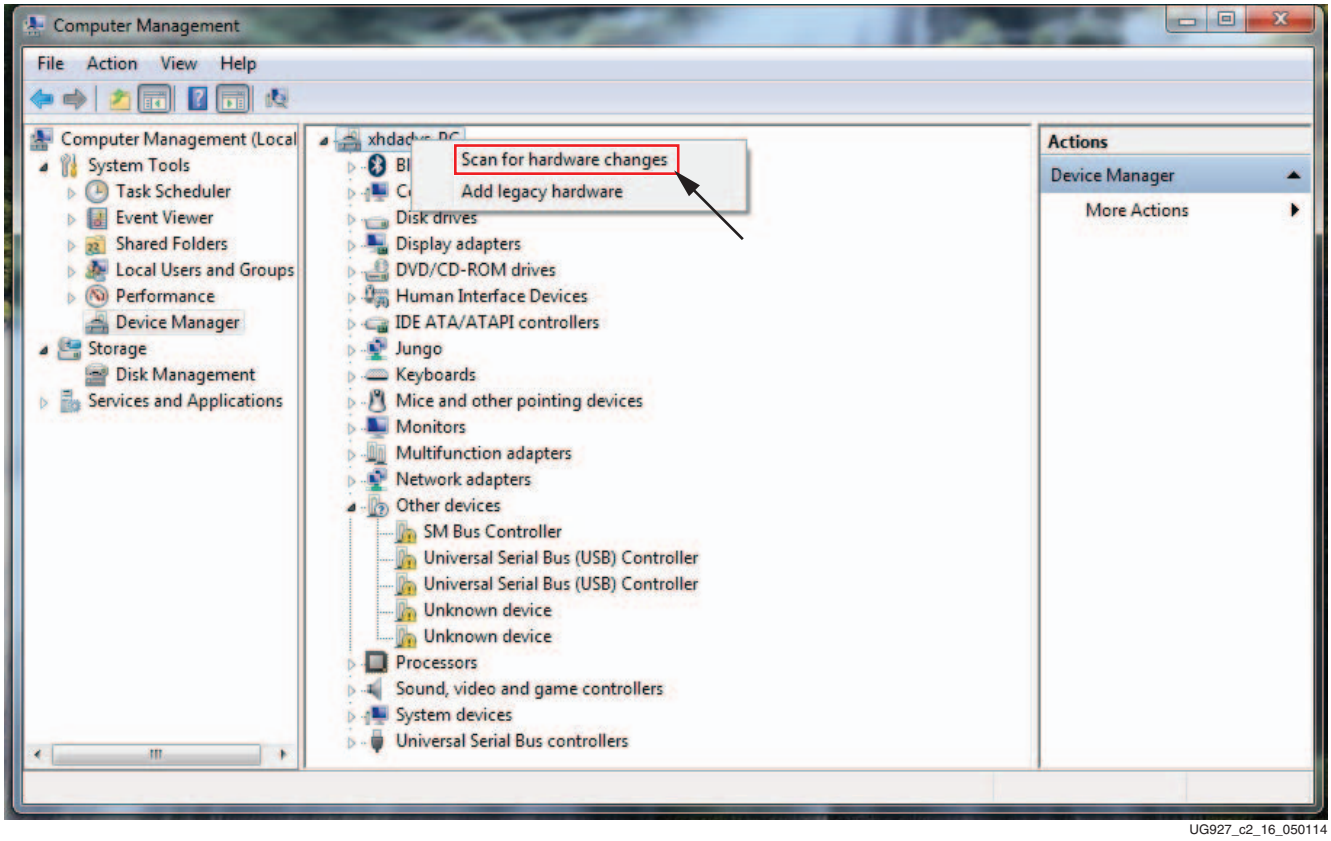


Figure 2-16: **Initiating Scan for Hardware Changes in Device Manager**

3. Figure 2-17 shows the Xilinx devices associated with the design have been detected after the scan.

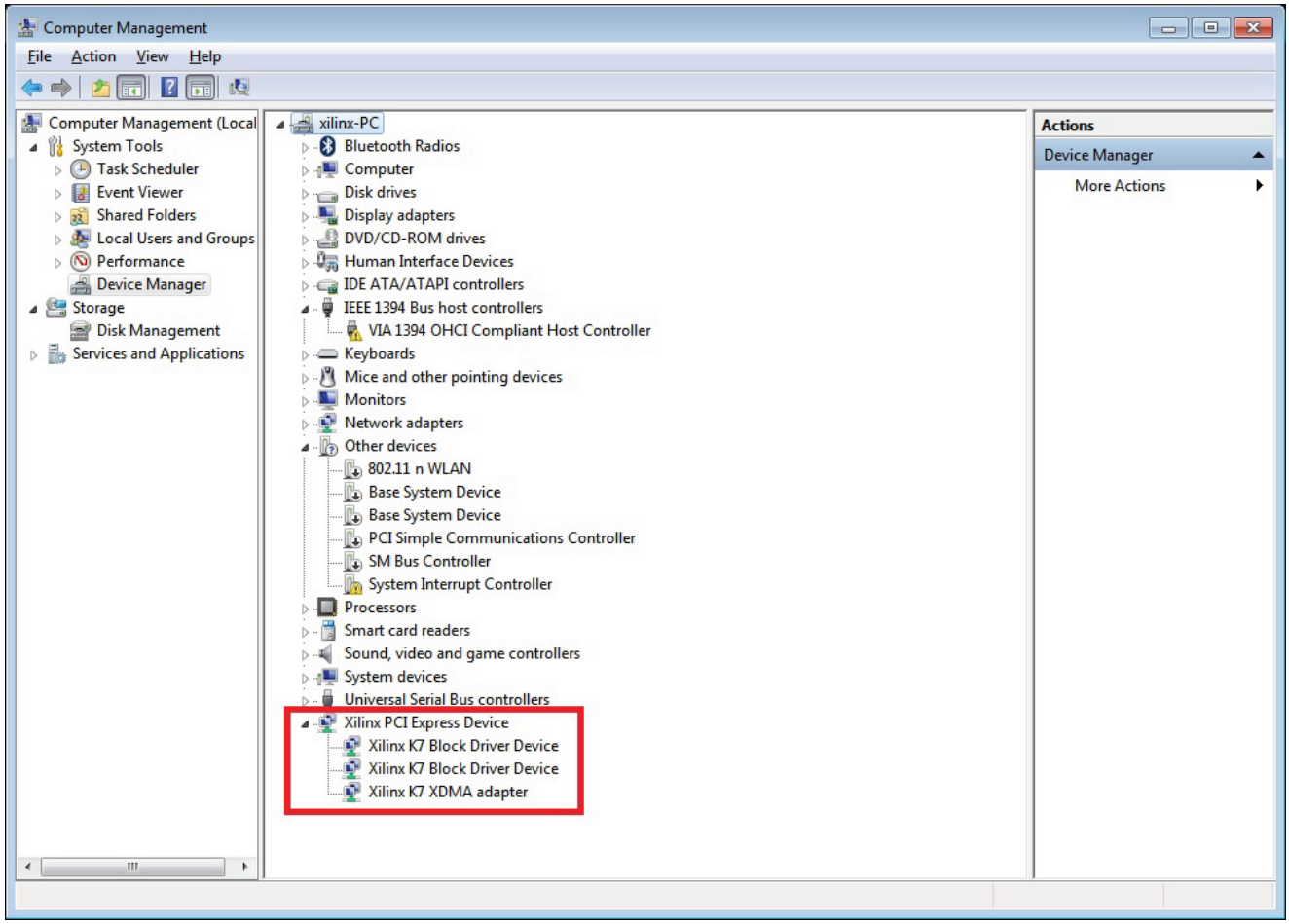


Figure 2-17: **Xilinx XDMA Adapter and Two Block Driver Devices are Detected**

4. Open a command terminal with administrative privileges. Click **Start**, enter **cmd**, and press the ***Enter*** key.
5. Navigate to the installation directory, and then to the `gui` directory. Execute **`rungui.bat`** to invoke the GUI (Figure 2-18).

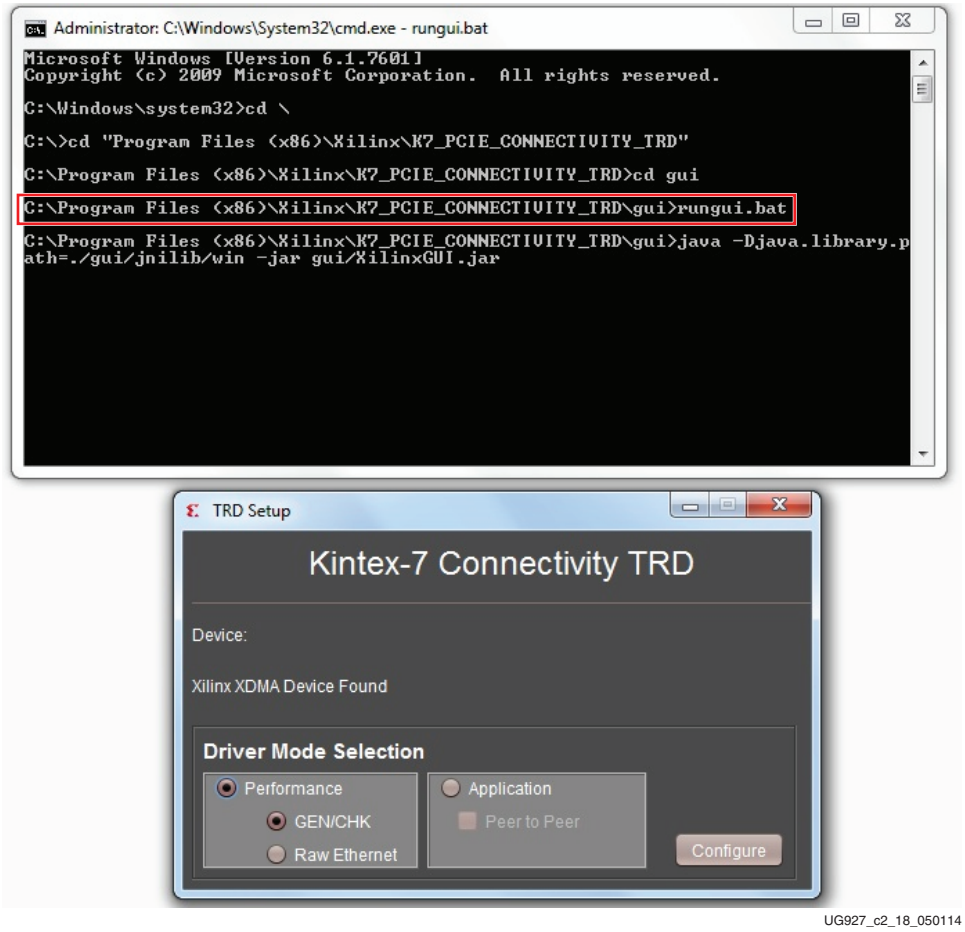
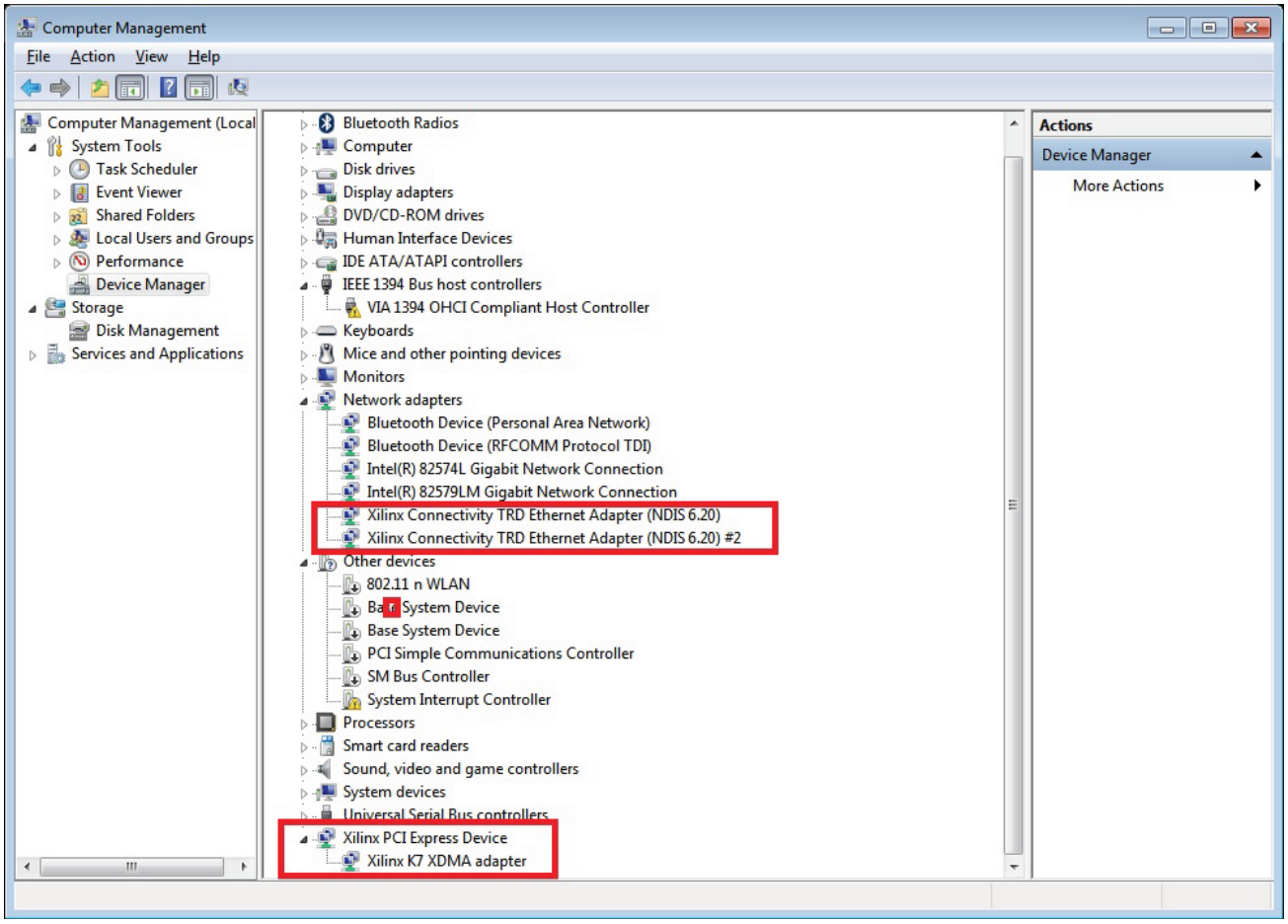


Figure 2-18: **Invoking the GUI**

6. If **application mode** is selected in the GUI, two new local area networks will be displayed in device manager, and the NDIS drivers in place of the Xilinx block drivers (Figure 2-19).



UG927_c2_19_120914

Figure 2-19: **NDIS 6.20 Ethernet Drivers for the KC705 Design**

GEN/CHK Performance Mode

1. After installing the GEN/CHK Performance Mode driver, the control and monitor user interface pops up as shown in Figure 2-20. The control pane shows control parameters such as test mode (loopback, generator, or checker) and packet length. The user can select PCIe link width and speed while running a test if the host machine supports link width and speed configuration capability. The System Monitor tab in the GUI also shows system power and temperature. DDR3 ready status and 10GBASE-R link status are displayed on the top left corner of the GUI.

UG929_c2_20_050114

Figure 2-20: **GEN/CHK Performance Mode**

2. Click **Start** on both Data Path-0 and Data Path-1. Go to the Performance Plots tab. The Performance Plots tab shows the system-to-card and card-to-system performance numbers for a specific packet size. The user can vary packet size and see performance variation accordingly (see Figure 2-21).

Figure 2-21: **GEN/CHK Performance Mode Plots**

3. Close the GUI – a pop up message asks whether you want to un-install the drivers. Click on **Yes**. This process opens the landing page of the Kintex-7 Connectivity TRD. (Driver un-installation requires the GUI to be closed first.)
4. Select **Raw Ethernet** performance as shown in Figure 2-22. Click **Install**.

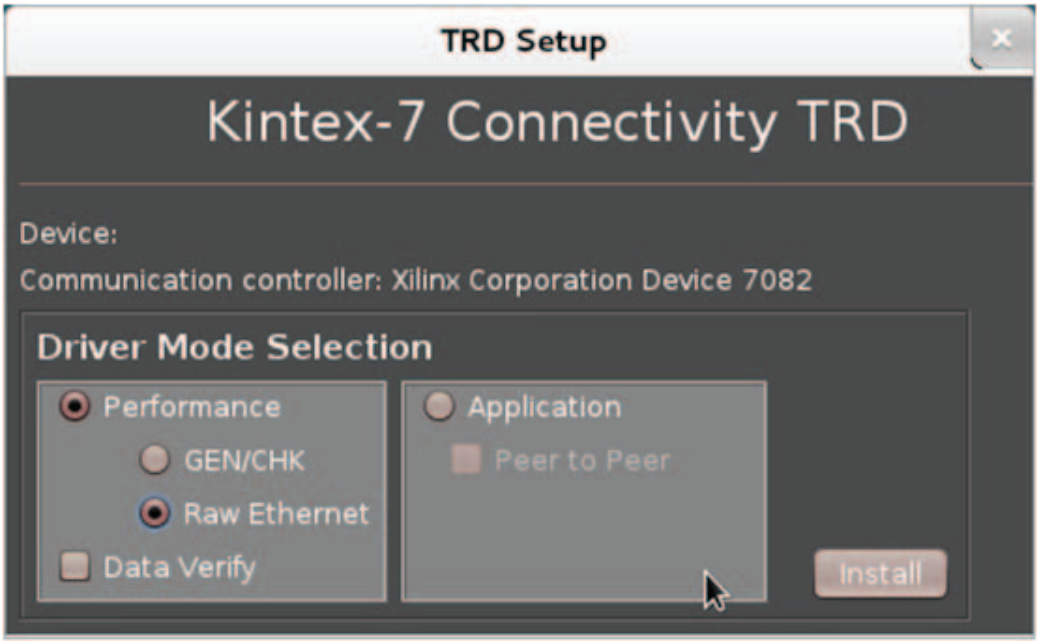


Figure 2-22: **Raw Ethernet Driver Installation**

5. The GUI for raw Ethernet mode driver is invoked. The user can configure packet size in raw Ethernet mode and can control PCIe link width and speed change if the host machine supports this. The System Monitor tab monitors system power and temperature (see Figure 2-23).

UG927_c2_23_050114

Figure 2-23: **Raw Ethernet Driver GUI**

6. Click **Start** on both Data Path-0 and Data Path-1. Navigate to the Performance Plots tab to see performance on system-to-card and card-to-system (see Figure 2-24).

Figure 2-24: **Raw Ethernet Driver Performance Plots**

7. Close the GUI – this un-installs driver and opens the Kintex-7 Connectivity TRD landing page. Note that driver un-installation requires the GUI to be closed first.
8. Select the Application mode driver as shown in Figure 2-25. For using peer-peer option refer to Appendix C, Software Application and Network Performance. Click **Install**.

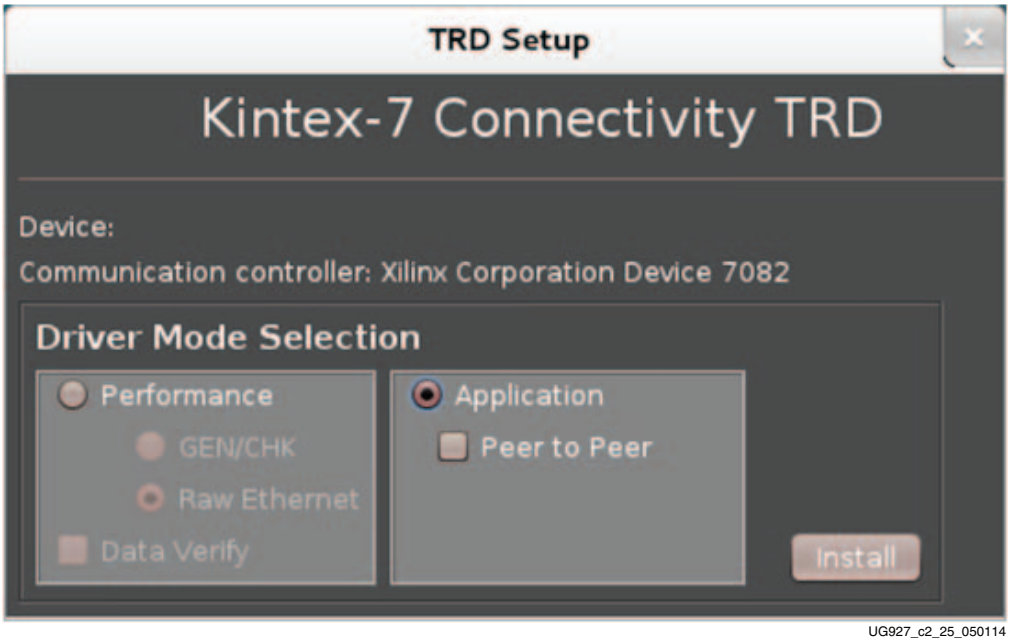
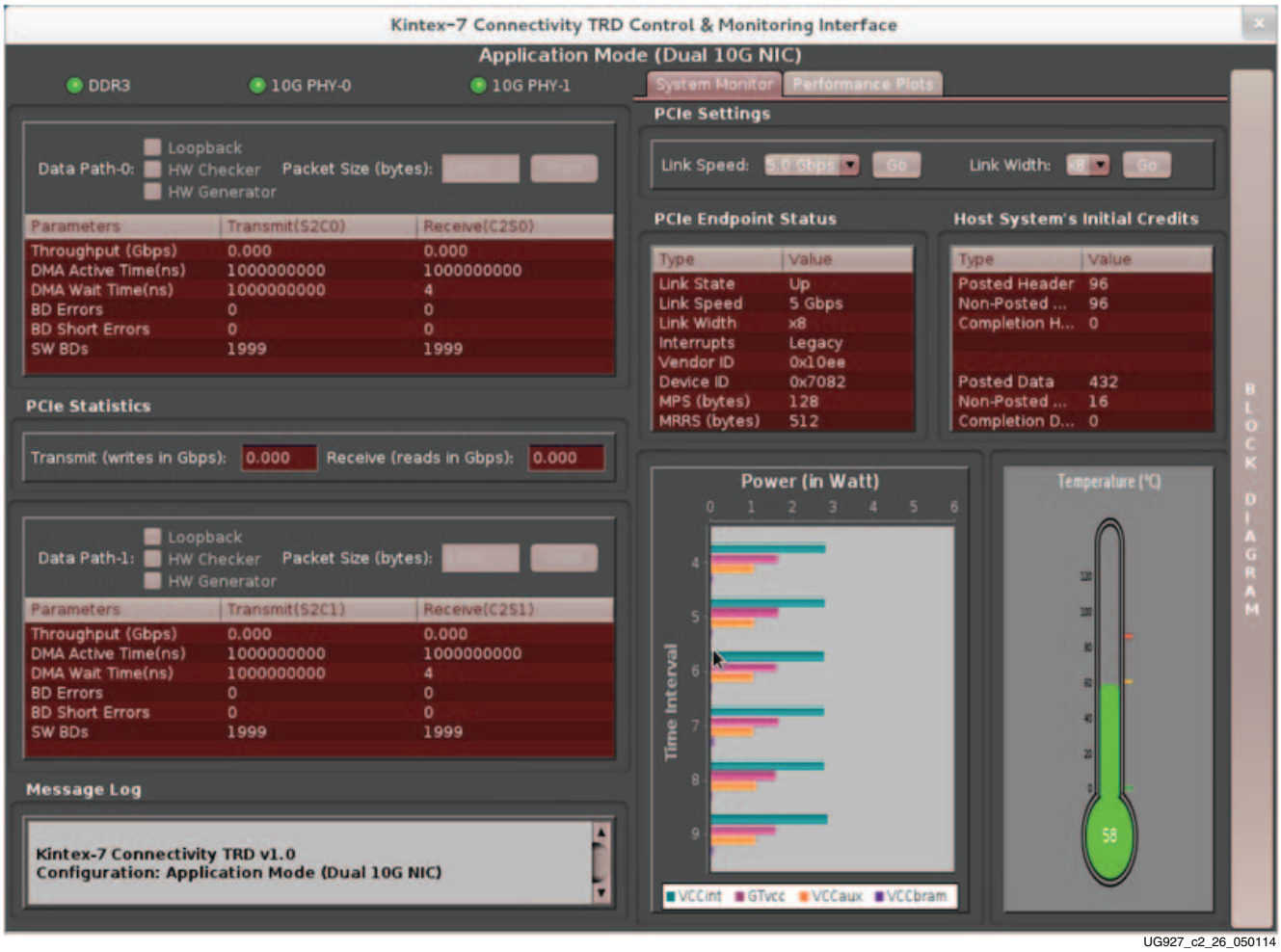


Figure 2-25: **Application Mode Driver Installation**

9. The GUI is invoked after the driver is installed. However, in Application mode, the user cannot start or stop a test – the traffic is generated by the networking stack. The system monitor shows the system power and temperature (see Figure 2-26).



UG927_c2_26_050114

Figure 2-26: **Application Mode Driver GUI**

10. Open another terminal on the host machine and run ping (see Figure 2-27) using the following command:

```
$ ping 10.60.0.1
$ ping 10.60.1.1
```

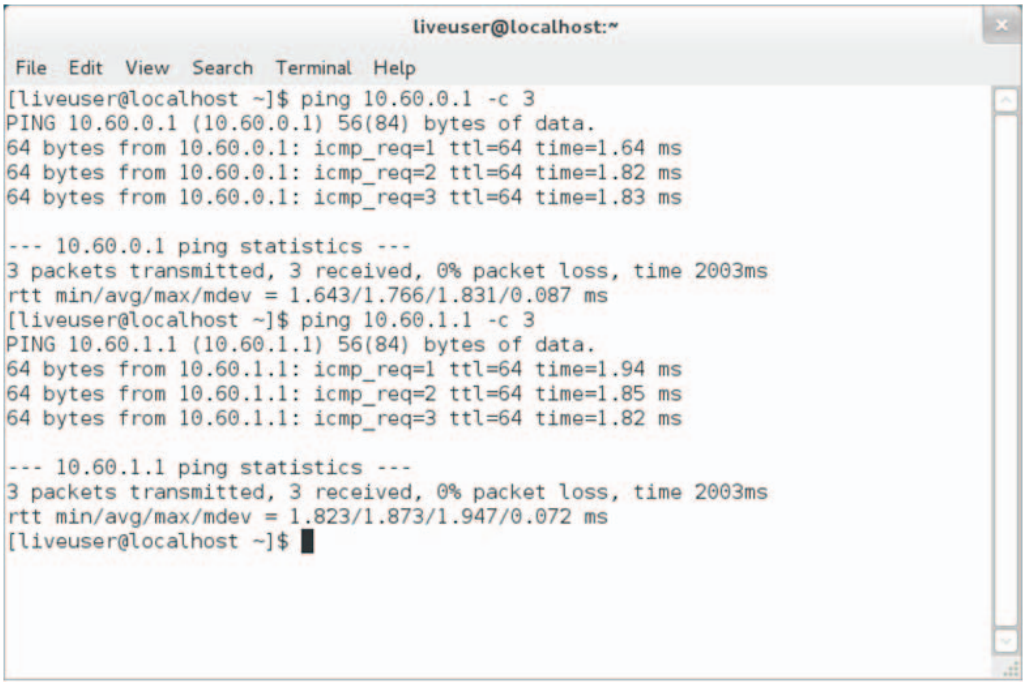
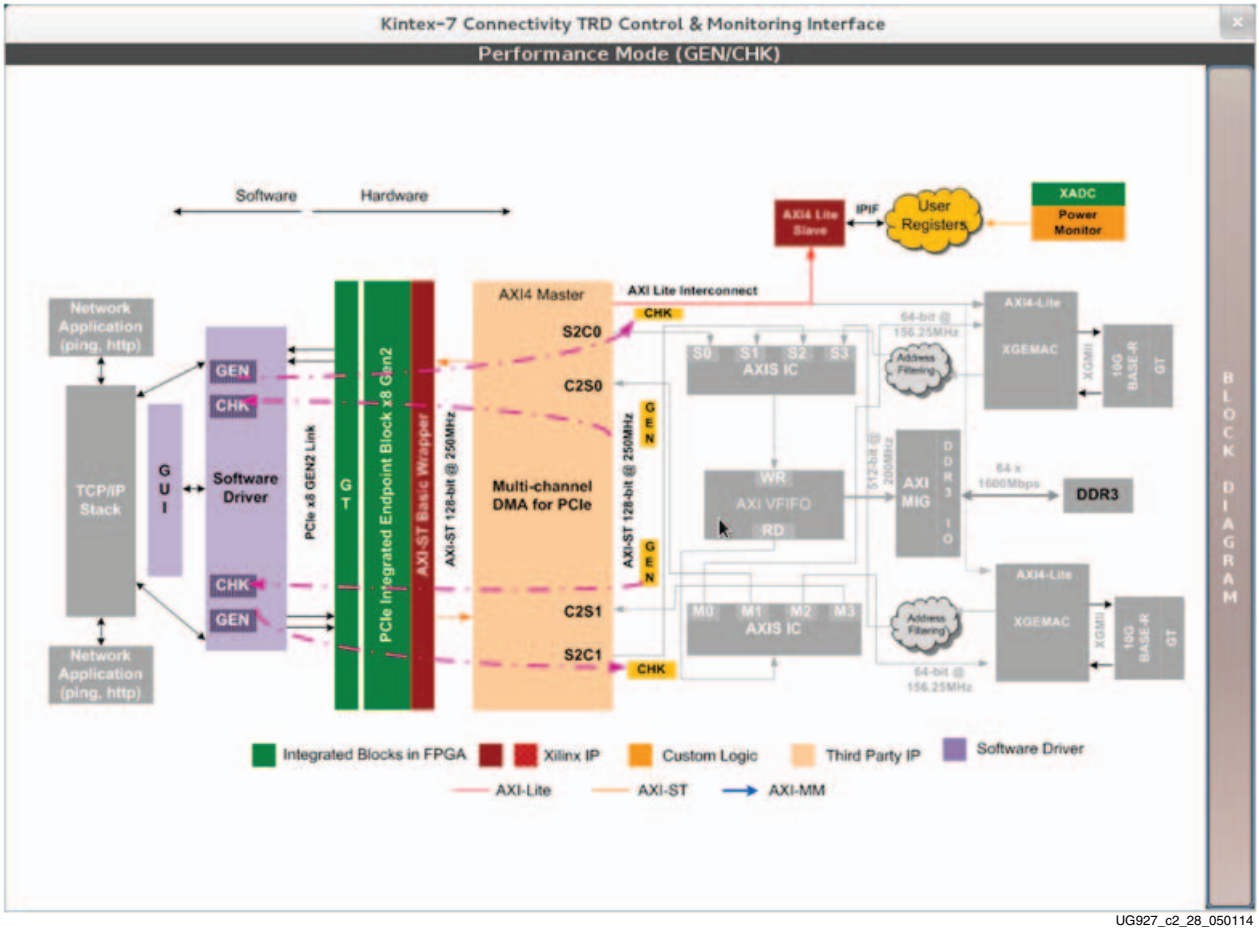


Figure 2-27: **Ping Application on Application Mode Driver**

Note: In the Windows operating system only peer to peer mode can be enabled. For enabling peer to peer mode, two PCIe compatible host PCs are required.

11. The user can click on the Block Diagram option to view the design block diagram as shown in Figure 2-28.
12. Close the GUI – this un-installs driver and opens the Kintex-7 Connectivity TRD landing page. Note that driver un-installation requires the GUI to be closed first.



UG927_c2_28_050114

Figure 2-28: **Design Block Diagram**

Ethernet Specific Features

The Ethernet specific features can be exercised by using command line utilities such as **`ifconfig`** and **`ethtool`** present in Linux.

The Ethernet driver provides functions which are used by **`ifconfig`** and **`ethtool`** to report information about the NIC. The **`ifconfig`** utility is defined as the interface configurator and is used to configure the kernel-resident network interface and the TCP/IP stack. It is commonly used for setting an interface's IP address and netmask and disabling or enabling a given interface apart from assigning MAC address, and changing maximum transfer unit (MTU) size. The **`ethtool`** utility is used to change or display Ethernet card settings. **`ethtool`** with a single argument specifying the device name prints the current setting of the specific device. More information about **`ifconfig`** and **`ethtool`** can be obtained from the manual (man) pages on Linux machines.

NIC Statistics

The NIC statistics can be obtained using the **ethtool** command:

$ **ethtool -S ethX**

The error statistics are obtained by reading the registers provided by the Ethernet Statistics IP. PHY registers can be read using the following command:

$ **ethtool -d ethX**

Certain statistics can also be obtained from the **ifconfig** command:

$ **ifconfig ethX**

Rebuilding the Design

The design can also be re-implemented using Vivado software. Before running any command line scripts, refer to the *Vivado Design Suite Migration Methodology Guide* (UG911)[Ref 1] and the *Vivado Design Suite Implementation User Guide* (UG904) [Ref 2] to learn how to set the appropriate environment variables for the operating system. All scripts mentioned in this user guide assume that 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 TRD.

Copy the `k7_connectivity_trd` files to the PC with the Vivado software installed.

The DMA netlist and AXILite Interconnect IP is located in the `k7_connectivity_trd/hardware/sources/ip_cores` directory.

Detail of various IP cores under the ip_cores directory can be obtained from readme.txt.

Design Implementation

Implementing the Design Using Vivado IP Integrator (IPI) Flow

1. Navigate to the `k7_connectivity_trd/hardware/vivado/scripts` directory.
2. To invoke the Vivado tool GUI with the design loaded, open the Vivado Design Suite command prompt and enter:

   $ vivado -source k7_conn_gui_ipi.tcl

3. Click **Run Synthesis** in the Project Manager window. A window with the message `Synthesis Completed Successfully` appears after the Vivado synthesis tool generates a design netlist.
4. Close the message window.
5. Click **Run Implementation** in the Project Manager window. A window with the message `Implementation Completed Successfully` appears after implementation is complete.
6. Close the message window.
7. Click **Generate Bitstream** in the Project Manager window. A window with the message `Bitstream Generation Successfully Completed` appears at the end of this process.
8. Navigate to the `k7_connectivity_trd/hardware/vivado/scripts` directory, and generate the MCS file:

```
vivado -source k7_conn_ipi_flash.tcl
```

This command generates the MCS file in the `k7_connectivity_trd/hardware/vivado/scripts` directory.

9. Close the Vivado GUI.
10. The generated bitfile can be found under `runs_ipi/k7_connectivity_trd.runs/impl_1/k7_connectivity_trd_top.bit`.
11. By default, the scripts generate the bitstream with the evaluation version of the Northwest Logic DMA IP.

Reprogramming the KC705 Board

The KC705 board is shipped preprogrammed with the TRD, where the PCIe link is configured as x8 at a 5 Gb/s link rate. This procedure shows how to bring back 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 flash mode of the KC705 board. This is the only configuration option that meets the strict programming time of PCI Express. Refer to, *7 Series FPGA Integrated Block for PCI Express User Guide* (PG054) [Ref 3] for more information on PCIe configuration time requirements.

Ensure that the KC705 board switches and jumper settings are as shown in Figure 2-29. Connect the micro USB cable and use the power adapter to provide 12V power to the 6-pin connector as shown in the figure.

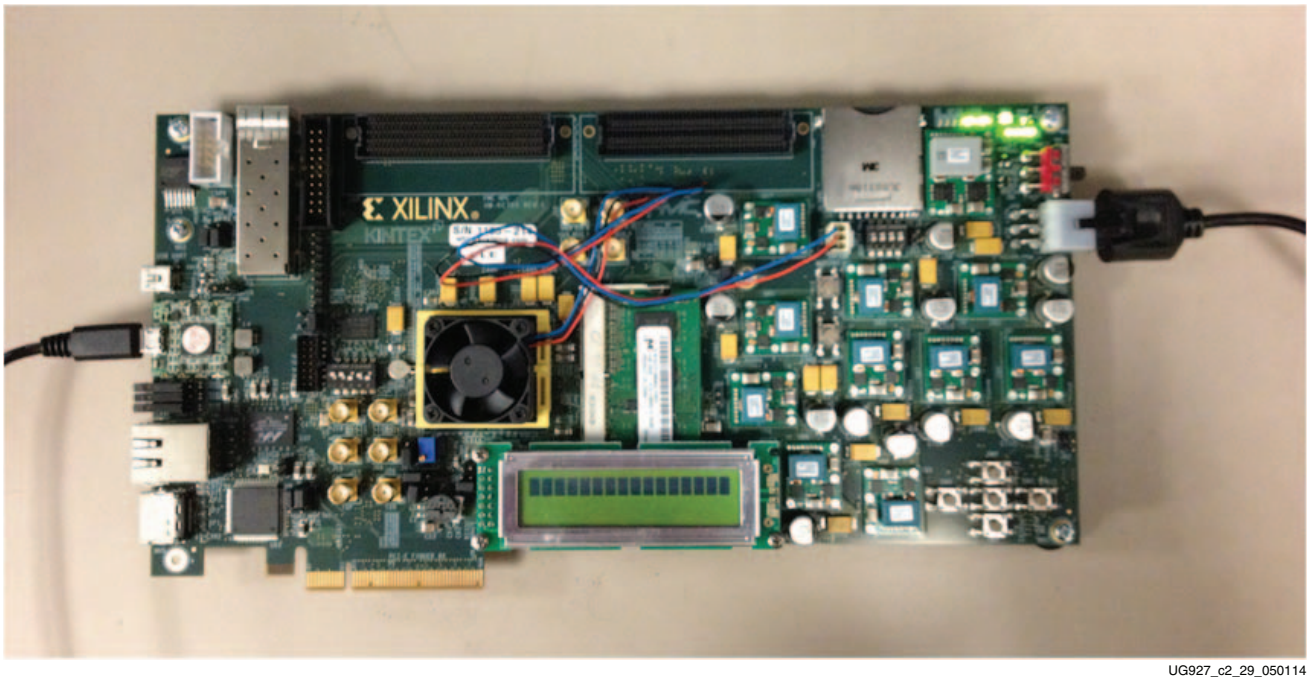


Figure 2-29: **Cable Installation for KC705 Board Programming**

To download the MCS file:

1. Open a hardware session in the Vivado GUI.
2. Connect to the hardware device (KC705 board).

3. Navigate to the `k7_connectivity_trd/hardware/vivado/scripts` directory and source the `program_flash.tcl` script.

The Kintex-7 Connectivity TRD is now programmed into the BPI flash and automatically configures at power up.

Simulation

This section details the out-of-box simulation environment provided with the design. This simulation environment provides the user with a feel for the general functionality of the design. The simulation environment shows basic traffic movement end-to-end.

Overview

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 (see Figure 2-30). This simulation environment demonstrates the basic functionality of the TRD through various test cases. The out-of-box simulation environment demonstrates the end-to-end (in Loopback mode) data flow for Ethernet packet.

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 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

To speed up the simulation, Physical Interface for PCI Express (PIPE) mode simulation is used in the TRD. For more details on PIPE mode simulation, refer to *7 Series FPGAs Integrated Block for PCI Express v2.2 Product* Guide (PG054) [Ref 3].

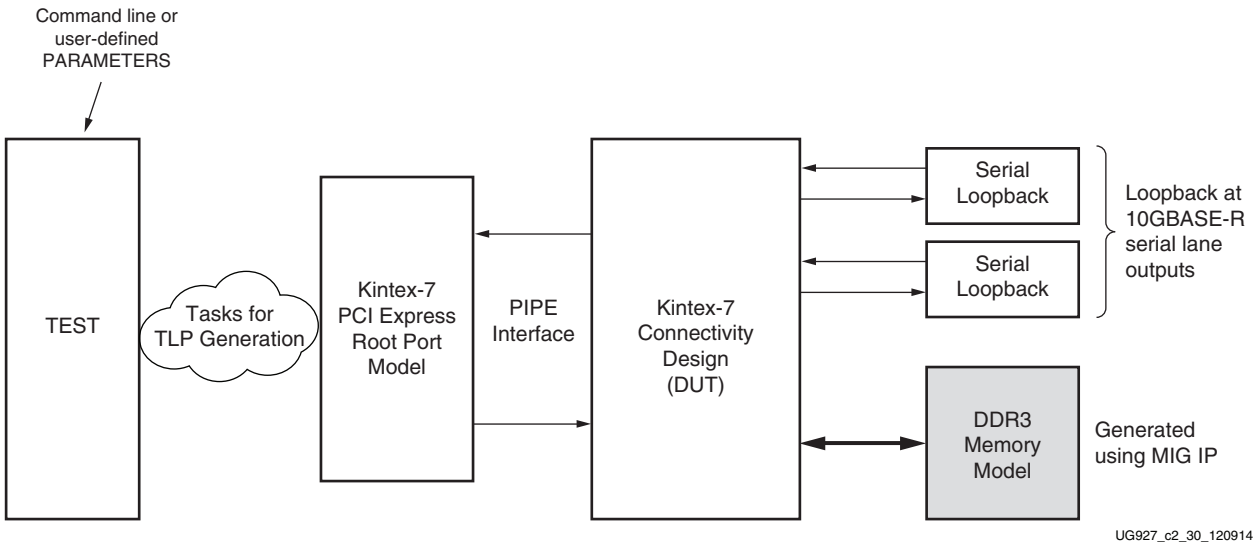


Figure 2-30: **Out-of-Box Simulation Overview**

Simulating the Design

Questa Simulation

To run the simulation, follow the steps outlined below.

1. When using QuestaSim, be sure to compile the required libraries and set the environment variables as required before running the script. Refer to *Synthesis and Simulation Design Guide* (UG626) [Ref 4], and *Vivado Design Suite Logic Simulation User Guide* (UG900) [Ref 5], which provide information on how to run simulations with different simulators.
2. Execute the **`k7_conn_trd_ipi_mti.tcl`** command under the `k7_connectivity_trd/hardware/vivado/scripts` directory.
3. After the QuestaSim GUI opens, run this command:

   `run -all`

Vivado Simulation (XSIM)

To run the XSIM simulation, follow the steps outlined below.

1. Set the environment variables required to setup the Vivado simulator simulation. Refer to UG900, Vivado Design Suite User Guide which provides information on how to run simulation with different simulators.
2. Execute the **`vivado -source k7_conn_trd_ipi_xsim.tcl`** command under the `k7_connectivity_trd/hardware/vivado/scripts` directory.
3. After the Vivado GUI is open, **Click Run Simulation → Run Behavioral Simulation** option.

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 2-1: **User-Controlled Macro Descriptions**

Macro Name	Default Value	Description
CH0	Defined	Enables Channel 0 initialization and traffic flow.
CH1	Defined	Enables Channel 1 initialization and traffic flow.
DETAILED_LOG	Not Defined	Enables a detailed log of each transaction.

Table 2-2: **Macro Description for Design Change**

Macro Name	Description
DMA_LOOPBACK	Connects the design in Loopback mode at DMA user ports – no other macro should be defined.

Test Selection

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 two 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 two 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. In order to run this test, only one channel (either CH0 or CH1) should be enabled in include/`user_defines.v`
dma_disable	DMA Disable Test This test shows the DMA disable operation sequence on a DMA channel.
pcie_link_change	PCIe Link Width & Speed Change Test This test changes the PCIe link from x8 GEN2 to x4 GEN1 and runs the test. This demonstrates how the demand driver power management concept can be exercised by changing the PCIe link configuration on the fly.

The name of the test to be run can be specified in the `k7_conn_trd_mti.tcl` and `k7_conn_trd_xsim.tcl` scripts. By default, the simulation script file specifies the basic test with the string: **`TESTNAME=basic_test`**.

The test selection can be changed by specifying a different test case, as specified above.

XILINX

Chapter 3

Functional Description

This chapter describes the hardware and software architecture in detail.

Hardware Architecture

The hardware design architecture is described under the following sections:

- Base System Components: Describes PCIe-DMA and the DDR3 virtual FIFO components
- Application Components: Describes the user application design
- Utility Components: Describes the power monitor block, the PCIe link width and speed change module etc.
- Register Interface: Describes the control path of the design
- Clocking and Reset

Base System Components

PCI Express® is a high-speed serial protocol that allows transfer of data between host system memory 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 v2.1 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.

This TRD uses PCIe in x8 GEN2 configuration with credits/buffering enabled for high performance bus mastering applications.

For details on the Kintex-7 FPGA Integrated Block for PCI Express, refer to *7 Series FPGAs Integrated Block for PCI Express User Guide* (PG054) [Ref 3].

Performance Monitor for PCI Express

This monitor snoops on the AXI4-Stream PCIe 128-bit interface operating at 250 MHz and provides the following measurements which are updated once every second:

- Count of active beats upstream which include the TLP headers for various transactions
- Count of active beats downstream which include the TLP headers for various transactions
- Count of payload bytes for upstream memory write transactions – this includes buffer write (in C2S) and buffer descriptor updates (for both S2C and C2S)
- Count of payload bytes for downstream completion with data transactions – this includes buffer fetch (in S2C) and buffer descriptor fetch (for both S2C and C2S)

These performance numbers measured are reflected in user space registers which software can read periodically and display.

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[127:0]	Input	Data to be transmitted via PCIe link.
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]	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[127:0]	Input	Data received on the PCIe link. Valid only if m_axis_rx_tvalid is also asserted.
m_axis_rx_tlast	Input	End of frame indicator for received packet. Valid only if m_axis_rx_tvalid is also asserted.
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.

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

Port Name	Type	Description
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 about 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 DMA documentation, available in `k7_connectivity_trd/hardware/sources/ip_cores/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. The packet DMA is configured to support simultaneous operation of two user applications utilizing four channels in all. 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 A, Register Description.

The front end of DMA interfaces to the AXI4-Stream interface on PCIe Endpoint IP core. The back end of the DMA provides an AXI4-Stream interface as well which connects to the user.

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 8 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 3-1 shows the buffer descriptor layout for S2C and C2S directions.

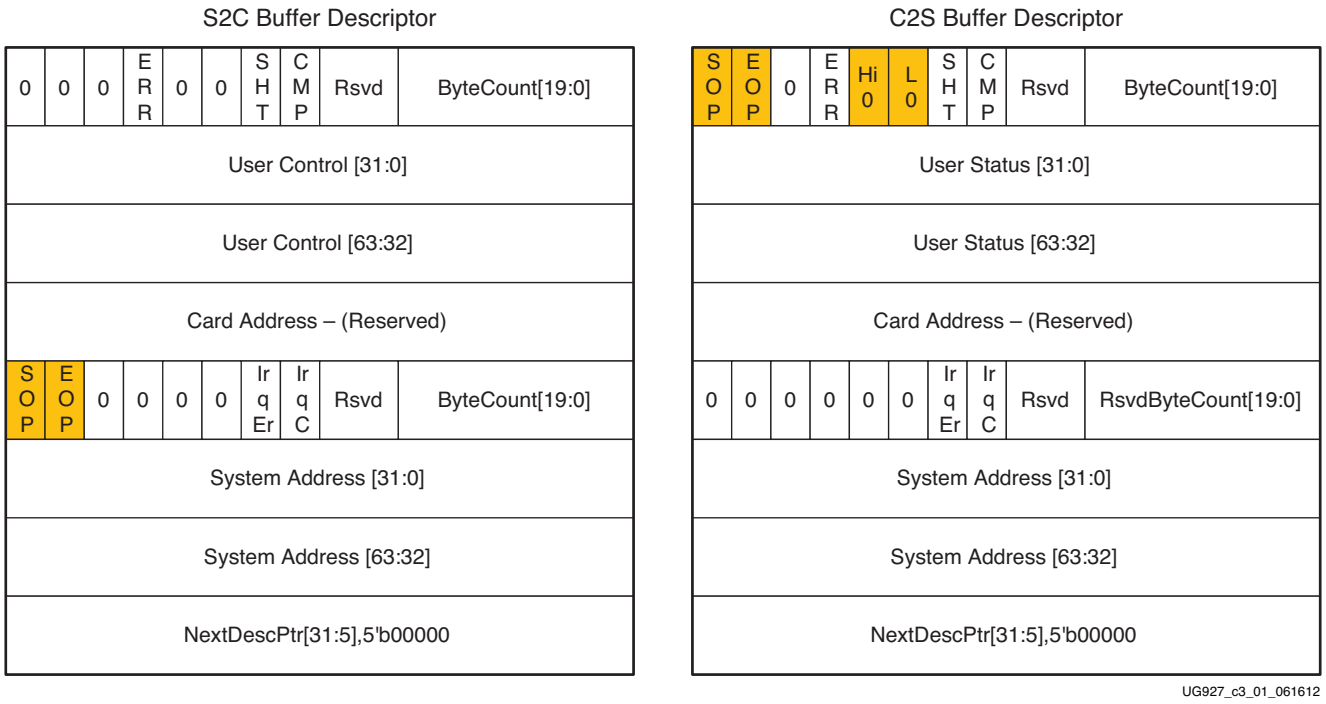


Figure 3-1: **Buffer Descriptor Layout**

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.
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 instructs DMA to issue an interrupt when the descriptor results in error.
Irq C	Interrupt on Completion This bit instructs DMA to issue an interrupt when operation associated with the descriptor is completed.
ByteCount[19:0]	Byte Count In S2C direction, indicates 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.

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

Descriptor Fields	Functional Description
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 reserved for Packet DMA.
System Address	System Address This defines the system memory address from which 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.

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, DMA fetches the descriptor followed by the data buffer to which it points. 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 3-2 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 next source valid after end of packet or tlast indicates start of packet.

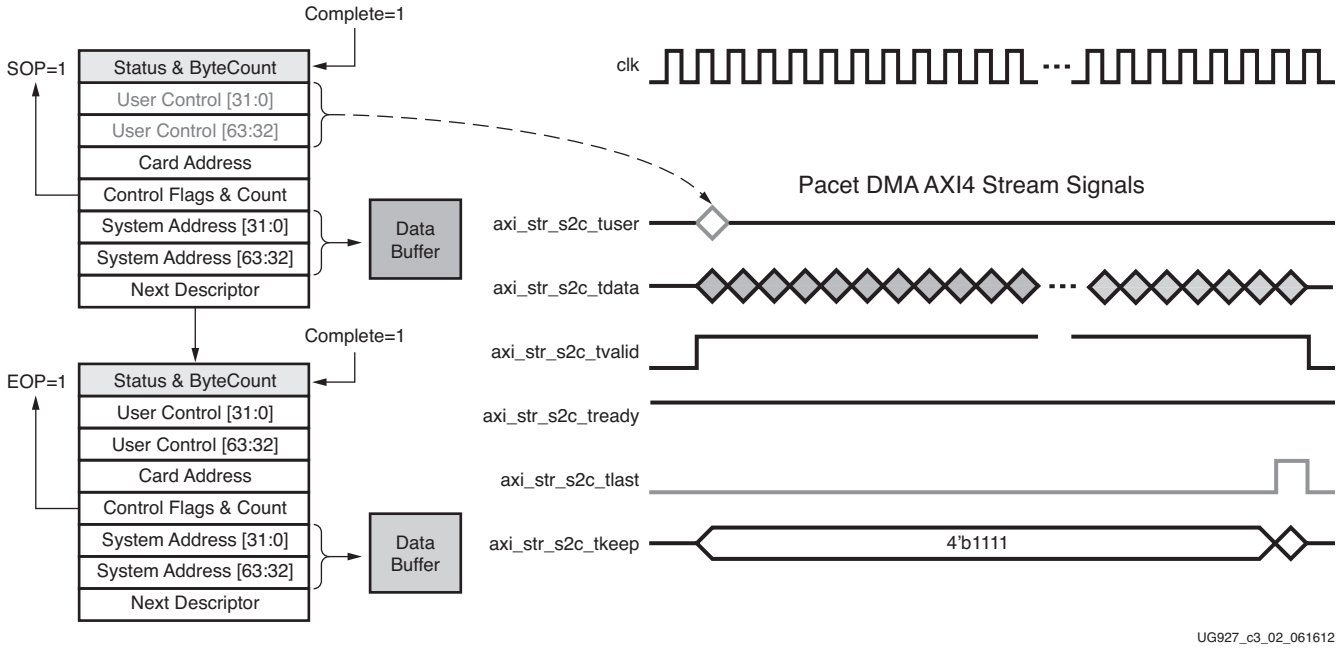


Figure 3-2: **Data Transfer from System to Card**

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, 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, 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 3-3 shows the card-to-system- 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 being asserted indicates start of a new frame.

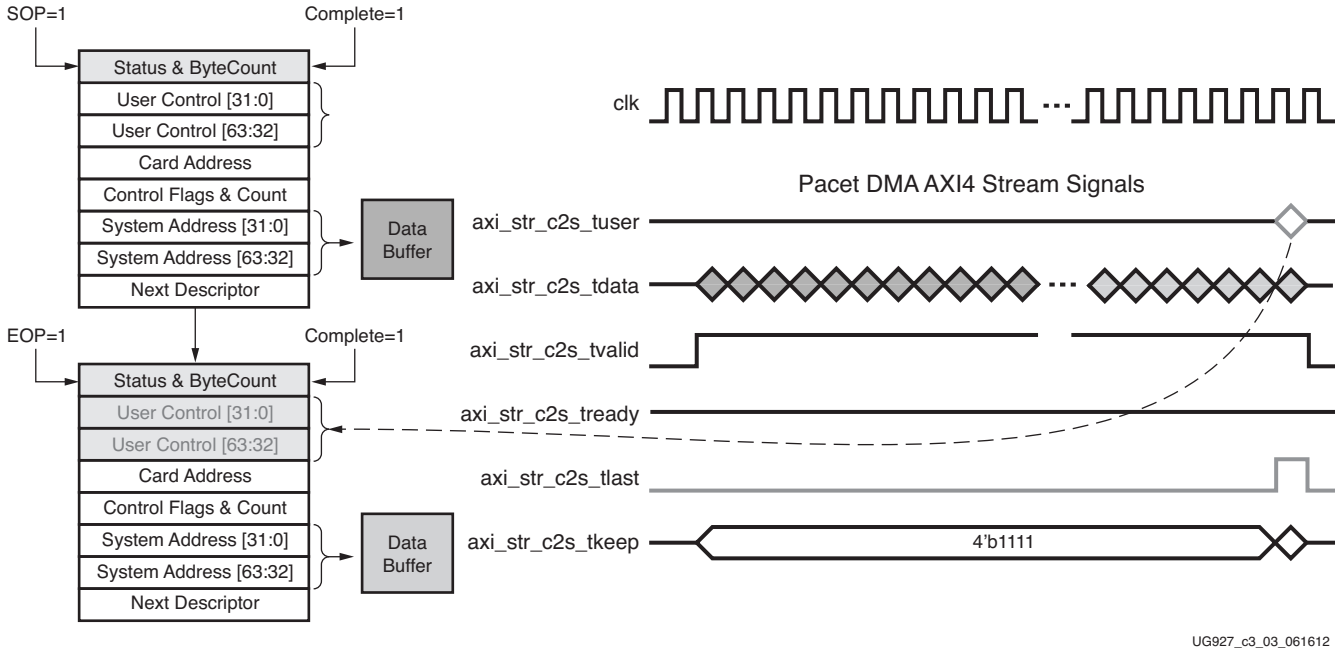


Figure 3-3: **Data Transfer from Card to System**

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 TRD uses DDR3 space as multiple FIFO for storage. It achieves this by use of following IP cores:

1. AXI Stream Interconnect, in 4x1 and 1x4 fashion and also used for width conversion and clock domain crossing
2. AXI VFIFO Controller, 4 channels used for interfacing stream interface to AXI-MM provided by MIG and also handles the addressing needs for DDR3 FIFO
3. MIG, which provides the DDR3 memory controller for interfacing to external SODIMM

Figure 3-3 shows the connection of these IPs to form a multiport virtual packet FIFO.

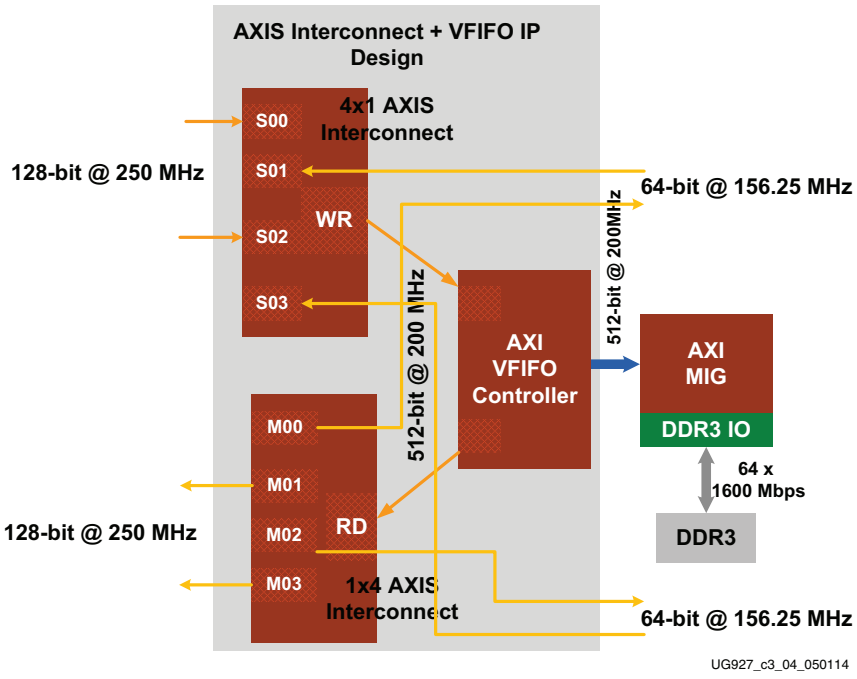


Figure 3-4: **Virtual FIFO Based on AXIS-IC and AXI-VFIFO IP**

AXI Stream Interconnect

The AXI4 stream interconnect provides the following:

1. Multiplexes four write channels to one AXI4 stream for AXI-VFIFO and demultiplexes one read channel from AXI-VFIFO to four read channels based on the tdest field.
2. Provides packet mode FIFO support on read interface connecting to XGEMAC to enable a frame transmission without any pause in between
3. Width and clock conversion
   a. 128-bit @ 250 MHz from DMA S2C interface and 64-bit @ 156.25 MHz from XGEMAC-RX interface to 512-bit @ 200 MHz to AXI-VFIFO interface on writes
   b. 512-bit @ 200 MHz from AXI-VFIFO interface to 128-bit @ 250 MHz to DMA interface and 64-bit @ 156.25 MHz to XGEMAC-TX interface on reads
4. Buffer for storage in order to avoid frequent back-pressure to PCIe-DMA

Further information on this IP can be obtained from *LogiCORE IP AXI4-Stream Interconnect Product Guide* (PG035) [Ref 13].

AXI VFIFO Controller

This Virtual FIFO controller manages the DDR3 address space for FIFO mode of operation for four channels. This block operates 512-bits at a 200 MHz clock across the AXI4-MM interface for the MIG controller.

Further information on this IP can be obtained from *LogiCORE IP AXI Virtual FIFO Controller Product Guide* (PG038) [Ref 14].

Application Components

The application components are described under the following sections:

- AXI4 Stream Packet Generator and Checker Interface
- Network Path Components (describes XGEMAC, 10GBASE-R PHY, and associated logic)

AXI4 Stream Packet Generator and Checker Interface

The traffic generator and checker interface follows AXI4 stream protocol. The packet length is configurable through control interface. Refer to Performance Mode: Generator/Checker/Loopback Registers for User APP 0, page 110 for details on registers.

The traffic generator and checker module can be used in three different modes: a Loopback mode, a Data Checker mode, and a Data Generator mode. The module enables specific functions depending on the configuration options selected by the user (which are programmed through control interface to user space registers). 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 from the host system can be looped back to itself. Based on user inputs, the software driver programs user space registers to enable checker, generator, or loopback mode of operation.

If the Enable Loopback bit is set, the transmit data from DMA in the S2C direction is looped back to receive data in the C2S direction. In the Loopback mode, data is not verified by the checker. Hardware generator and checker modules are enabled if the Enable Generator and Enable Checker bits are set via software.

The data received and transmitted by the module is divided into packets. The first two bytes of each packet define the length of the packet. All other bytes carry the tag, which is the sequence number of the packet. The tag increases by one per packet. Table 3-3 shows the pre-decided packet format used.

Table 3-3: **Packet Format**

[127:120] [119:112]	[111:104] [103:96]	[95:88] [87:80]	[79:72] [71:64]	[63:56] [55:48]	[47:40] [39:32]	[31:24] [23:16]	[15:8] [7:0]
TAG	TAG	TAG	TAG	TAG	TAG	TAG	PKT_LEN
TAG	TAG	TAG	TAG	TAG	TAG	TAG	TAG
TAG	TAG	TAG	TAG	TAG	TAG	TAG	TAG
--	--	--	--	--	--	--	--
--	--	--	--	--	--	--	--
TAG	TAG	TAG	TAG	TAG	TAG	TAG	TAG

The tag or sequence number is 2-bytes long. The least significant 2 bytes of every start of a new packet is formatted with packet length information. Remaining bytes are formatted with a sequence number which is unique per packet. The subsequent packets have an incremental sequence number.

The software driver can also define the wrap around value for the sequence number through a user space register.

Packet Checker

If the Enable Checker bit is set (Registers as defined in Appendix-X), as soon as data is valid on the DMA transmit channels (namely S2C0 and S2C1) each data byte received is checked against a pre-decided data pattern. If there is a mismatch during a comparison, the data_mismatch signal is asserted. This status is reflected back in register which can be read through control plane.

Packet Generator

If the Enable Generator bit is set (Register as defined in Appendix-X) and the data produced by the generator is passed to the receive channel of the DMA (namely C2S0 and C2S1). The data from the generator also follows the same pre-decided data pattern as the packet checker.

Network Path Components

A network interface card (NIC) is a device used to connect computers to a local area network (LAN). The software driver interfaces to the networking stack (or the TCP-IP stack) and the Ethernet frames are transferred between system memory and Ethernet MAC in hardware using the PCIe interface.

The XGEMAC block connects to 10GBASE-R IP through the ten gigabit media independent interface (XGMII) operating at 156.25 MHz clock. The XGMII is a 64-bit wide single data rate (SDR).

The XGEMAC IP requires interface logic to support AXI-ST compliant flow control. The following sections describe the custom IP blocks that implement the flow control logic for the XGEMAC block.

For details on ten gigabit Ethernet MAC and 10 gigabit PCS-PMA IP cores, refer to PG072. *LogiCORE IP 10-Gigabit Ethernet MAC Product Guide* (PG072) [Ref 6] and *LogiCORE IP 10-Gigabit Ethernet PCS/PMA Product Guide* (PG068) [Ref 7], respectively.

Note on Dual 10GBASE-R Implementation

The design optimizes the clocking resource used in two GT instances corresponding to the 10GBASE-R core by sharing the following:

1. Transmit user clock sharing for GTs belonging to same quad
2. Transmit user clock and GT reference clock sharing for GTs belonging to different quad

The 10GBASE-R IP uses two GTs from quad 118; reference clock 0 for quad 118 is sourced from the FMC card and all clock nets required for the IP are derived from this reference clock.

The receive clock output from the GT cannot be shared across multiple GTs as these clocks are out of phase. In the transmit direction, the phase mismatch between clocks in the PCS and PMA domain is taken care of by the use of the transmit FIFO in the GT.

The reference clock frequency for GTs used in the 10GBASE-R IP is 312.5 MHz sourced from the FMC card connected to the KC705 board. The output clock from the GT is divided by 2 using an MMCM to generate the 156.25 MHz clock which is supplied to the 10GBASE-R core.

Transmit Path

The transmit interface logic does the following:

- Reads packets from the virtual packet FIFO and provides them to the XGEMAC transmit interface after relevant width conversion
- Ensures that there is no pause for the packet under transmission

Figure 3-5 represents the block diagram of the transmit interface logic. The datapath from the virtual packet FIFO is 128 bits wide. The data width down converter converts the 128-bit wide data to 64-bit wide data, required for the XGEMAC transmit interface. The Tx interface logic block controls the valid signal to the XGEMAC based on the data available in the virtual packet FIFO, ensuring continuous flow of data (for that packet) once a packet transmission has started.

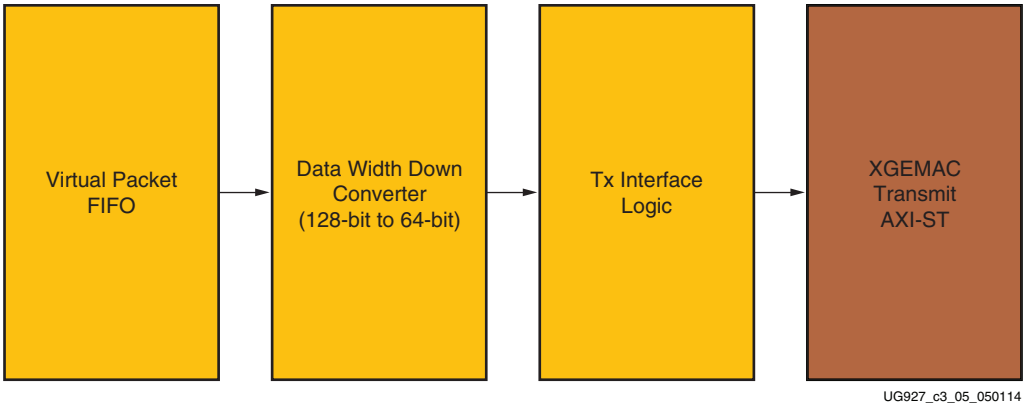


Figure 3-5: **Transmit Interface Block Diagram**

Data Width Down Converter

The data width down converter module converts 128-bit data from packet buffer to 64-bit data. The converter works in the 156.25 MHz clock domain. It reads one cycle of 128-bit data from the FIFO and sends two cycles each of 64-bit data to the XGEMAC. This is achieved by handling the read from the FIFO appropriately i.e., reading every alternate cycle instead of reading continuously.

Transmit Interface Logic

The transmit interface logic monitors the data count in the packet FIFO from its read data count field and once the count indicates that the entire packet is available in packet FIFO, asserts *ready to packet buffer* in order to read the packet stored in the packet buffer and also *valid* to the XGEMAC-TX to begin data transmission. This logic assures that once a packet transmission has begun, it ends without any pause in between to comply with XGEMAC-TX interface requirements.

Receive Path

The receive interface logic does the following:

- Receives incoming frames from the XGEMAC and performs address filtering (if enabled to do so)
- Based on packet status provided by the XGEMAC-RX interface, decides whether to drop a packet or pass it ahead to the system for further processing

Figure 3-6 represents the block diagram of the receive interface logic.

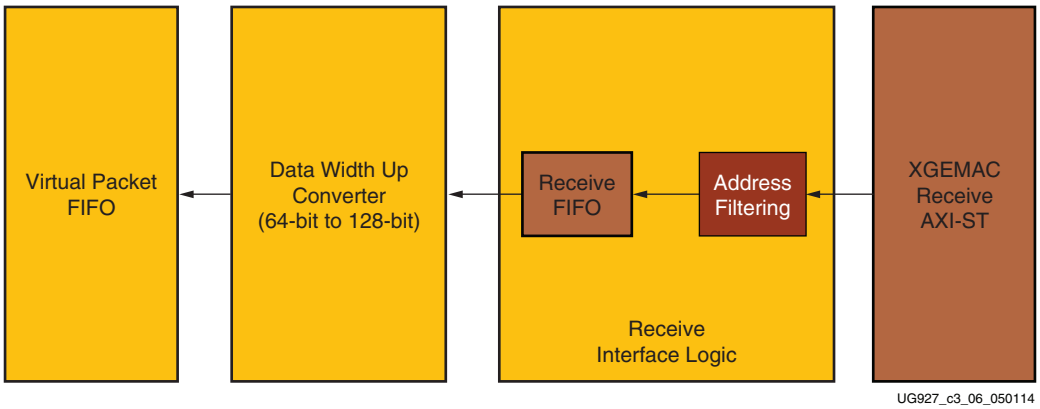


Figure 3-6: **Receive Interface Block Diagram**

Receive Interface Logic

The XGEMAC-RX interface does not allow back-pressure i.e., once a packet reception has started it completes the entire packet. The receive interface logic stores the incoming frame in a local receive FIFO. This FIFO stores the data until it receives the entire frame. If the frame is received without any error (indicated by tlast and tuser from the XGEMAC-RX interface), it is passed ahead, otherwise it is dropped. The Ethernet packet length is read from the receive statistics vector instead of implementing a separate counter in logic. This limits the upper bound on packet length to be 16,383B as supported by the receive statistics packet count vector in the XGEMAC IP.

The depth of the FIFO in the receive interface logic is decided based on the maximum length of the frame to be buffered and the potential back pressure imposed by the packet buffer. The possible scenario of FIFO overflow occurs when the received frames are not drained out at the required rate in which case receive interface logic drops Ethernet frames. The logic also takes care of clean drop of entire packets due to this local FIFO overflowing.

Address Filtering

Address filtering logic filters out a specific packet which is output from the XGEMAC receive interface if the destination address of the packet does not match with the programmed MAC address. MAC address can be programmed by software using the register interface.

Address filtering logic:

- Performs address filtering on-the-fly based on the MAC address programmed by software
- Allows broadcast frames to pass through
- Allows all frames to pass through when promiscuous mode is enabled

The receive interface state machine compares this address with the first 48 bits it receives from XGEMAC-RX interface during start of a new frame. If it finds a match it writes the packet to the receive FIFO in the receive interface, otherwise, the packet is dropped as it is comes out of the XGEMAC receive interface.

Data Width Up Converter

This module converts the 64-bit wide data received from the XGEMAC-RX interface to 128-bit wide data and sends the data for storage in the virtual FIFO. For every two cycles of data read from the receive FIFO, one cycle of data is written to the virtual FIFO.

Utility Components

The utility components are described under the following sections.

- PicoBlaze Based Power Monitor
- Application Demand Driven Power Management

PicoBlaze Based Power Monitor

The TRD uses PicoBlaze based power monitoring logic to monitor power consumed by the FPGA on various voltage rails and the die temperature. The logic interfaces with the built in Xilinx analog to digital converter (XADC) to read the die temperature. In order to read voltage and current values of different voltage rails in the FPGA, the power monitoring logic interfaces with TI's power regulators (UCD9248) present on KC705 board. Communication with the power regulator (UCD9248) occurs using the standard PMBus (power management bus) interface.

Figure 3-7 represents the block diagram of the power monitoring logic. PicoBlaze is a light weight soft core processor targeting Xilinx FPGAs. The PicoBlaze processor manages the communication with UCD9248 using PMBus protocol. The XADC acts as a second peripheral to PicoBlaze. Once voltage and current values are read from on board regulators, PicoBlaze calculates the power values and updates the specified block RAM locations (block RAM is used as a register array). Block RAM locations are read periodically by a custom user logic block and are accessible to user through the control plane interface.

The register interface interacts with the read logic block. Power and temperature numbers are read periodically from block RAM locations by the software using the DMA backend interface. The GUI displays V_{CCINT}, V_{CCAUX} and V_{CCBRAM} power. User can read VCC 3.3V, VCC 2.5V, VCC 1.5V, VADJ, MGT_AVCC, MGT_AVTT, MGT_VCCAUX, VCCAUX_IO power values by enabling DEBUG_VERBOSE flag in the makefile provided in the xdma driver subdirectory.

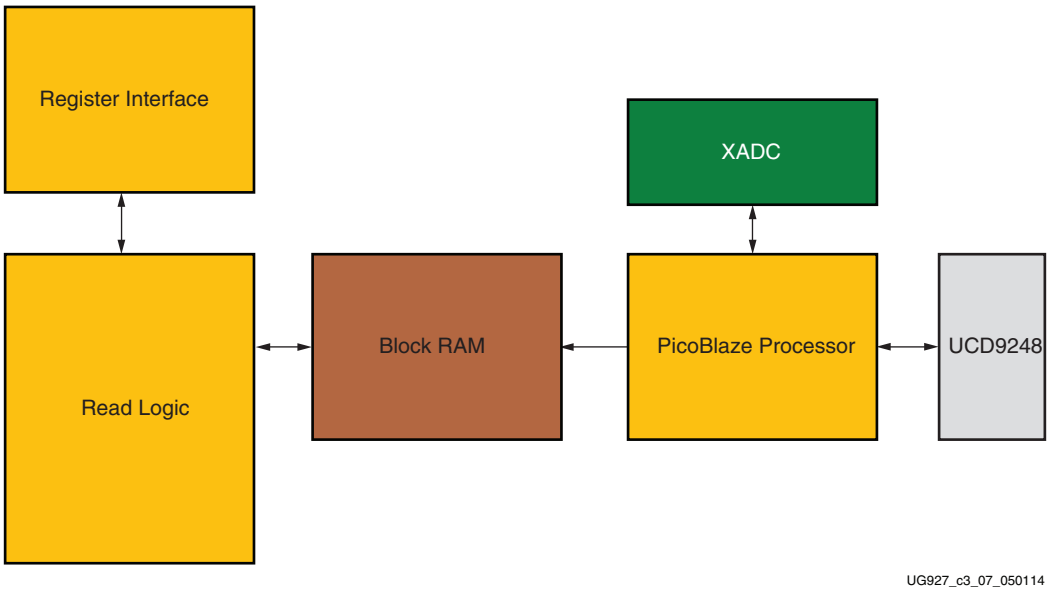


Figure 3-7: **Power Monitor Logic Block Overview**

Application Demand Driven Power Management

Based on application traffic demand, PCIe link width and speed can be down configured to the smallest values to save power. On full traffic resumption, this can be reversed by up configuring the link speed and width.

This directed change is implemented in hardware and control is provided to software through registers.

Hardware advertises its capability through registers. Software, on reading the capability registers, drives the control register appropriately. Based on further control validation checks in hardware, relevant ports on PCIe block are asserted and the result of operation is indicated back in the status register.

Note: Link width/speed change operations can be initiated only when the link is up and in the L0 state, and the device is in the D0 state.

The next two sections provide a brief summary of directed link width and link speed change algorithms independently. However, these operations can also be done together.

Software can check the capability register and issue a *target_link_width* or *target_link_speed* based on validity checks. As a double check, the hardware also implements the validity checks to make sure the ports on the PCIe core are not put into any controversial state.

Link Width Change Scheme

The following summarizes the steps for directed link width change. *target_link_width* is the width driven by the application. *pl_sel_link_width* is the current width indicated as output port by the PCIe core.

1. Check that the link is up and *pl_ltssm_state* = L0.
2. If (*target_link_width* ! = *pl_sel_link_width*), proceed with width change. This makes sure that the target width and current width are not equal.

3. Check the validity of the request:
   a. If (*pl_link_upcfg_capable* = `1`), and (*target_link_width* <= p*l_initial_link_width*), then proceed, otherwise abort.
   b. If (*pl_link_upcfg_capable* = `0`), and (*target_link_width* < *pl_sel_link_width*), then proceed, otherwise abort.
4. Assign *pl_directed_link_width* = *target_link_width* and *pl_directed_link_change[0]* = `1`.
5. Wait until (*pl_ltssm_state* == *Configuration.Idle*) or (*link_up* = `0`).
6. Assign *pl_directed_link_change[0]* = `0`.
7. Update the status register.

Link Speed Change Scheme

The following summarizes the steps for directed link speed change operation. *target_link_speed* is the speed driven by the application. *pl_sel_link_speed* is the current speed indicated as output port by the PCIe core.

1. Check that link is up and *pl_ltssm_state* = L0.
2. If (*target_link_speed ! = pl_sel_link_speed*), proceed with speed change. This makes sure that the target speed and current speed are not equal.
3. Check the validity of the request:
   a. If current link speed is 2.5 Gb/s ensure that *pl_linkgen2_capable* and *pl_linkpartner_gen2_capable* are asserted.
4. Assign *pl_directed_link_speed* = *target_link_speed* and *pl_directed_link_change[1]* = `1`.
5. Wait until (*pl_ltssm_state* == *Recovery.Idle*) or (*link_up* = `0`).
6. Assign *pl_directed_link_change[1]* = `0`.
7. Update the status register.

Register Interface

DMA provides the AXI4 target interface for user space registers. Register address offsets from `0x0000` to `0x7FFF` on BAR0 are consumed internally by the DMA engine. Address offset space on BAR0 from `0x8000` to `0xFFFF` is provided to user. Transactions targeting this address range are made available on the AXI4 target interface.

The design has the following control interfaces:

a. XGEMAC registers – one set for each instance

b. User space registers defining design mode configuration, control and status

AXI4LITE Interconnect is used to fan out the AXI4 target interface to the appropriate slave address region as defined in Figure 3-8.

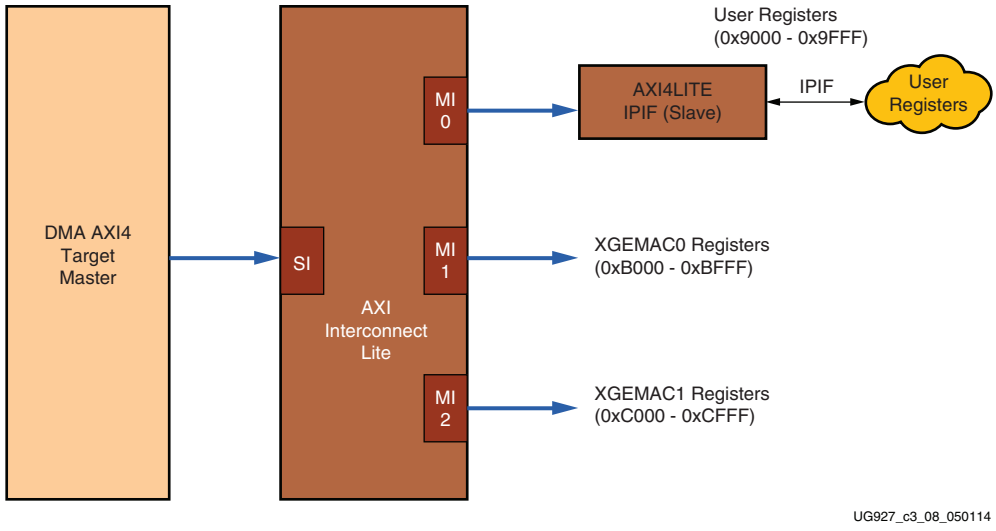


Figure 3-8: **Register Interface**

Details of user registers are provided in User Space Registers, page 105. XGEMAC registers are defined in the *LogiCORE IP 10-Gigabit Ethernet MAC User Guide* (PG072) [Ref 6]. The XGEMAC provides an MDIO interface for accessing registers of the attached PHY. In the design, 10G BASE-R PHY registers are accessed through the XGEMAC MDIO interface.

Clocking and Reset

This section describes the clocking and reset scheme of the design.

Clocking Scheme

The design uses the following clocks from the external world:

- 100 MHz differential PCIe reference clock from the motherboard PCIe slot
- 200 MHz differential clock from the on-board source for the MIG IP
- 312.5 MHz differential clock from the clock source on the FMC for 10GBASE-R IP

Figure 3-9 summarizes the various clock domains of this design.

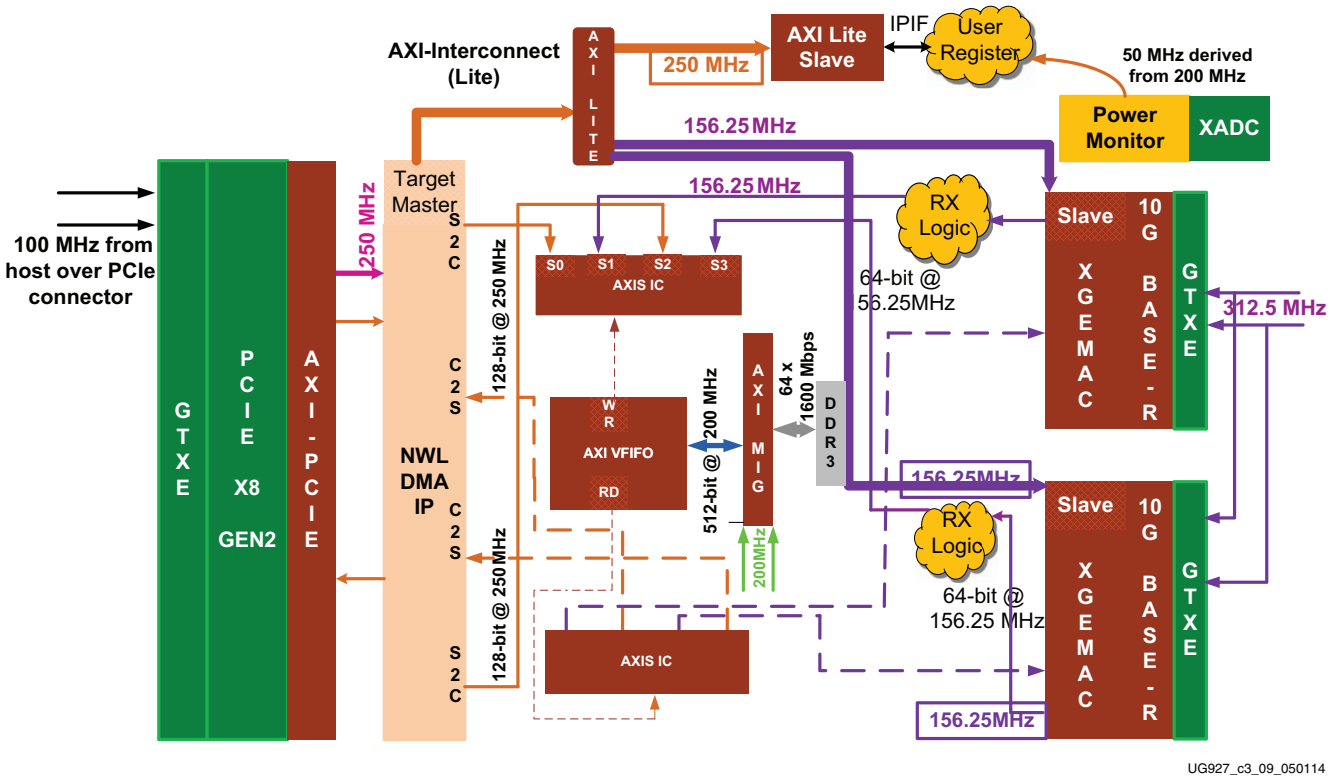


Figure 3-9: **Clocking Scheme**

Reset Scheme

The design uses only one external hard reset – PERST# provided by the motherboard through PCIe slot. This also resets the memory controller and the 10G PHYs apart from resetting all other design components. In addition, various soft resets are provided as listed in Table 3-4.

Table 3-4: **Resets**

Module	PERST#	PCIe Link Down	DDR3 Calibration	10G PHY Link	Soft Resets
PCIe Wrapper	X				
DMA	X	X			X
DDR3 Memory Controller	X	X	X		
AXI Interconnect	X	X	X		X
AXI4LITE Interconnect	X	X	X	X	
Ten Gig Ethernet MAC	X	X		X	
10G BASE-R PHY	X	X		X	

Table 3-4: **Resets *(Cont'd)***

Module	PERST#	PCIe Link Down	DDR3 Calibration	10G PHY Link	Soft Resets
AXI Virtual FIFO IP	X	X		X	X
Packet Generator/ Checker	X	X			X
Power Monitor	X	X			

PERSTN or PCIe Link down is the master reset for everything. The PCIe wrapper, memory controller, and 10GBASE-R PHY get PERSTN directly. These blocks have higher initialization latency hence these are not reset under any other condition. Once initialized, PCIe asserts *user_lnk_up*, the memory controller asserts *calib_done*, and the 10G PHY asserts *block_lock* (bit position zero in the status vector).

The DMA provides per channel soft resets which are also connected to the appropriate user logic. Additionally, to reset only the AXI wrapper in the MIG and AXI-Interconnect, another soft reset via a user space register is provided. However, this reset is to be asserted only when the DDR3 FIFO is empty and there is no data lying in FIFO or in transit in FIFO.

Linux Device Driver and Application

The software component of the Kintex-7 Connectivity TRD comprises one or more Linux kernel-space driver modules with one user-space application that controls design operation. The software building blocks are designed with scalability in mind. It enables a user to add more user-space applications to the existing infrastructure.

The software has been designed to meet the following requirements:

- Ability to source application data at very high rates to showcase the performance capabilities of the hardware design.
- Effectively showcase the use of multi-channel DMA to support different applications.
- Provide a user interface that is easy to use and is intuitive.
- Provide a modular design which is extensible, reusable, and can be customized.

The feature list of the user application and Linux kernel-space drivers that enables the above requirements to be met are as follows.

User-space Application Features

The user-space application GUI provides the following features:

- GUI management of the driver and device – for configuration control, and for status display
- GUI front-end for a graphical display of performance statistics collected at the PCIe transaction interface, DMA engine, and kernel level
- In Performance mode the GUI also spawns a multi-threaded application traffic generator which generates and receives data

For control of Ethernet specific features, standard Linux tools should be used as described in Ethernet Specific Features, page 38.

Kernel-space Driver Features

- Configuration of the DMA engine . to achieve data transfer between the hardware and host system memory.
- Transfer of Ethernet packets from Linux TCP/IP stack to network path in hardware for transmission into the LAN and from network path in hardware to Linux TCP/IP stack for handling by networking applications. This is the Ethernet data flow.

Data Flow Model

This section provides an overview of the data flow in both software and hardware.

Application (Ethernet) Data Flow

Figure 3-10 illustrates the Ethernet data flow. On the transmit path, data from the networking application (for example, an internet browser) is packetized in the TCP/IP stack, converted into Ethernet frames, and handed over to the driver for transmission. The Ethernet driver then queues up the packet for scatter gather DMA in the TRD. The DMA fetches the packet through the PCIe Endpoint and transfers it to the XGEMAC where it is transmitted through the Ethernet link to the LAN.

On the receive side, packets received by the XGEMAC are pushed to scatter gather DMA. The DMA in turn pushes the packet to the driver through the PCIe Endpoint. The driver hands off the packet to the upper layers for further processing.

In this mode, the user starts the test through the GUI. The GUI also displays the live performance statistics for the test.

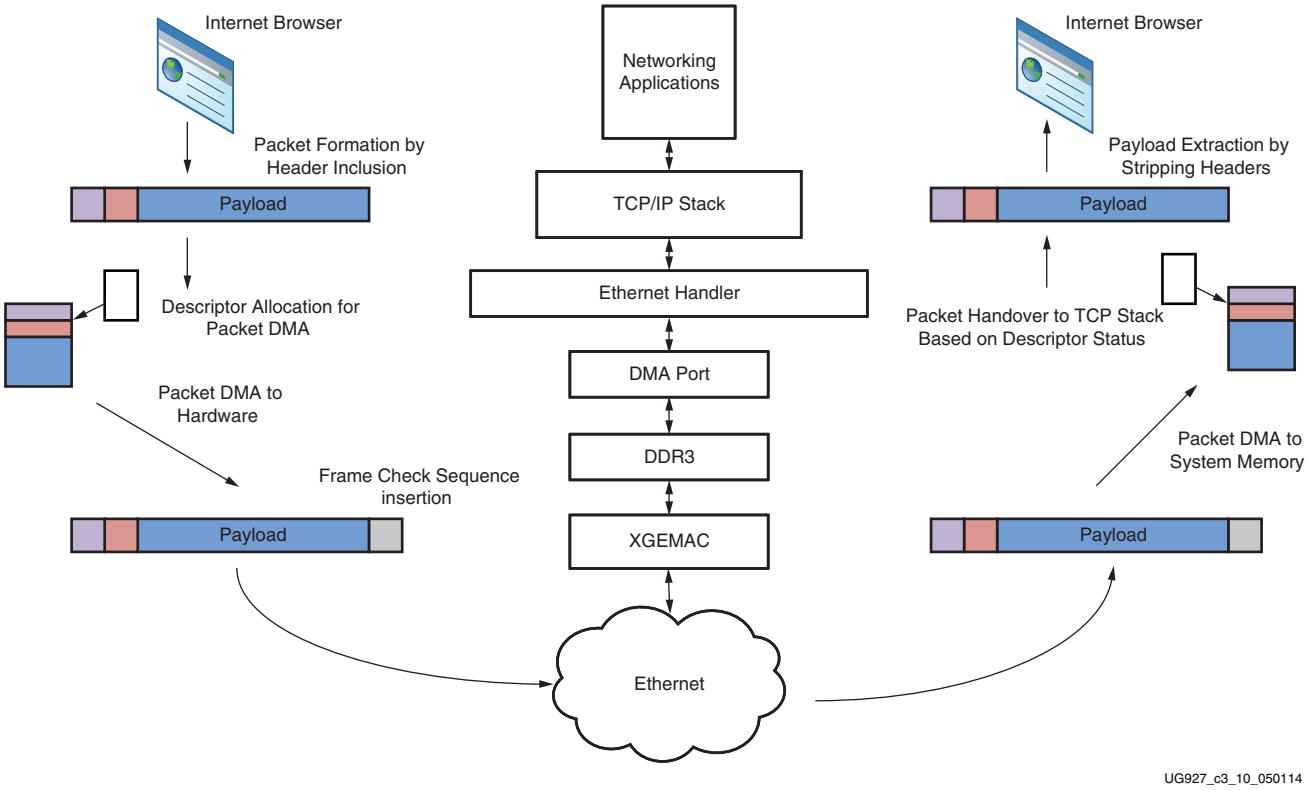


Figure 3-10: **Ethernet Data Flow**

Performance Mode Data Flow

Figure 3-11 illustrates the data flow in Performance mode. On the transmit side, the GUI spawns multiple threads (application traffic generator) according to the mode selected. The data buffers are generated in the application traffic generator passed to the driver and queued up for transmission in the host system memory. The scatter gather DMA fetches the packets through the PCIe Endpoint and transfers them to the Virtual FIFO. In raw Ethernet mode data written to the DDR3 is read and sent to the XGEMAC; data received is then again stored in DDR3 and transferred back to the DMA creating a loopback scenario. On the receive side, DMA pushes the packets to the software driver through the PCIe Endpoint. The driver receives the packets and pushes them to a software queue. The application traffic generator polls the queue periodically and verifies the data.

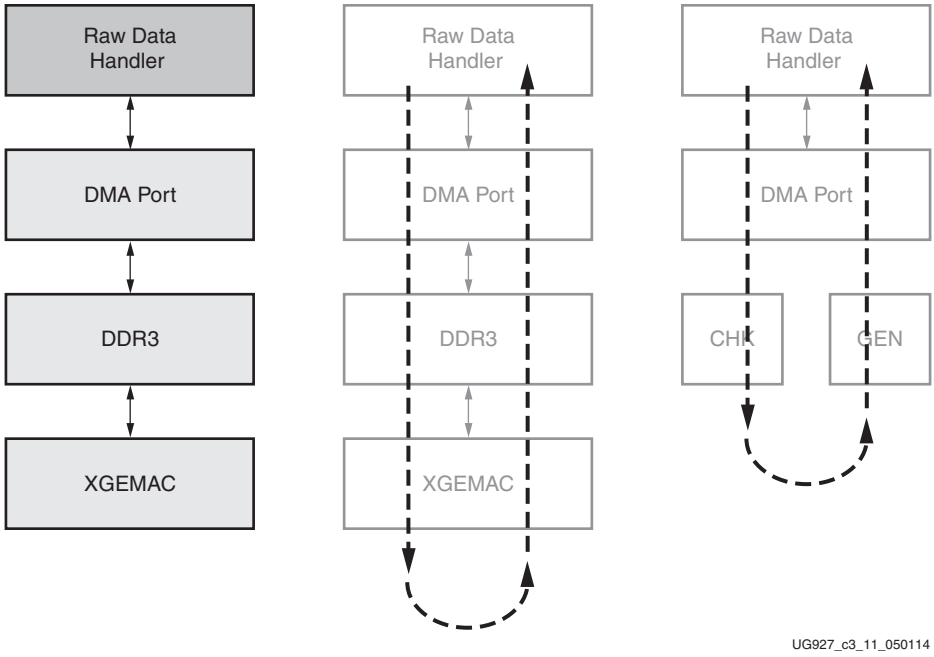


Figure 3-11: **Performance Mode Data Flow**

In a typical use scenario, the user starts the test through the GUI. The GUI displays the performance statistics collected during the test until the user stops the test.

Software Architecture

The software for the Kintex-7 Connectivity TRD comprises several Linux kernel-space drivers and a user-space application. Traffic is generated from the user application. Format of data changes from raw data to raw Ethernet data modes. The following sections explain data and control path flow.

Performance Mode (Gen/Chk and Raw Ethernet Mode)

Figure 3-12 depicts the software driver components. The description is divided into data and control path components.

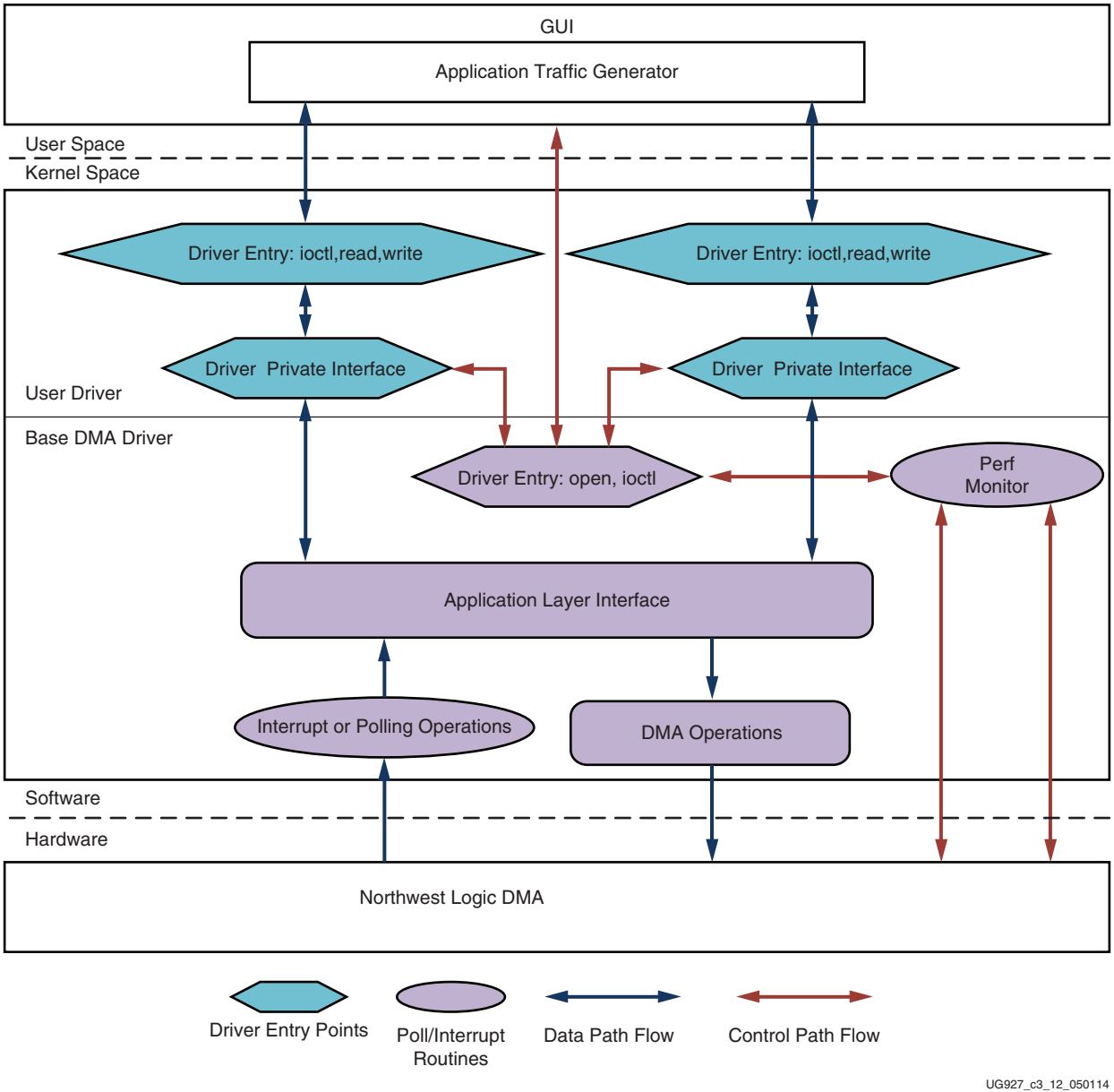


Figure 3-12: **Performance Mode Software Architecture**

Data Path Components

Application Specific Traffic Generator

This block generates the raw data or raw Ethernet data according to the mode selected in the user interface. The application opens the interface of the application driver through exposed driver entry points. The application transfers the data using read and write entry points provided by the application driver interface. the application traffic generator also performs the data integrity test in the receiver side, if enabled.

Driver Entry Point

This block creates a character driver interface and enhances different driver entry points for the user application. The driver entry point also enables sending of free user buffers for filling the DMA descriptor. Additionally, the driver entry point conveys completed transmit and receive buffers from the driver queue to the user application.

Driver Private Interface

This block enables interaction with the DMA driver through the private data structure interface. The data that comes from the user application through the driver entry points is sent to the DMA driver through the private driver interface. The private interface handles received data and housekeeping of completed transmit and receive buffers by putting them in a completed queue.

Application Driver Interface

This block is responsible for dynamic registering and unregistering of user application drivers. The data that is transmitted from the user application driver is sent over to the DMA operations block.

DMA Operations

For each DMA channel, the driver sets up a buffer descriptor ring. At test start, the receive ring (associated with a C2S channel) is fully populated with buffers meant to store incoming packets, and the entire receive ring is submitted for DMA while the transmit ring (associated with a S2C channel) is empty. As packets arrive at the base DMA driver for transmission, they are added to the buffer descriptor ring and submitted for DMA transfer.

Interrupt or Polling Operation

If interrupts are enabled (by setting the compile-time macro TH_BH_ISR), the interrupt service routine (ISR) handles interrupts from the DMA engine. 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 schedules the bottom half (BH) which invokes the functionality in the driver private interface pertaining to handling received data and housekeeping of completed transmit and receive buffers.

In polling mode, the driver registers a timer function which periodically polls the DMA descriptors. The poll function performs the following:

1. Housekeeping of completed transmit and receive buffer
2. Handling of received data

Control Path Components

Graphical User Interface

The control and monitor GUI is a graphical user interface tool used to monitor device status, run performance tests, configure PCIe link speed and width, monitor system power, and display statistics. It communicates the user-configured test parameters to the user traffic generator application which in turn generates traffic with the specified parameters. Performance statistics gathered during the test are periodically conveyed to the GUI through the base DMA driver for display as graphs.

When installed, the base DMA driver appears as a device table entry in Linux. The GUI uses the file-handling functions (open, close, and ioctl) on this device, to communicate with the driver. These calls result in the appropriate driver entry points being invoked.

Driver Entry Points

The DMA driver registers with the Linux kernel as a character driver to enable the GUI to interface with the DMA driver. The driver entry points allow conveying of application specific control information to the user application driver through the private interface.

A driver entry point also allows collecting and monitoring periodic statistical information from hardware by means of the performance monitor block.

Performance Monitor

The performance monitor is a handler that reads all the performance-related registers (PCIe link status, DMA Engine status, power monitoring parameters). Each of these parameters is read periodically at an interval of one second.

Performance Mode Design Implementation

This section provides an overview of software component implementation. Users are advised to refer to the driver code along with Doxygen generated documentation for further implementation details.

User Application

The user traffic generator is implemented with multiple threads. The traffic generator application spawns thread according to parameter and mode selected in the GUI. For transmit, two threads are needed, one for transmitting and one for transmitter done housekeeping. For receive, one thread provides free buffers for DMA descriptors and the other thread receives packets from the driver. The receive thread is also responsible for a data integrity check, if enabled in the GUI.

For one path two threads are needed for transmitting and two threads for receiving. On both paths eight threads are needed to run full traffic. Performance can be maximized if all of the threads are running on different CPUs. Any system having less than eight CPUs or any other application or kernel housekeeping affects the scheduling of the thread which intern affects performance. For running loopback or Gen/check on both paths, the threads are reduced which is achieved by combining housekeeping threads to single threads. A total of six threads are spawned for generating full traffic on both paths in the design.

To separate the application generator from the GUI, thread related functionality should be decoupled from GUI.

Driver implementation

Improved performance can be achieved by implementing zero copy. The user buffers address is translated into pages and mapped to PCI space for transmission to DMA. On the receive side packets received from DMA are stored in a queue which is then periodically polled by the user application thread for consumption.

Application Mode

This section describes the Ethernet Application mode (see Figure 3-13).

Control Path Components

Networking Tools

Unlike the raw data driver, the Ethernet functionality in the driver does not require the control and monitor GUI to be operational. Ethernet comes up with the prior configured settings. Standard Linux networking tools (for example, `ifconfig` and `ethtool`) can be

used by the system administrator when the configuration needs to be changed. The driver provides the necessary hooks which enable standard tools to communicate with it.

Graphical User Interface

Unlike the Performance mode, the GUI does not control test parameters and traffic generation in the Application mode. The GUI periodically polls and updates the various statistics through DMA driver entry points.

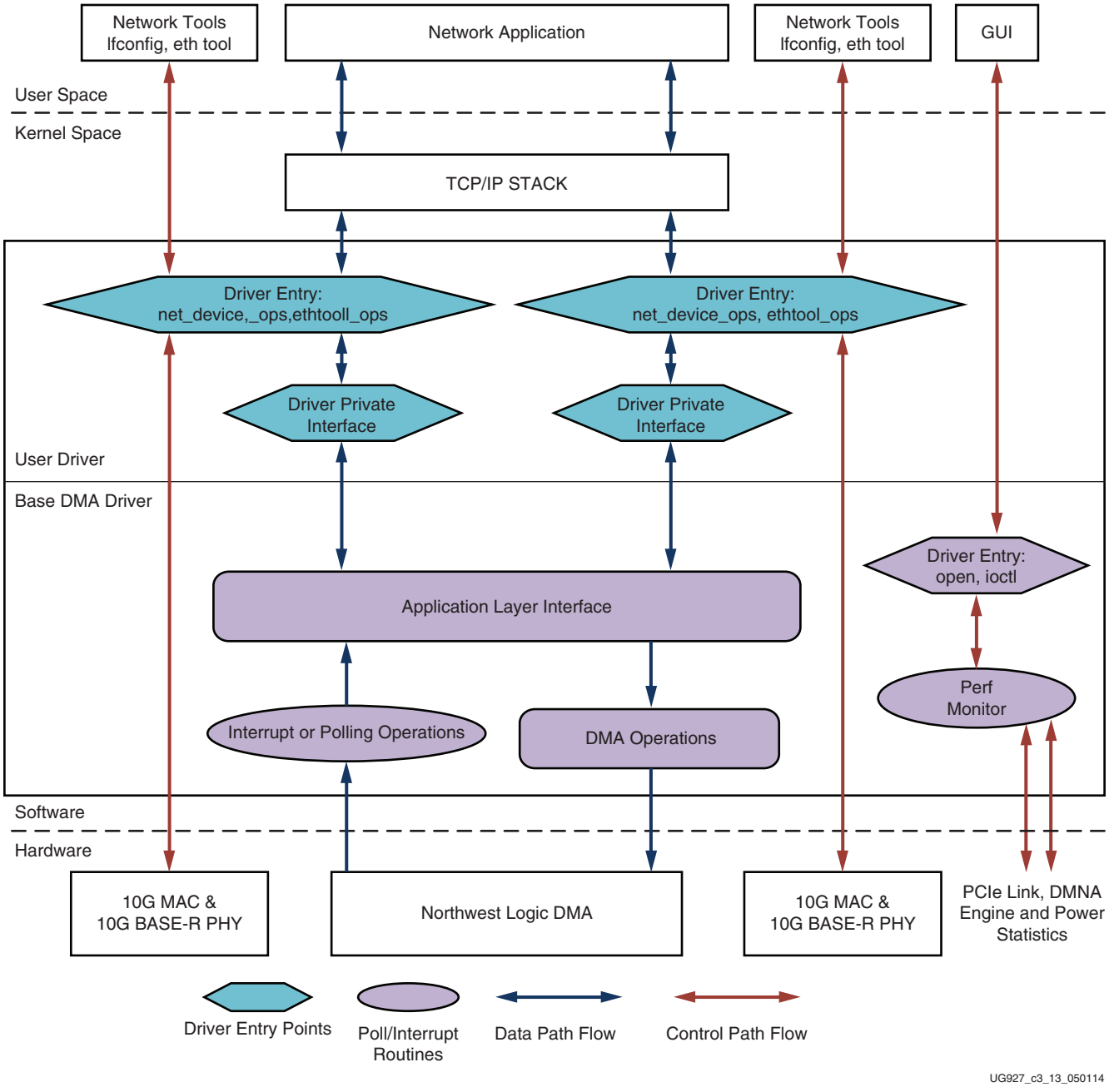


Figure 3-13: **Network Application Mode Software Architecture**

Performance Monitor

The performance monitor is a handler which reads all the performance-related registers (link level for PCI Express, DMA engine level and power level). Each of these parameters is read periodically at an interval of one second.

Data Path Components

Networking Applications

Standard networking applications such as web browser, telnet, or Netperf can be used to initiate traffic in the Ethernet Application mode. In this mode the driver hooks up with the TCP/IP stack software present in the Linux kernel and enables transmission and reception of Ethernet data.

TCP/IP Stack

The TCP/IP stack has defined hooks for the Ethernet driver to attach and allows communication of all standard networking applications with the driver. TCP/IP stack calls appropriate driver entry points to transfer data to driver.

Driver Entry Points

The driver has several entry points, some points are used for data connectivity and others are used for Ethernet configurations. Standard network tools use driver entry points for Ethernet configurations. The driver hooks in entry points configure 10G Ethernet MAC and PHY. The other driver entry points are mainly used in the data flow for transmitting and receiving Ethernet packets.

Application Driver Interface

This block is responsible for dynamic registering and unregistering of user application drivers. The data that is sent from user application driver are sent to DMA operations block.

The DMA and interrupt or polling mode operations remain the same as explained above for Performance mode drivers.

Application Mode Implementation

This section provides an overview of software component implementation for the Application mode. Users are advised to refer to the driver code along with Doxygen generated documentation for further implementation details.

User Application

User applications in this mode are standard network applications such as ping, ftp, http, and web browser. Networking tools open a socket interface and start transmitting the data. The TCP/IP stack segments the packets according to MTU size set in the network device structure. The TCP/IP stack opens the driver interface and sends the packet which is then transmitted to hardware.

Driver Implementation

The user application driver sends the received socket buffer packet to the DMA driver for mapping to PCI space. On the receiver side buffers are pre-allocated to store incoming packets. These packets are allocated from networking stack. The received packets are added to the network stack queue for further processing.

DMA Descriptor Management

This section describes the descriptor management portion of DMA operation. It also describes the data alignment requirements of the DMA engine.

The nature of traffic, especially on the Ethernet side of the design, is bursty, and packets are not of fixed sizes. For example, connect/disconnect establishment and ACK/NAK packets are small. Therefore, the software is not able to determine in advance the number of packets to be transferred, and accordingly set 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 occurs.
- On the receive side, the software needs a way of knowing the size of the actual received packet.

The rest of this section describes how the driver designed uses the features provided by third party DMA IP to achieve the earlier stated objectives.

The status fields in descriptor help define the completion status, start, and end of packet to the software driver.

Table 3-5 presents a summary of the terminology used in the upcoming sections:

Table 3-5: **Terminology Summary**

Term	Description
HW_Completed	Register with the address of the last descriptor for which the DMA engine has completed processing.
HW_Next	Register with the address of the next descriptor that the DMA engine processes.
SW_Next	Register with the address of the next descriptor that software submits for DMA.
ioctl()	Input/output control function is a driver entry point invoked by the application tool.

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. It does not give details on the driver's interactions with upper software layers.

Initialization Phase

The driver prepares descriptor rings, each containing a configurable number of descriptors, for each DMA channel. In the current design, driver thus prepares four rings.

Transmit (S2C) Descriptor Management

In Figure 3-14, the shaded blocks indicate descriptors that are under hardware control and the un-shaded blocks indicate descriptors that are under software control.

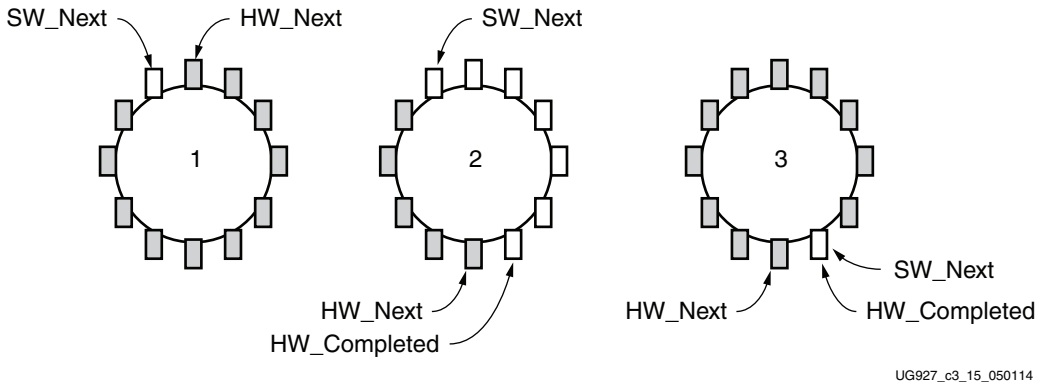


Figure 3-14: **Transmit Descriptor Ring Management**

Initialization Phase (continued):

- 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 arrives in Ethernet packet handler
- Packet is attached to one or more descriptors in ring
- Driver marks SOP, EOP and IRQ_on_completion in descriptors
- Driver adds any user control information (e.g., checksum-related) to 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. Since 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 3-15, the shaded blocks indicate descriptors that are under hardware control and the un-shaded blocks indicate descriptors that are under software control.

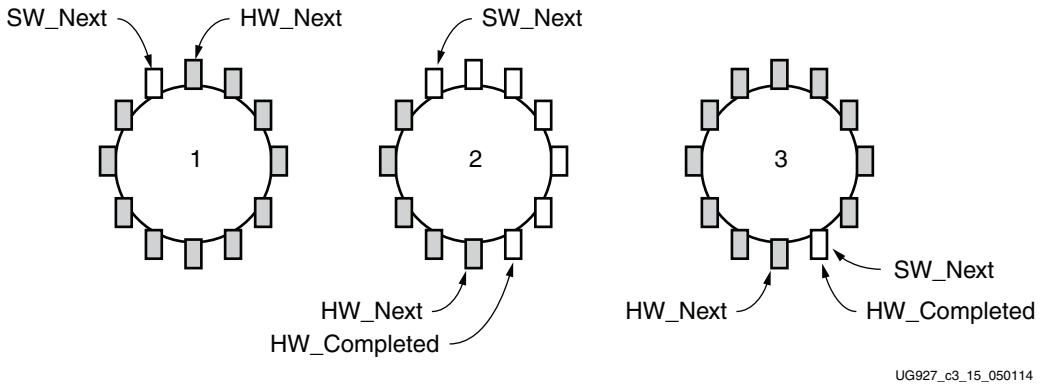


Figure 3-15: **Transmit Descriptor Ring Management**

Initialization Phase (continued)

- Driver initializes each receive descriptor with an appropriate Ethernet or block 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 forwards completed packet buffer(s) to upper layer
- 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. Since the descriptors are already arranged in a ring, post-processing of descriptors is minimal and dynamic allocation of descriptors is not required.

Windows Device Driver and Application

The Windows driver components are divided into different atomic blocks that generate user traffic and control hardware blocks. These atomic blocks can be used in user driver environment to build more sophisticated PCIe-based applications.

The Windows device driver architecture is intended to meet the following objectives:

- Arrange in layers so that user application can be hooked up directly to the driver
- Create a private driver interface which can interact with the block driver and the Ethernet driver interface

Figure 3-16 depicts the software architecture.

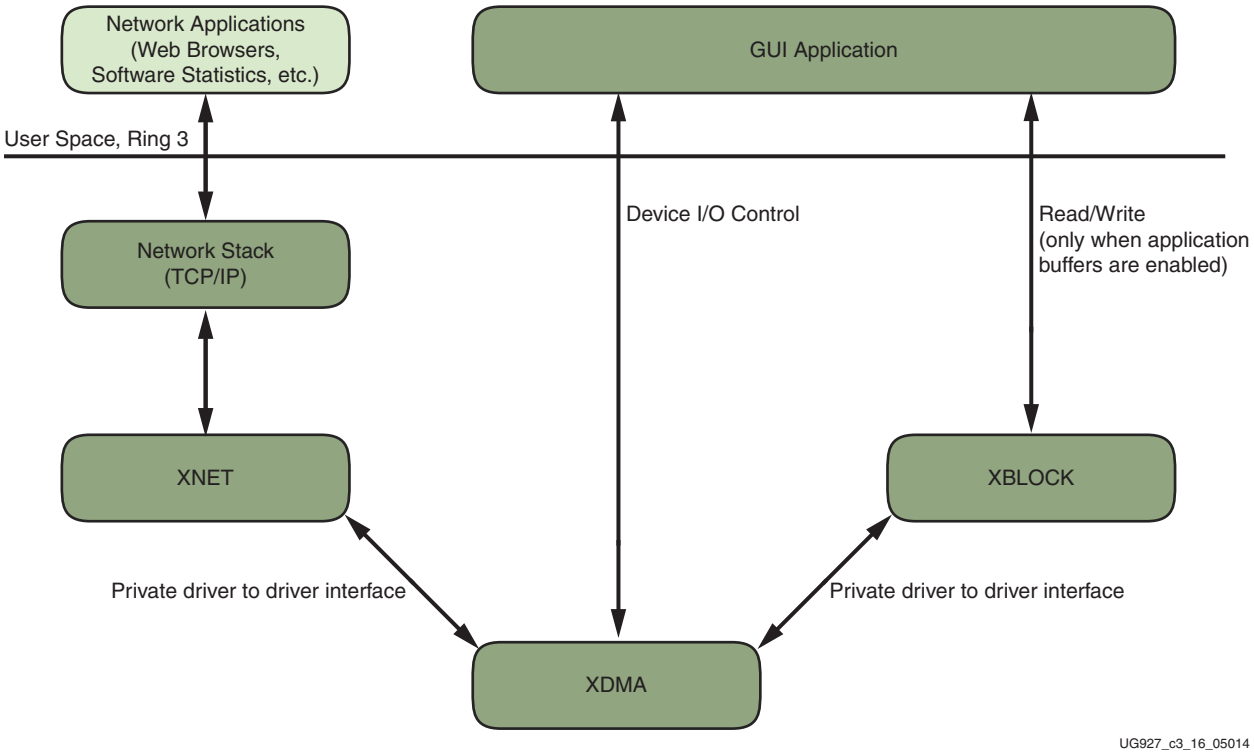


Figure 3-16: **Software Architecture**

At the bottom of the Figure 3-16 is the XDMA PCI Express driver, XBlock functional and XNet NDIS drivers. These drivers operate in the kernel mode that operates in processor ring 0 protection. The purpose of a kernel driver is to be the interface between a software application and the hardware device. The driver is designed with specific knowledge of the hardware and how best to make this interface available to applications. For this discussion the term application is a program like a web browser, editor, viewer, etc. Applications use Windows APIs to interface with the operating system.

In this design, the XDMA driver is the controlled access point to the NWL DMA block present in the hardware design. The XDMA controls the DMA engines, handles interrupts and has interfaces for both driver and application status requests. It also handles system requests like system sleep (D1-D3) and wake (back to D0) functions. The XNet and XBlock drivers are function specific drivers (FSD) that must interface with the XDMA driver. The XDMA driver exposes a private driver to driver interface in which the XNet and XBlock drivers communicate.

The FSD interface abstracts all the details of the DMA engines. The FSDs handle the interface to the operating system or applications. The FSD performs the necessary scatter/gather operations and general request handling. A single function call is made to the XDMA driver to start a transfer and a callback is supplied for request completions. The two FSDs in this architecture are XBlock and XNet.

The XNet driver is a NDIS 6.x compliant driver and works with existing Microsoft Windows networking stacks such as TCP/IP. The XNet driver works like any other compliant NDIS device and presents a wired 10 gigabit Ethernet interface device to the network stacks. Any standard network applications such as ping, iperf, ftp or web browsers can be used to communicate through these interfaces.

The XBlock driver provides a block like read/write interface to the DMA engines on the Xilinx hardware facilitating transmit or receive of data present in the buffers, from the application user space, if that mode is enabled using the Windows registry key TestConfig. By default, Application Buffer Transfer mode is enabled wherein buffer addresses to be programmed in the buffer descriptors are provided by the user application. These modes are described in Internal Buffer Transfers, page 78 and Application Buffer Transfers, page 78. The XBlock drivers can also be configured to generate data with a standard Ethernet header to showcase the capabilities of the 10G MAC used in the design. This mode is called RAW Ethernet mode.

The application is the graphical user interface into the XBlock and XDMA driver in the case of Performance/Raw Ethernet mode of operation. The application communicates to the XBlock driver using standard Windows operating system calls such as CreateFile, ReadFile, WriteFile, CloseFile, etc. The XDMA driver communicates using standard Windows operating system calls such as CreateFile, DeviceIOControl, and CloseFile.

XDMA Overview

The XDMA driver is responsible for all things related to the NWL DMA block. It handles the initialization of the DMA Engines, Interrupt control and routing, deferred procedure calls (DPC), timers, request handlers, etc. The XDMA driver presents both a user application interface via DeviceIOControl (ioctl) and a driver-to-driver interface. The driver-to-driver interface allows for function specific drivers to be layered above the XDMA driver. The interface is designed to be direct and efficient.

The XDMA driver verifies the hardware and initializes it according to the given configuration. The driver creates one context space that is shared for all DMA Engines for a particular board. The XDMA driver also creates separate data spaces for each DMA engine yielding each DMA engine its own context to operate and does not share anything with the other DMA engines with the exception of timers and interrupts. Each DMA engine allocates its own DMA objects (both 32- and 64-bit), descriptor pool and DPC. The driver creates a linked list of descriptors at initialization time along with head and tail pointers that point directly at the DMA descriptors.

When a DMA request is given to the XDMA, the driver retrieves the next DMA descriptor, fills out the descriptor with the scatter/gather information provided, and moves the DMA engine software descriptor pointer to start the DMA For more information on DMA data transfer please refer to Linux Device Driver and Application, page 63.

Interrupts are used to task the completion of the DMA. Once the interrupt has occurred and is acknowledged for a DMA engine, a DPC is scheduled for that DMA engine alone. When the DPC runs, the actual completion of the DMA transaction occurs. This includes freeing the DMA descriptors for reuse, issuing the callback to the FSD to free the scatter/gather list and to complete the WDF Request.

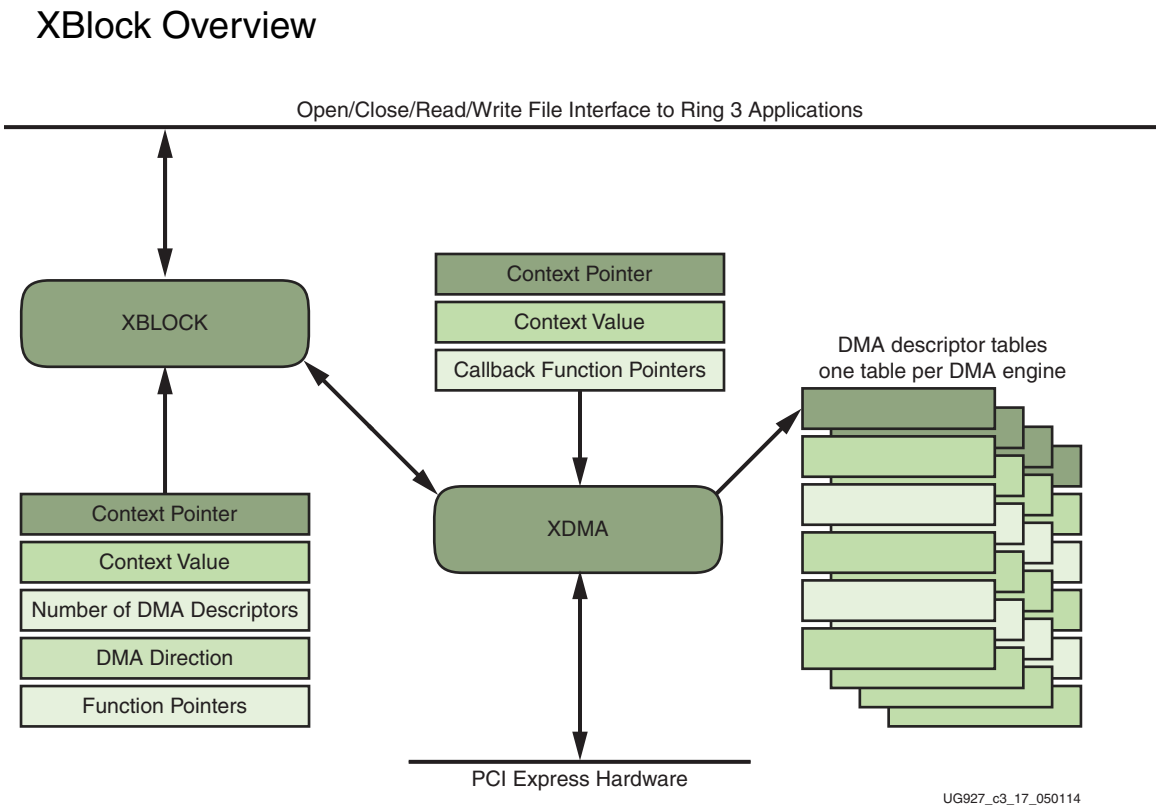


Figure 3-17: **XBlock to XDMA Interface**

When an application (in the Processor Ring 3 protection) wants to transfer data via a DMA device it must use a kernel driver to do so. The application allocates a buffer but the memory address is a virtual address. This virtual address can represent many fragments of physical memory scattered throughout the system. It is the job of the kernel driver along with the operating system WDF support functions to convert the virtual address into something usable by the DMA device. In addition to converting the virtual address into physical addresses the data at these addresses must be locked down, meaning the operating system must not touch or swap the memory out to disk while a DMA is taking place. The memory must also not be cached otherwise the DMA will have replaced the contents while the processor reads from a stale cache.

The XBlock driver has read and write functions exposed to applications when Application Buffers mode is enabled. This allows applications to access the DMA engines using standard Open/CreateFile, ReadFile, WriteFile, CloseFile operating system functions to move data across the PCI Express bus. ReadFile is used to program free buffers into the DMA descriptors to receive data from the card and WriteFile is used to transmit data to card. Multiple threads invoking ReadFile and WriteFile can be used to transmit/receive data simultaneously.

The XBlock driver does not know about the underlying hardware as the XDMA driver abstracts that information. The XBlock driver communicates through the XDMA_Link structures that contains information on how to submit transactions, and contains a WDF object that has specific information used by calls in the FSD to do scatter/gather operations.

When the XBlock driver receives a request from the operation system, it performs a scatter/gather operation using the WDF object to retrieve the physical buffer addresses and to lock the buffers in preparation for DMA. Once the scatter/gather operation is

complete the XDMA driver is called. The XDMA driver retrieves the next available DMA descriptors in the list, fills them out with the scatter/gather information and starts the DMA by moving the DMA engine pointer. Control is returned to XBlock where the request is marked pending and control returns to the kernel.

When an interrupt occurs the DMA engines are checked to determine which engine(s) caused an interrupt and a DPC is schedule for each DMA engine that requires service. When the DPC runs the DMA descriptor list is traversed looking for a completed packet. If a completed packet is found the FSD (XBlock in this case) information is retrieved and the FSD is called. The FSD releases the scatter/gather information along the with the DMA transaction object. The request is completed by a WDF call with the request handle and the information about the transfer. Control is returned to XDMA to look for more completed packets or exit the DPC.

Windows requires the DMA transaction is complete before indicating completion to the operating system. If this is not adhered to the operating will free resources being used during the DMA transfer with disastrous consequences.

XNet Overview

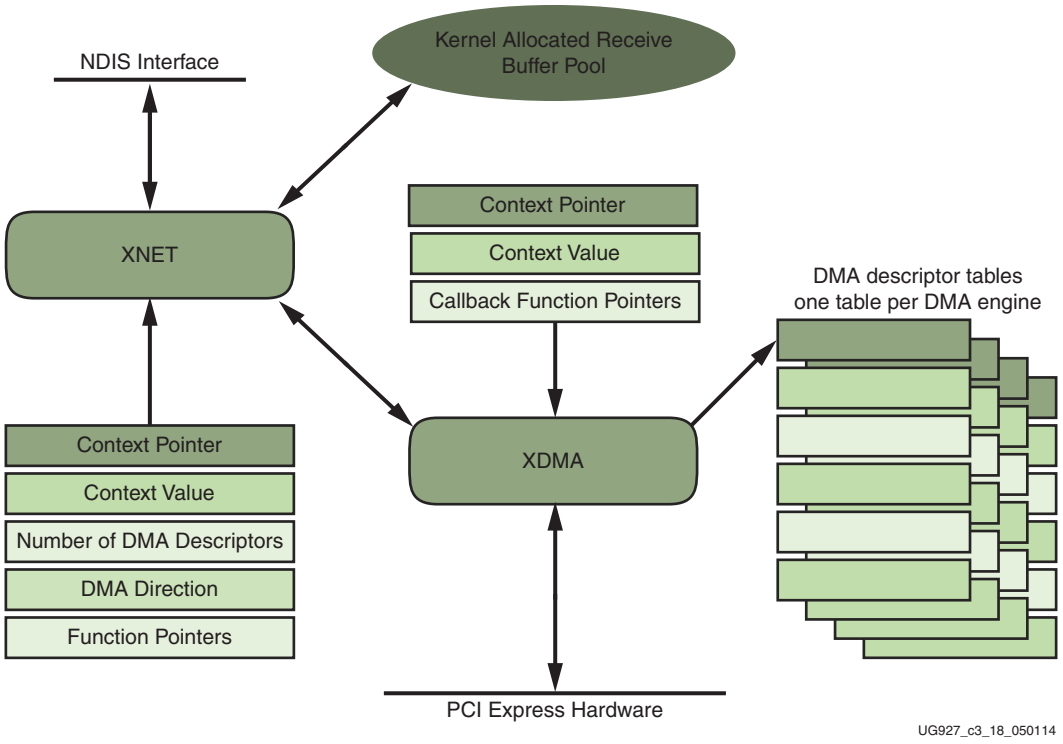


Figure 3-18: **XNet to XDMA Interface**

The XNet driver is a standard NDIS 6.x driver that communicates on the bottom end with the XDMA driver instead of directly with the hardware and presents a wired 10 gigabit Ethernet network device the protocol stacks above. The XNet driver communicates through the XDMA link structures that contains information on how to submit transactions, and contains a WDF object that has specific information used by calls in the FSD to do Scatter/Gather operations.

Modes of Operation

XDMA

Child Driver Configuration

The child driver configuration uses the registry key name ChildDriverConfig to control what function specific drivers are loaded. The registry key is a DWORD value that has the following settings:

- XBlock - This loads up to two XBlock drivers. One for each DMA engine pair found.
- XNet - This loads up to two XNet drivers. One for each DMA engine pair found.

The registry key is created by the `XDMA.inf` file in the section XDMAInst_ChildDriver_AddReg.

Performance

The scatter gather DMA fetches the buffers queued up for transmission in the host system memory as packets to transfer them over the PCIe link. The performance mode depends on the value of the registry key name RawEthernet. The registry key is a DWORD value. For Performance mode the value is set to 0.

This key is created by the `XDMA.inf` file in the section XDMAInst_RawEthernet_AddReg.

RAW Ethernet

The Scatter Gather DMA fetches the buffers queued up for transmission in the host system memory as packets to transfer them over the PCIe link. In the transmit direction, data is transferred over to the 10G Ethernet MAC block in hardware. In the receive direction, the data received in the 10G Ethernet MAC is transferred over to the host system.. The Raw Ethernet mode depends on the value of registry key name RawEthernet. Also, all traffic generated through the XBlock drivers contains an Ethernet header and this design demonstrates the maximum capability of the 10G MAC because there are no TCP/IP stack overheads. The registry key is a DWORD value. For raw Ethernet mode the value is set to 1.

This key is created by the `XDMA.inf` file in the section XDMAInst_RawEthernet_AddReg.

Internal Buffer Transfers

The Internal Buffer Transfers mode uses the registry key name TestConfig to control the application or kernel buffer type to be programmed into the DMA buffer descriptor rings. The registry key is a DWORD value. For Internal Buffer Transfers mode the value is set to 1.

This key is created by the `XDMA.inf` file in the section XDMAInst_TestConfig_AddReg.

When TestConfig key is set to 1, the driver uses an internal kernel buffer to transfer data to/from the device. The DMA descriptors are set to kernel buffer addresses. In the Internal Buffer Transfer mode interrupts are not used. A low resolution timer is used to poll the descriptor list and gather statistics. PCIe and DMA performance numbers observed in this mode are much higher than what is observed with application buffer transfers.

Application Buffer Transfers

The Application Buffer Transfers mode uses the registry key name TestConfig to control the configuration of the driver when a start test command is issued. The Registry key is a DWORD value. For Application Buffer Transfers mode the value is set to 0.

This key is created by the `XDMA.inf` file in the section XDMAInst_TestConfig_AddReg.

When application buffer transfers is selected the driver the buffer(s) are provided by the function specific driver to transfer data to/from. Interrupts are used to control the DMA engine and determine completion of a DMA. This is necessary in Windows so resources are not released prematurely which can cause a system crash.

XBlock

There are no configurable modes for the XBlock driver.

XNet

Network Address override

The network address uses the registry key name NetworkAddress to override the hardware-supplied MAC address used by the instance of the driver. This is a standard NDIS registry key.

This key is created by the `XNet.inf` file in the section XilNetworkAddress.reg.

It is necessary to change the MAC address if two KC705 boards are connected back-to-back in peer-to-peer mode because both of their MAC addresses will be initially the same.

Jumbo Frame

The jumbo frame uses the registry key name JumboFrame to control the maximum frame size supported by the NDI driver.

This key is created by the `XNet.inf` file in the section XilJumboFrame.reg. The NDI installer performs the transform from UI selection into a number ranging from 1,514 to 7,168 bytes. The frame size can be modified from the Windows standard networking and sharing utility.

Data Flow

XDMA

Initialization

The driver starts at the DriverEntry function. DriverEntry registers the driver with the PnP Manager. When a device is found the PnP Manger will call XlxEvtDeviceAdd. This is where most of the driver setup is done and various driver functions are registered. The driver allocates the device extension memory to hold information about the card. The driver reads the registry entries to gather configuration information for what FSDs to load, configure performance, or Raw Ethernet, etc. and sets up the interrupts based on the card and system capabilities to support MSI-X, MSI or legacy interrupts. The driver creates a private driver-to-driver interface in which the FSDs communicate. This is the interface that the XNet and XBlock drivers use to establish connection to the XDMA driver.

When the operating system is ready it will call XlxEvtDevicePrepareHardware. The Prepare hardware function is where we setup the individual DMA Engines. The driver reads registers mapped via BAR0 to retrieve the configuration of the DMA engine. The driver then initializes and sets up each DMA Engine with its own data space to be able to operate autonomously. The driver pre-allocates space in the first 4 GB of memory for the DMA descriptor pools, one pool for each DMA engine. The DMA descriptors are

initialized and statically setup into a linked list with the head and tail pointers pointing to the beginning of the list.

When the operating system is fully operational it signals the driver to enter D0 state which corresponds to system power states. In this state the timers are started and the DMA engines are marked as AVAILABLE. At this point linking to the FSBs are allowed.

DMA Engines 0 and 1 (hardware designation 0 and 1) are S2C or write interfaces while DMA Engines 2 and 3 (hardware designation 32 and 33) are C2S or read interfaces.

Functional interfaces

The XDMA does not initiate DMA transfers unless it is configured for internal transfers. The FSBs initiate data transfers. The XDMA driver does present an abbreviated xpmon_be (Xilinx Performance Monitor Back End) I/O Control interface. XDMA supports the following xpmon_be functions:

- IGET_TEST_STATE
- ISTART_TEST
- ISTOP_TEST
- IGET_LED_STATISTICS
- IGET_PCI_STATE
- IGET_ENG_STATE
- IGET_DMA_STATISTICS
- IGET_TRN_STATISTICS
- IGET_SW_STATISTICS

XBlock

Initialization

The XBlock driver performs the standard WDF driver entry and setup requires along with establish a link to the XDMA driver. After the link to XDMA, the XBlock checks the capabilities of the underlying hardware to make sure it is suitable for its use. XBlock sets up entry points for read and write.

XBlock attempts to establish a linkage to a pair of DMA engines corresponding to App0. If the linkage fails then App1 is attempted. If App0 is linked, XBlock will use DMA engines 0 and 2 (hardware designation 0 and 32). If App1 is linked, XBlock will use DMA engine 1 and 3 (hardware designation 1 and 33).

Functional interfaces

The XBlock driver can be opened like a file but the driver name uses a GUID, which is a driver specific unique id as part of it name so it takes a little more work to resolve the name. The GUI program includes a routine called OpenDriverInterface in the `DriverGenInfo.cpp` file that can be an example of how to resolve the driver name. After the driver name is resolved the interface can be opened using the Windows operating system CreateFile call. ReadFile, WriteFile and CloseHandle can all be used when interfacing with XBlock.

XNet

Initialization

The XNet driver on the top presents a NDIS 6.x interface while the bottom edge is WDF. The XNet driver performs all the normal initialization functions necessary for an NDIS 6.x driver in addition it also creates a WDF context for the bottom edge of the driver to allow it to link with the XDMA driver. Once the NDIS initialization is accomplished the XNet driver then attempts to link to the XDMA driver.

XNet establishes a linkage to a pair of DMA engines depending on what interface is set in the registry when the driver is loaded. The definition XNET_SEND_DMA_ENGINE can be defined as either DMA engines 0 or 1 (hardware designation 0 or 1). The definition XNET_RECV_DMA_ENGINE can be defined as either DMA engines 2 or 3 (hardware designation 32 or 33). DMA engines 0 and 1 (hardware designation 0 and 1) are S2C or send interfaces while DMA Engines 2 and 3 (hardware designation 32 and 33) are C2S or receive interfaces.

When the XNet driver loads it creates a network buffer list (NBL), scatter/gather list entries and packet buffers for receives. These components are scatter/gathered once at initialization time and recycled between the NDIS protocol stacks and the driver. This increases the efficiency greatly since the scatter/gather is not in the performance path. NDIS send packets do require scatter/gather in the performance path. This is due to data in the buffer can be the application buffer.

Functional interfaces

The XNet driver supports the standard NDIS OIDs necessary to be NDIS 6.0 compliant. XNet does not expose an xpmon_be interface since the NDIS driver naming is obscured making a CreateFile, DeviceIOControl more difficult. Please refer to the NDIS 6.x documentation regarding supported OIDs.

NDIS does not use or expose any standard operating system interfaces like ReadFile, WriteFile, etc.

User Interface – Control and Monitor GUI

While invoking the GUI, a launching page is displayed which detects the PCIe device for this design (Vendor ID = `0x10EE` and Device ID = `0x7082`). It allows driver installation to proceed only on detection of the appropriate device. The user can select any one of the following operating configurations:

1. Performance (PCIe-DMA or Gen/CHK) Mode
2. Performance Mode (Raw Ethernet)
3. Application Mode

All three modes of configuration are mutually exclusive. In Performance or Raw Ethernet mode, the user can select an additional option to enable a data integrity check. Upon successful installation of drivers, the control and monitor GUI is displayed.

GUI Control Function

The following parameters are controlled through the GUI:

- Packet size for traffic generation
- Test type loopback in case of raw Ethernet and loopback/Hw checker/Hw Generator for Performance mode.

- Changing PCIe Link speed and width

GUI Monitor Function

The driver always maintains information about the hardware status. The GUI periodically invokes an I/O control, ioctl() to read this status information which comprises:

- PCIe link status, device status
- DMA engine status
- Power 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

The various GUI fields highlighted in Figure 3-19 are explained as follows:

1. Indicates DDR3 calibration information, green on calibration red otherwise.
2. 10G PHY-0 link status.
3. 10G PHY-1 link status.
4. Mode of operation – In Performance GEN/CHEK mode the user has the option to select Loopback or Hw Gen/Hw checker while in raw Ethernet only loopback is allowed. In Application mode it is grayed out as traffic is generated from a networking tool.
5. Packet size for test run. Allowed packet size is shown in tool tip.
6. Test start/stop control for Performance mode.
7. DMA statistics and software BD provides the following information:
    - Throughput (Gb/s) – DMA payload throughput in gigabits per second for each engine.
    - DMA active time (ns) – The time in nanoseconds that the DMA engine has been active in the last second.
    - DMA wait time (ns) – The time in nanosecond that the DMA was waiting for the software to provide more descriptors.
    - BD errors – Indicates a count of descriptors that caused a DMA error. Indicated by the error status field in the descriptor update.
    - 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.
    - SW BDs – Indicates the count of total descriptors set up in the descriptor ring.

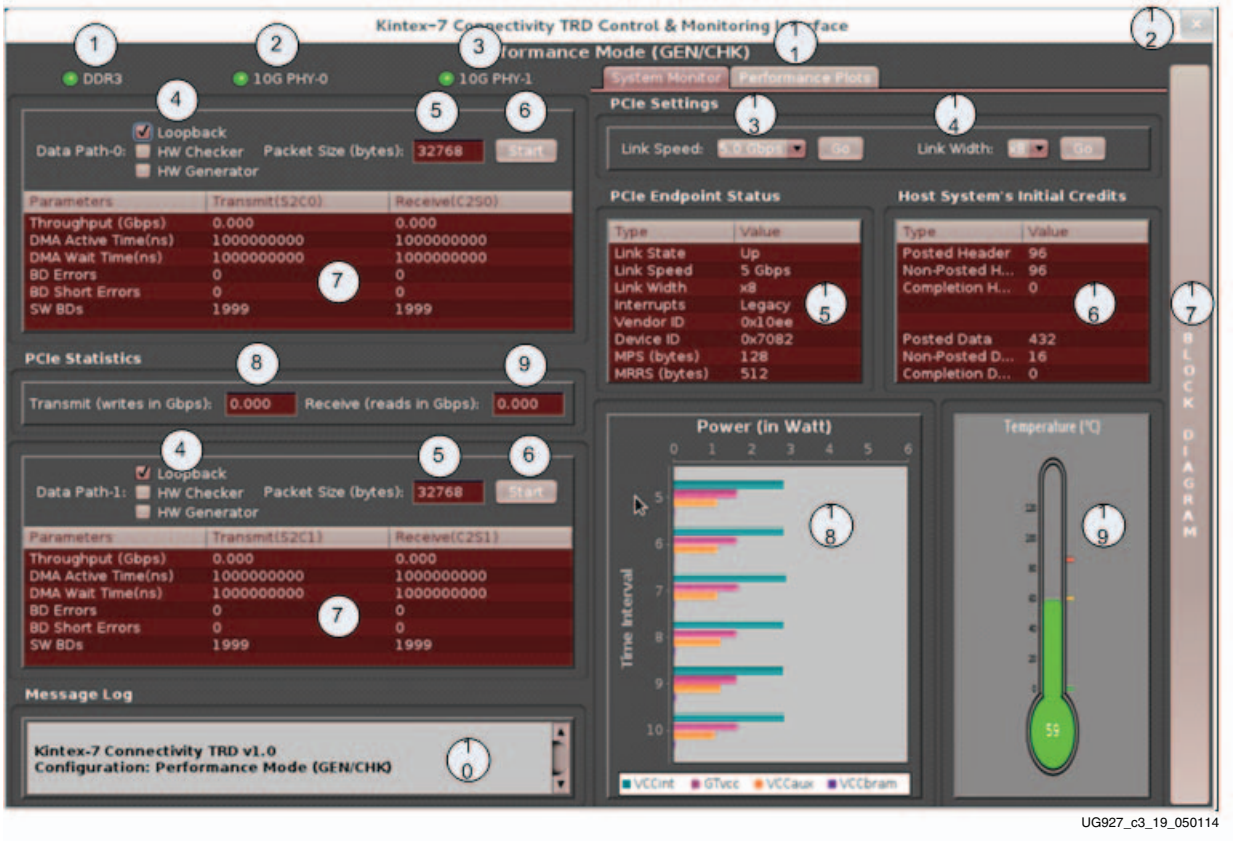


Figure 3-19: **Software GUI Screen Capture**

8. PCIe transmit (writes) (Gb/s) – Reports transmitted (Endpoint card to host) utilization as obtained from the PCIe performance monitor in hardware.
9. PCIe receive (reads) (Gb/s) – Reports received (host to Endpoint card) utilization as obtained from the PCIe performance monitor in hardware.
10. Message log – The text pane at the bottom shows informational messages, warnings, or errors.
11. Performance plots tab – Plots the PCIe transactions on the AXI4-Stream interface and shows the payload statistics graphs based on DMA engine performance monitor.
12. Close button – This button closes the GUI.
13. Directed link speed change – Option to change link speed. Drop down box shows allowed speed changes. GO Button sets the corresponding speed.
14. Directed link width change – Option to change link width. Drop down box show cases allowed width changes. GO button sets the corresponding width.
15. PCIe Endpoint status – Reports the status of various PCIe fields as reported in the Endpoint's configuration space.
16. Host system's initial 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. Block diagram button – This button show cases block diagram of each mode which is running.

18. Power statistics – Power in watts is plotted for various rails namely, V_{CCINT}, G_{TVCC}, V_{CCAUX} and V_{CCBRAM}.
19. Temperature monitor shows current die temperature.

This GUI is developed in JAVA environment. Java Native Interface (JNI) is used to build the bridge between driver and UI. Same code can be used for windows operating system with minor changes in JNI for operating system related calls.

Power Management

The power management in the Kintex-7 Connectivity TRD supports various system power states. System power management is based on PM events raised across the system. Events such as standby and hibernate are raised as a move to bring the entire system to low power states. Various system level transactions are show in Table 3-6.

Table 3-6: **Power States**

Global	OS	PCI Device	Link State	Description
G0	S0	D0	L0	Working
			L0s	Hardware autonomous, software independent low resume latency ASPM state
G1				Sleep
	S1	D1	L1	Caches flushed, CPU stops execution, CPU, RAM power is ON, and devices might/might not be up.
	S2			CPU is powered OFF (not commonly implemented)
	S3	D2	L1	Standby (suspend-to-RAM), remaining power ON
	S4	D3 Hot	L2	Hibernation (suspend-to-Disk), powered down
G2	S5	D3 Cold	L2 (aux power)	Soft OFF; some peripherals are ON for wake signal (keyboard, clock, modem, LAN, USB etc.)
G3				Mechanical OFF

The Kintex-7 TRD supports four system power transitions:

- System Suspend
- System resume
- System Hibernate
- System Restore.

These power transitions are supported by registering a set of call back functions with the PM subsystem. These call back functions are invoked by the PM subsystem based on the system level power state transitions.

Implementation Details of PM

Table 3-7 explains each call back function hook description and its corresponding implementation in the Kintex-7 FPGA Connectivity TRD.

Table 3-7: **Call Back Function Hook Description and Corresponding Implementation**

Original Description	Implementation
prepare()	

• It is executed during
  • Suspend
  • Hibernation (image about to be created)
  • Power off, after saving hibernate image
  • System restore (hibernate image has just been restored to memory)
• Role of prepare()
  • Prevent new children being registered until any of these callbacks are invoked: resume_noirq(), thaw_noirq(), restore_noirq()
  • It should not allocate any memory
	• Change the DriverState to PM_PREPARE
	• Flag is checked in:
	  • Do not allow any application driver registration
	  • Do not allow any application to open the driver interface
	• Invoke the application driver's `prepare()` hook function.
	  • This makes the application driver stop the TX queue.

Table 3-7: **Call Back Function Hook Description and Corresponding Implementation *(Cont'd)***

Original Description	Implementation
suspend()	

• It is executed during
  • Suspend
• Role of suspend()
  • Makes the device quiescent and prepares it for a low power state.
  • DO NOT save the configuration registers, prepare wakeup signaling, or put the device in low power state, because the PCI subsystem takes care of these (a few drivers might deviate from this norm).
  • Interrupts are still enabled.
	• Ensure TX BD ring is empty (no more TX)
	  • Wait for DMA to complete all queued up packets.
	  • Do not schedule further packet transmission.
	• Sleep awhile based on timeout
	  • This ensures that packets in transit have made it out of the FPGA to their respective destinations.
	• Invoke the App Driver's suspend_early() hook function.
	  • This hook function is to perform SUSPEND related activities on the application hardware by the application driver.
	  • Disable the MAC engine for TX.
	  • Disable the MAC engine for RX.
	• Get the leftover contents from VFIFO.
	  • This data is passed to the application layer.
	  • Because RX has BDs already posted to DMA, the leftover data comes to XDMA automatically.
	  • The driver has to handle this data normally. No special action is required from the driver to get the data from VFIFO.
	• Issue a soft reset to DMA C2S engine after a timeout.
	  • This ensures that unused BDs (the ones that have been pre-fetched by XDMA) are posted back by the DMA engine.
	• Ensure the RX BD ring is empty (no more RX).
	  • Pointers being the same in driver RX ring
	  • When RX is done, RX-XDMA is automatically stopped.
	• Invoke the application driver's suspend_late() hook function. This hook function is to perform SUSPEND related activities on the application hardware by the application driver.
	  • Stop TX and RX queues in the network interface. Detach the network interface.
	• Disable interrupts, timers, and/or polling.

Table 3-7: **Call Back Function Hook Description and Corresponding Implementation *(Cont'd)***

Original Description	Implementation
freeze()	

• It is executed during
  • Hibernation, after prepare() callbacks have been executed for all devices in preparation for the creation of a system image.
  • Restore, after a system image has been loaded into memory from persistent storage and the prepare() callbacks have been executed for all devices.
• Role of freeze()
  • It is analogous to suspend().
  • Saves the configuration register.
  • DO NOT otherwise put the device into a low power state and DO NOT emit system wakeup events.
	• Invoke SUSPEND.
	• Set PCI device states.
	• Save current PCI device state into hibernate image.
	• Set PCI power state to current state.

poweroff()	

• It is executed during
  • Hibernation, when the system is about to be powered off, after the system image is saved onto disk.
• Role of poweroff()
  • It is analogous to suspend() and freeze().
  • It does not save the configuration registers.
  • It saves other hardware registers, in case the driver handles the low-power state.
  • Interrupts are still enabled.
	• Invoke SUSPEND

resume()	

• It is executed during
  • System resume, after enabling CPU cores (the contents of main memory were preserved).
  • Interrupts are enabled.
• Role of resume()
  • Used to restore the pre-suspend configuration of the device.
	• Normal operation enable for DDR3.
	• Enable interrupts, timers, and/or polling.
	• Invoke application drivers' resume() hook function. This hook function does the following:
	  • Performs RESUME related activities on the application hardware by the application driver.
	  • Enables PHY engine for the TX.
	  • Enables PHY engine for the RX.
	  • Enables MAC engine for the TX.
	  • Enables MAC engine for the RX.
	  • Attaches the network interface.
	  • Starts the TX queue in the network interface.
	  • Starts the RX queue in the network interface.
	• Changes the flag DriverState to REGISTERED. This allows the TX traffic to resume.

Table 3-7: **Call Back Function Hook Description and Corresponding Implementation *(Cont'd)***

Original Description	Implementation
thaw()	

• It is executed during
  • Hibernate (after invoking thaw_noirq()).
  • Interrupts are enabled.
• Role of thaw()
  • Similar to resume().
  • This call can modify the hardware registers.
	• Invoke RESUME

restore()	

• It is executed during
  • Hibernate, specifically after invoking thaw_noirq()
  • Interrupts are disabled.
• Role of restore_noirq()
  • Similar to resume_noirq()
	• Invoke RESUME

Test Procedure through Sys File System

Power management can be tested in Test mode or actual Power Management (PM) mode using sys file system. In Test mode, the system transits to a changed state and resumes after few seconds. In actual PM mode, the system transits to a corresponding state change. This mode requires super user permission.

- Suspend in PM Test mode
  - echo devices > /sys/power/pm_test
  - echo platform > /sys/power/disk
  - echo mem > /sys/power/state

  Expected behavior:

  Ping response stops for five seconds and resumes automatically.
- Suspend in actual PM mode
  - echo none > /sys/power/pm_test
  - echo platform > /sys/power/disk
  - echo mem > /sys/power/state

  Expected behavior:

  Host machine turns off and ping response stops. If the host machine is powered up, the ping response resumes automatically.
- Hibernate in PM Test mode
  - echo devices > /sys/power/pm_test
  - echo platform > /sys/power/disk
  - echo disk > /sys/power/state

  Expected behavior:

  Ping response stops for five seconds and resumes automatically.

- Hibernate in Actual PM mode (echo none > /sys/power/pm_test; echo disk > /sys/power/state)
  - echo none > /sys/power/pm_test
  - echo platform > /sys/power/disk
  - echo disk > /sys/power/state

  Expected behavior:

  Host machine turns off and ping response stops. If the host machine is powered up, the ping response resumes automatically.

For further details, refer to corresponding kernel documentation available at the Linux Kernel Organization.

Application Driven Power Management

The user can initiate application driven power management using the Kintex-7 Connectivity TRD GUI. The user can select link width and speed, and the driver programs the appropriate registers. The power statistics changes are reflected in the power graph in the GUI.

For more information on hardware programming, refer to Application Demand Driven Power Management, page 59.

XILINX

Chapter 4

Performance Estimation

This chapter presents a theoretical estimation of performance, lists the performance measured, and provides a mechanism for the user to measure performance.

Theoretical Estimate

This section provides a theoretical estimate of performance.

PCI Express - DMA

PCI Express® is a serialized, high bandwidth and scalable point-to-point protocol that provides highly reliable data transfer operations. The maximum transfer rate for a 2.1-compliant device is 5 Gb/s/lane/direction. The actual throughput would be lower due to protocol overheads and system design tradeoffs. Refer to *Understanding Performance of PCI Express Systems* for more information (WP350) [Ref 8].

This section gives an estimate on performance on the PCI Express link using Northwest Logic Packet DMA.

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

- Each buffer descriptor points to a 4 KB data buffer space
- Maximum payload size (MPS) = 12B
- Maximum read request size (MRRS) = 128B
- Read completion boundary (RCB) = 64B
- TLPs of 3DW considered without extended CRC (ECRC) – total overhead of 20B
- One ACK assumed per TLP – DLLP overhead of 8B
- Update FC DLLPs are not accounted for but they do affect the final throughput slightly.

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

The following conventions are used in the calculations that follow.

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).

Note: Traffic on upstream (card to system) PCIe link is **bolded** and traffic on downstream (system to card) PCIe link is *italicized*.

The C2S DMA engine (which deals with data reception, that is, 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 – C2S Desc Fetch = 20/4096 = 0.625/128	*8/4096 = 0.25/128*	One descriptor fetch in C2S engine for 4 KB data (TRN–TX); 20B of TLP overhead and 8 bytes DLLP overhead
CPLD – C2S Desc Completion = (20+32)/4096 = 1.625/128	**8/4096 = 0.25/128**	Descriptor reception C2S engine (TRN-RX). CPLD header is 20 bytes and the C2S Desc data is 32 bytes.
MWR – C2S buffer write = 20/128	*8/128*	MPS = 128B; Buffer write C2S engine (TRN-TX).
MWR – C2S Desc Update = (20+12)/4096 = 1/128	*8/4096 = 0.25/128*	Descriptor update 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 card to the system, the overhead on the upstream link (in **bold**) is 21.875 bytes.

% Overhead = 21.875/(128 + 21.875) = 14.60%

The throughput per PCIe lane is 5 Gb/s, but because of 8B/10B encoding, the throughput comes down to 4 Gb/s.

Maximum theoretical throughput per lane for Receive = (100 – 14.60)/100 * 4 = 3.40 Gb/s

Maximum theoretical throughput for a x8 Gen2 link for Receive = 8 * 3.4 = 27.2 Gb/s

The S2C DMA engine (which deals with data transmission, that is, 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 – S2C Desc Fetch =20/4096=0.625/128	*8/4096 = 0.25/128*	Descriptor fetch in S2C engine (TRN-TX)
CPLD – S2C Desc Completion =(20+32)/4096=1.625/128	**8/4096 = 0.25/128**	Descriptor reception S2C engine (TRN-RX). CPLD header is 20 bytes and the S2C Desc data is 32 bytes.
MRD – S2C Buffer Fetch = 20/128	*8/128*	Buffer fetch S2C engine (TRN-TX). MRRS=128B
CPLD – S2C buffer Completion = 20/64 = 40/128	**8/64 = 16/128**	Buffer reception S2C engine (TRN-RX). Because RCB=64B, 2 completions are received for every 128 byte read request
MWR – S2C Desc Update =20+4/4096=0.75/128	*8/4096=0.25/128*	Descriptor update 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 5 Gb/s, but because of 8B/10B encoding the throughput comes down to 4 Gb/s.

Maximum theoretical throughput per lane for Transmit = (100 – 28.14)/100 * 4 = 2.86 Gb/s

Maximum theoretical throughput for a x8 Gen2 link for Transmit = 22.88 Gb/s

For transmit (S2C), the effective throughput is 22.8 Gb/s and for receive (C2S) it is 27.2 Gb/s.

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

- The transaction interface of PCIe is 128-bit wide. The data sent is not always 128-bit aligned and this could cause some reduction in throughput.
- Changes in MPS, MRRS, RCB, or buffer descriptor size also have significant impact on the throughput.
- If bidirectional traffic is enabled, overhead incurred reduces throughput further.
- Software overhead or latencies contribute to throughput reduction.

AXI Virtual FIFO

The design uses 64-bit DDR3 operating at 800 MHz or 1600 Mb/s. This provides a total performance of 64 x 1600 = 100 Gb/s.

For a burst size of 128, the total bits to be transferred is

64 x 128 = 8192 bits

For DDR3, the number of bits transferred per cycle is

64 (DDR3 bit width) x 2 (double data rate) = 128 per cycle

The total number of cycles for transfer of 8192 bits is

8192/128 = 64 cycles

Assuming 10 cycles read to write overhead, efficiency is

64/74 = 86%

Assuming 5% overhead for refresh and so on, the total achievable efficiency is ~81%, which is ~81 Gb/s throughput on the AXI Virtual FIFO controller.

Ten Gig Ethernet MAC

The XGEMAC operates at 156.25 MHz clock and a 64-bit datapath width (64 x 156.25 = 10 Gb/s).

For XGMII, three cycles of Interframe gap is the minimum required. Additionally, one byte each for Start and Terminate control characters is needed. Ethernet frame in itself requires 1 byte of preamble, 6 bytes each of source and destination address and 4 bytes of FCS. This gives a total overhead of 43 bytes per Ethernet packet.

Table 4-3: **XGEMAC Performance Estimate**

Ethernet Payload Size in Bytes	Overhead	Effective Throughput in Gb/s
64	43/(64 + 43) = 40.1%	5.98
512	43/(43 + 512) = 7.7%	9.2
1024	43/(43 + 1024) = 4.02%	9.59
16384	43/(16384 + 43) = 0.26%	9.9

Measuring Performance

This section shows how performance is measured in the TRD.

Note 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 the PCIe AXI TX interface including TLP headers for all transactions.
PCIe AXI RX Utilization	This register counts traffic on the 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.

Table 4-4: **Performance Registers in Hardware *(Cont'd)***

Register	Description
PCIe AXI RX Payload	This register counts payload for completion transactions downstream, which includes descriptor or data buffer fetch completions.
XGEMAC Statistics registers	The XGEMAC core provides transmit and receive frame statistics.

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.

Performance Observations

This section summarizes the performance measured and the trends seen.

Note: The performance measured on a system at the user end might be different due to PC configuration and PCIe parameter differences.

PCIe-DMA Performance

This section summarizes performance as observed with PCIe-DMA Performance mode (GEN/CHK mode). See Figure 4-1.

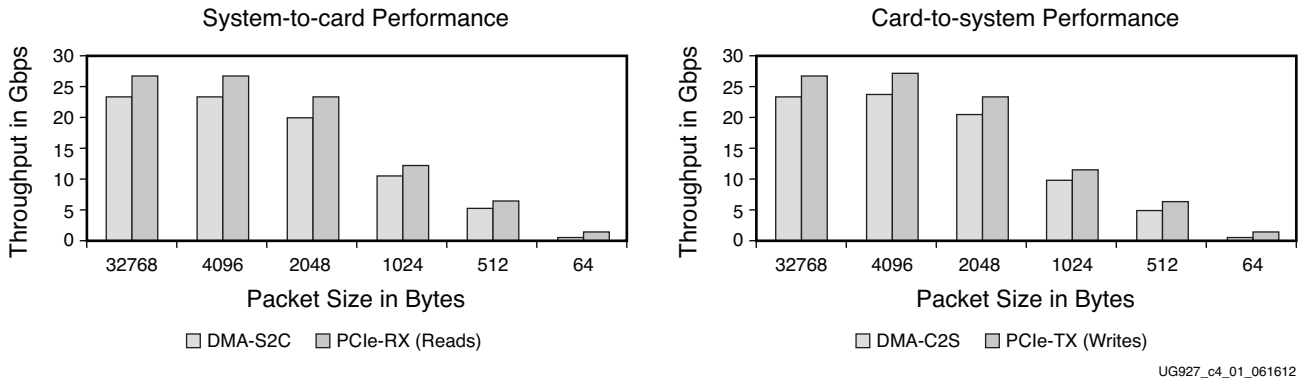


Figure 4-1: **PCIe-DMA Performance**

As can be seen,

- Performance improves with increasing packet size as with the same setup overheads, DMA can fetch more data (actual payload)
- PCIe transaction layer performance (reads and writes) include the DMA setup overheads, whereas DMA performance includes only the actual payload.

Raw Ethernet Performance

This section presents performance as seen with raw Ethernet, that is XGEMAC included but broadcast Ethernet frames generated by the software and no connection to the networking (TCP/IP) stack. See Figure 4-2.

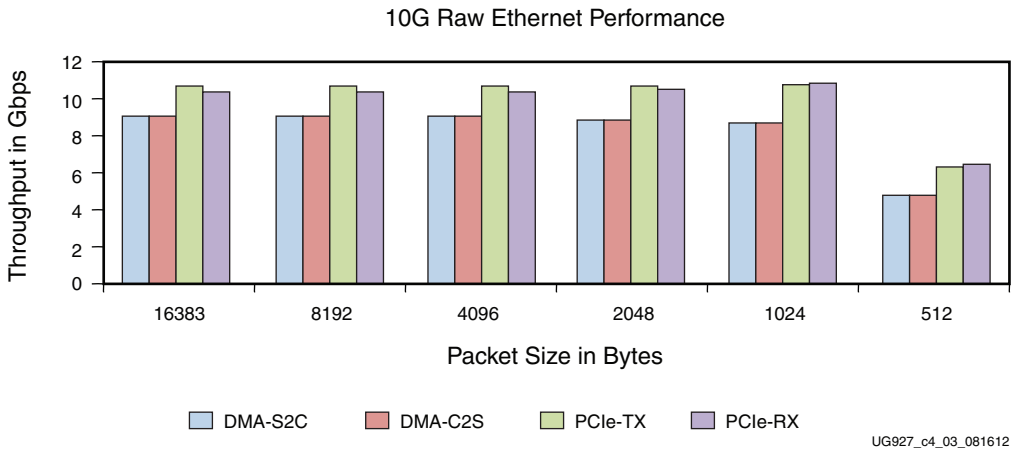


Figure 4-2: **Raw Ethernet Performance**

This depicts that the network path hardware can achieve ~92% throughput on a 10 Gb/s link.

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—Modify software component only (drivers, demo parameters, and so on). The design does not need to be re-implemented.
- Design (top-level only) modifications—Changes to parameters in the top-level of the design. Modify hardware component only (change parameters of individual IP components and custom logic). The design must be re-implemented through the Vivado tool.
- Architectural changes—Modify hardware and software components. The design must be re-implemented through the Vivado tool. Remove or add IP blocks with similar interfaces (supported by Xilinx and its partners). The user needs to do some design work to ensure the new blocks can communicate with the existing interfaces in the framework. Add new IP so as to not impact any of the interfaces within the framework. The user is responsible for ensuring that the new IP does not break the functionality of the existing framework.

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 Compiling Traffic Generator Applications, page 115.

Macro-Based Modifications

This section describes the modifications that 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.

Note: The DMA_BD_CNT in the driver is set to 1999. 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 provided driver directories. This causes the drivers to generate verbose logs.
- Add DEBUG_NORMAL in the Makefiles in the provided driver directories. This causes the drivers to generate informational logs.

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

64-Bit Driver Compilation

The internal scripts detect the operating system and install the drivers accordingly.

- To compile specific 64 bit drivers, add `X86_64` in the Makefile for the corresponding drivers.
- To compile all drivers as 64 bit, initialize and export `OS_TYPE` as 64 bit in `linux/driver/Makefile`.

Driver Mode of Operation

The base DMA driver can be configured to run in either Interrupt mode (Legacy or MSI as supported by the system) 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.

Jumbo Frames

The corresponding change in software requires jumbo frames to be enabled in the Ethernet driver:

- Add ENABLE_JUMBO in the `linux/driver/xxgbeth0/Makefile`
- Add ENABLE_JUMBO in the `linux/driver/xxgbeth1/Makefile`

Enabling JUMBO allows networking stack to send big packets. User can change MTU with standard networking tools for sending bigger packets.

Driver Queue Depth

The depth of queue implemented in the driver can be modified through these changes:

- Edit macro MAX_BUFF_INFO in the `linux/driver/xrawdata0/sguser.c`
- Edit macro MAX_BUFF_INFO in the `linux/driver/xrawdata1/sguser.c`

The depth increase helps in queuing more packets of the receiver side and transmit housekeeping. This helps reduce packet drop when the thread is not able to pool in time.

Design Changes

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

Packet Generator/Checker Behind AXI Virtual FIFO

The packet generator and checker module provided with the TRD can be placed behind the virtual FIFO to estimate the performance of the PCIe-DMA-DDR3 FIFO system. See Figure 5-1.

This requires removing the network path and connecting the virtual FIFO AXI4-Stream interface ports to generator/checker modules.

The same raw data driver works with this hardware change.

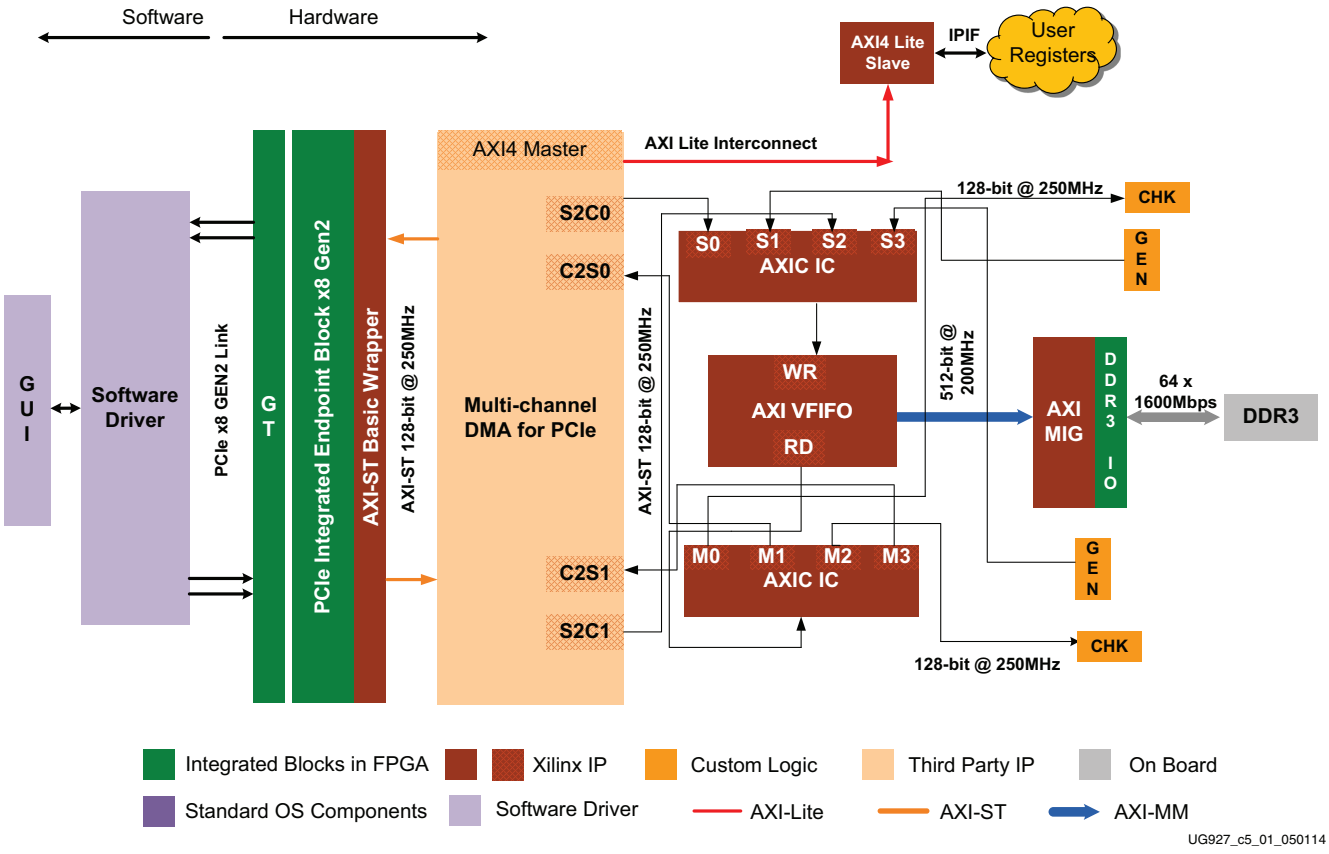


Figure 5-1: **Virtual FIFO with Generator/Checker**

XILINX

Appendix A

Register Description

The appendix describes registers most commonly accessed by the software driver. The registers implemented in hardware are mapped to base address register (BAR0) in PCIe.

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

Table A-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 A-1 shows the layout of registers.

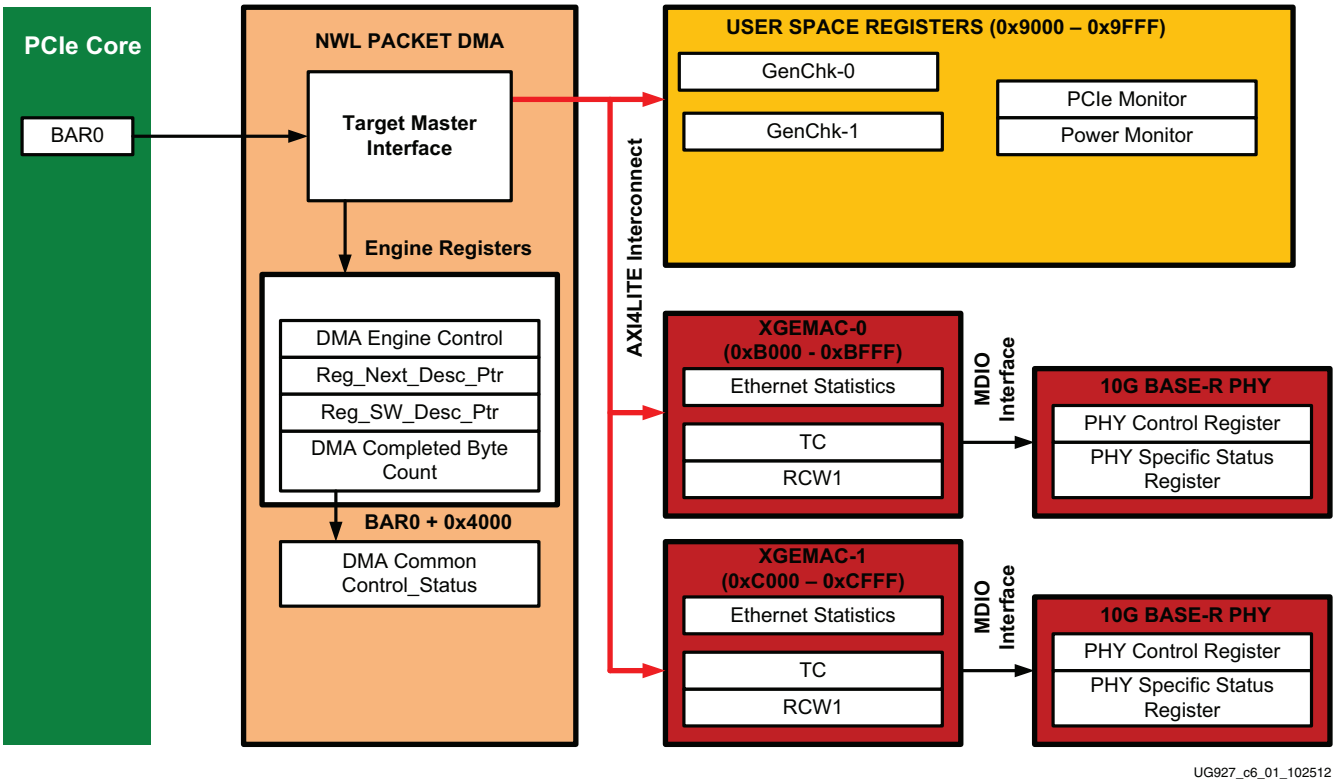


Figure A-1: **Register Map**

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

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

User Logic Register Group	Range (Offset from BAR0)
PCIe performance registers Design version and status registers	0x9000 - 0x90FF
Performance mode GEN/CHK 0 registers	0x9100 - 0x91FF
Performance mode GEN/CHK 1 registers	0x9200 - 0x92FF
Power Monitor registers	0x9400 - 0x94FF
XGEMAC - 0 registers	0xB000 - 0xBFFF
XGEMAC - 1 registers	0xC000 - 0xCFFF

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 user guides.

Channel Specific Registers

The registers described in this section are present in all channels. The address of the register is the channel address offset from BAR0 plus the register offset.

Engine Control (0x0004)

Table A-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 descriptor with interrupt on completion bit set is seen.
3	Descriptor Alignment Error	RW1C	0	This causes interrupt when descriptor address is unaligned and that DMA operation is aborted.
4	Descriptor Fetch Error	RW1C	0	This causes interrupt when descriptor fetch errors, that is, completion status is not successful.
5	SW_Abort_Error	RW1C	0	This is asserted when the DMA operation is aborted by software.
8	DMA Enable	RW	0	Enables the DMA engine and once enabled, the engine compares the next descriptor pointer and software descriptor pointer to begin execution.
10	DMA_Running	RO	0	Indicates DMA is in operation.
11	DMA_Waiting	RO	0	Indicates DMA is 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 the 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 A-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 A-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 the chain that is still owned by the software.
[4:0]	Reserved	RO	5'b00000	Required for 32-byte alignment.

Completed Byte Count (0x001C)

Table A-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 a one second interval.

Common Registers

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

Common Control and Status (0x4000)

Table A-7: **DMA Common Control and 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 of global interrupt enable.
2	Interrupt Pending	RO	0	Reflects the state of DMA interrupt output without considering 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.

Table A-7: **DMA Common Control and Status Register *(Cont'd)***

Bit	Field	Mode	Default Value	Description
5	User Interrupt Active	RW1C	0	Indicates active user interrupt
23:16	S2C Interrupt Status	RO	0	Bit[i] indicates the interrupt status of S2C DMA engine[i]. If the S2C engine is not present, this bit is read as zero.
31:24	C2S Interrupt Status	RO	0	Bit[i] indicates the interrupt status of C2S DMA engine[i]. If the C2S engine is not present, 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. See Table A-8 through Table A-19.

Design Version and Status Registers

Design Version (0x9000)

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

Bit Position	Mode	Default Value	Description
3:0	RO	0000	Minor version of the design
7:4	RO	0001	Major version of the design
15:8	RO	0100	NWL DMA version
19:16	RO	0001	Device-0001 - Kintex-7

Design Status (0x9008)

Table A-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 When software writes to this bit position, the bit is automatically cleared after nine clock cycles.
5:2	RO	1	ddr3_fifo_empty Indicates the DDR3 FIFO and the preview FIFOs per port are empty.
31:30	RO	00	xphy0 and xphy1 link status.

Transmit Utilization Byte Count (0x900C)

Table A-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 A-11: **PCIe 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 A-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 A-13: **PCIe Performance Monitor - Downstream Completion Byte Count Register**

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 A-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 A-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 A-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 A-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 A-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 A-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
Directed Change Link Capability User Register (0x9034)			
0	RO	0	Link Status
1	RO	0	Current link speed 0: 2.5G 1: 5G
3:2	RO	0	Current link width 00: x1 01: x2 10: x4 11: x8
4	RO	0	Link up-configure capable
5	RO	0	Link GEN2 capable
6	RO	0	Link partner GEN2 capable
9:7	RO	000	Initial link width 000: Link not trained 001: x1 010: x2 011: x4 100: x8
Directed Change Link Control User Register (0x9038)			
1:0	RW	00	Directed link speed 00: 2.5 Gb/s 01: 5 Gb/s
4:2	RW	000	Directed Link Width 000: x1 001: x2 010: x4 011: x8
30	RW	0	Initiate speed change
31	RW	0	Initiate width change
Directed Change Link Status User Register (0x903C)			
0	RO	0	Width change done

Table A-19: **PCIe Performance Monitor - Initial PH Credits Register *(Cont'd)***

Bit Position	Mode	Default Value	Description
1	RO	0	Width change error
3:2	RO	00	Negotiated width 00: x1 01: x2 10: x4 11: x8
7	RO	0	Speed change done
8	RO	0	Speed change error
9	RO	00	Negotiated speed 00: 2.5 Gb/s 01: 5 Gb/s

Power Monitoring Registers

Table A-20 lists power monitoring registers.

Table A-20: **Power Monitoring Registers**

Bit Position	Mode	Default Value	Description
VCCINT Power Consumption (0x9040) [TI UCD Address 52 Rail 1]			
31:0	RO	00	Power for VCCINT
VCCAUX Power Consumption (0x9044) [TI UCD Address 52 Rail 2]			
31:0	RO	00	Power for VCCAUX
VCC3.3 Power Consumption (0x9048) [TI UCD Address 52 Rail 3]			
31:0	RO	00	Power for VCC3.3
VADJ Power Consumption (0x904C) [TI UCD Address 52 Rail 4]			
31:0	RO	00	Power for VADJ
VCC2.5 Power Consumption (0x9050) [TI UCD Address 53 Rail 1]			
31:0	RO	00	Power for VCC2.5
VCC1.5 Power Consumption (0x9054) [TI UCD Address 53 Rail 2]			
31:0	RO	00	Power for VCC1.5
MGT AVCC Power Consumption (0x9058) [TI UCD Address 53 Rail 3]			
31:0	RO	00	Power for MGT AVCC
MGT AVTT Power Consumption (0x905C) [TI UCD Address 53 Rail 4]			
31:0	RO	00	Power for MGT AVTT
VCCAUX_IO Power Consumption (0x9060) [TI UCD Address 54 Rail 1]			
31:0	RO	00	Power for VCCAUX_IO

Table A-20: **Power Monitoring Registers *(Cont'd)***

Bit Position	Mode	Default Value	Description
VCC_BRAM Power Consumption (0x9064) [TI UCD Address 54 Rail 2]			
31:0	RO	00	Power for MGT VCC_BRAM
MGT_VCCAUX Power Consumption (0x9068) [TI UCD Address 54 Rail 3]			
31:0	RO	00	Power for MGT_VCCAUX

Performance Mode: Generator/Checker/Loopback Registers for User APP 0

Table A-21 lists the registers to be configured in Performance mode for enabling generator/checker or Loopback mode.

Table A-21: **Registers to be Configured in Performance Mode for User APP 0**

Bit Position	Mode	Default Value	Description
PCIe Performance Module #0 Enable Generator Register (0x9100)			
0	RW	0	Enable traffic generator - C2S0
PCIe Performance Module #0 Packet Length Register (0x9104)			
15:0	RW	16'd4096	Packet length to be generated. Maximum supported is 32 KB size packets. (C2S0)
Module #0 Enable Loopback/Checker Register (0x9108)			
0	RW	0	Enable traffic checker - S2C0
1	RW	0	Enable loopback - S2C0 () C2S0
PCIe Performance Module #0 Checker Status Register (0x910C)			
0	RW1C	0	Checker error Indicates data mismatch when set (S2C0)
PCIe Performance Module #0 Count Wrap Register (0x9110)			
31:0	RW	511	Wrap count Value at which sequence number should wrap around.

Performance Mode: Generator/Checker/Loopback Registers for User APP 1

Table A-22 lists the registers to be configured in Performance mode for enabling generator/checker or Loopback mode.

Table A-22: **Registers to be Configured in Performance Mode for User APP 1**

Bit Position	Mode	Default Value	Description
PCIe Performance Module #0 Enable Generator Register (0x9200)			
0	RW	0	Enable traffic generator - C2S1

Table A-22: **Registers to be Configured in Performance Mode for User APP 1**

Bit Position	Mode	Default Value	Description
PCIe Performance Module #0 Packet Length Register (0x9204)			
15:0	RW	16'd4096	Packet length to be generated. Maximum supported is 32 KB size packets. (C2S1)
Module #0 Enable Loopback/Checker Register (0x9208)			
0	RW	0	Enable traffic checker - S2C1
1	RW	0	Enable loopback - S2C1 () C2S1
PCIe Performance Module #0 Checker Status Register (0x920C)			
0	RW1C	0	Checker error Indicates data mismatch when set (S2C1).
PCIe Performance Module #0 Count Wrap Register (0x9210)			
31:0	RW	511	Wrap count Value at which sequence number should wrap around.

XGEMAC Related User Registers

These registers in Table A-23 are not part of the IP and are registers implemented additionally for the TRD.

Table A-23: **Additional Registers**

Bit Position	Mode	Default Value	Description
XGEMAC0 Address Filtering Control Register (0x9400)			
0	RW	0	Promiscuous mode enable for XGEMAC0
31	RO	0	Receive FIFO overflow status for XGEMAC0
XGEMAC0 MAC Address Lower Register (0x9404)			
31:0	RW	32'hAABBCCDD	MAC address lower
XGEMAC0 MAC Address Upper Register (0x9408)			
15:0	RW	16'hEEFF	MAC address upper
XGEMAC1 Address Filtering Control Register (0x940C)			
0	RW	0	Promiscuous mode enable for XGEMAC1
31	RO	0	Receive FIFO overflow status for XGEMAC1
XGEMAC1 MAC Address Lower Register (0x9410)			
31:0	RW	32'hAAAACCCC	MAC address lower
XGEMAC1 MAC Address Upper Register (0x9414)			
15:0	RW	16'hEEEE	MAC address upper

XILINX

Appendix B

Directory Structure and File Description

Directory Structure

Figure B-1 shows the Kintex®-7 Connectivity TRD directory structure.

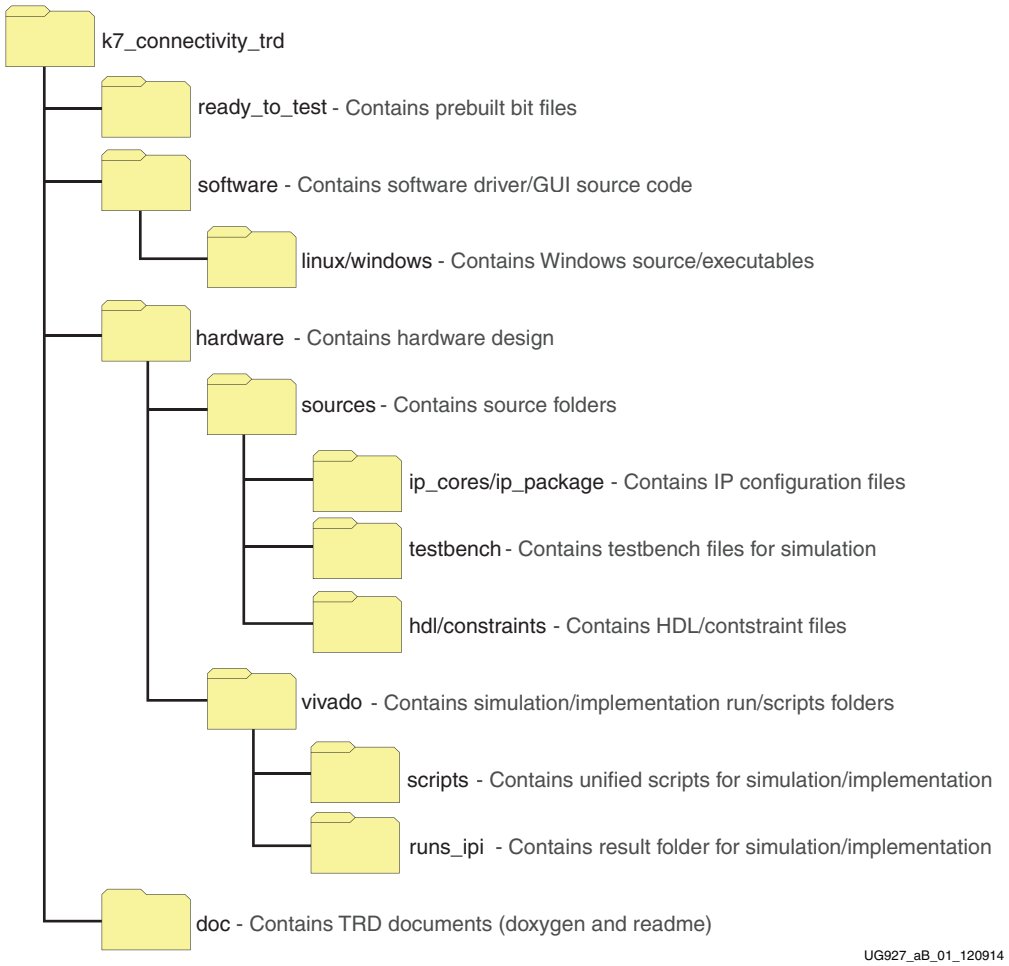


Figure B-1: **Kintex-7 Connectivity Directory Structure**

File Descriptions

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

- The `sources/hdl` folder contains source code deliverable files.
- The `sources/testbench` folder contains test bench related files for simulation.
- The `vivado/scripts` folder contains implementation and simulation scripts for the design for both Windows and Linux operating systems in command line and Vivado Design Suite GUI mode.
- The `sources/ip_package` and `sources/ip_cores` folders contain Xilinx IP cores required for this design and also the DMA netlists.

The doc folder contains this TRD documentation:

- Doxygen generated HTML files for software driver details
- Readme file
- Targeted Reference Design Documentation Advisory

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

The `linux` folder contains the software design deliverables.

The `windows` folder contains the `setup.exe` files for Windows operating system

- The `driver` folder contains these subdirectories:
  - `xrawdata0` contains raw datapath driver files for path 0.
  - `xrawdata1` contains raw datapath driver files for path 1.
  - `xgbeth0` contains 10G Ethernet driver files for path 0.
  - `xgbeth1` contains the 10G Ethernet driver files for path 1.
  - `xdma` contains the xdma driver files.
  - `include` contains the include files used in the driver.
  - `makefile` contains files for driver compilation.
- The `gui` folder contains the Java source files and executable file for running the control and monitor GUI.
- The `linux` folder contains various scripts to compile and execute drivers.

Other files in the top-level directory include:

- The `readme` file, which provides details on the use of simulation and implementation scripts.
- The `quickstart.sh` file, which invokes the control and monitor GUI for Linux.
- The `quickstart.bat` file which invokes the control and monitor GUI for the Windows operating system.

XILINX

Appendix C

Software Application and Network Performance

This appendix describes the software application compilation procedure and private network setup for the Linux operating system.

Note: The traffic generator needs the CPP (C++) compiler, which is not shipped with live OS. It needs additional installation for compilation. Likewise, Java compilation tools are not shipped as part of *LiveDVD*. So GUI compilation needs additional installations. The source code is provided for the user to build upon this design. For TRD testing, recompiling the application or GUI is not recommended.

Compiling Traffic Generator Applications

This section provides steps for traffic generator compilation. The source code for the design (`threads.cpp`) is available under the directory `k7_connectivity_trd/software/linux/gui/jnilib/src`.

The user can add debug messages or enable **log verbose** for verbosity to aid in debug.

Note: Any changes in data structure lead to GUI compilation, which is not recommended.

To compile the application traffic generator:

1. Open a terminal window.
2. Navigate to the `k7_connectivity_trd/linux/gui/jnilib/src` folder.
3. At the prompt, type:

   `$` **./genlib.sh**

The `.so` files (shared object files) are generated in the same folder. Copy all `.so` files to the `k7_connectivity_trd/software/linux/gui/jnilib` folder.

User can enable **log verbose** messages by adding a -DDEBUG_VERBOSE flag to `genlib.sh`. Enabling log verbose makes debug simpler (if needed).

Private Network Setup and Test

This section explains how to try network benchmarking with this design. The recommended benchmarking tool is netperf which operates in a client-server model. This tool can be freely downloaded and is not shipped as part of *LiveDVD*. Install netperf before proceeding further.

Default Setup

In the setup connected to same machine, the network benchmarking tool can be run as follows:

1. Follow the procedure to install Application mode drivers and try ping as documented in Installing the Linux Device Drivers, page 18. The two interfaces are ethX and eth(X+1) with IP addresses of 10.60.0.1 and 10.60.1.1, respectively.
2. Disable the firewall to run netperf.
3. Open a terminal and type:

   $ **netserver -p 5005**

   This sets up the netserver to listen at port 5005.
4. Open another terminal and type:

   $ **netperf -H 10.60.0.1 -p 5005**

   This runs netperf (TCP_STREAM test for 10 seconds) and targets the server at port 5005.
5. To repeat the same process for 10.60.1.1 IP, set up netserver at a different port, for example, 5006, and repeat the previous steps.

Peer Mode Setup and Test

This section describes steps to set up a private LAN connection between two machines for 10G Ethernet performance measurement. Figure C-1 shows the private LAN setup in peer mode.

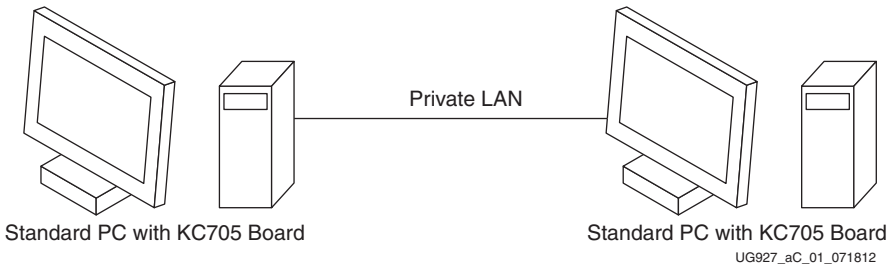


Figure C-1: **Private LAN Setup**

To set up a private LAN connection:

1. Connect two machines that contain the KC705 board and connect the fiber optic cable between the FMCs.

   Connect the fiber cable in a 1:1 manner, that is, FMC channel 2 connected together and FMC channel 3 connected together.

   For this procedure, these machines are called *A* and *B*.
2. Run the `quickstart.sh` script provided in the package. Select the **`Application`** mode with **`Peer to Peer`** option. Click **`Install`**. This installs the Application mode drivers.

3. After installing the Application mode driver at both ends using the steps documented in Installing the Linux Device Drivers, page 18):
   a. On end A, change the MAC address using ifconfig:

      $ **ifconfig ethX down**

      $ **ifconfig ethX hw ether 22:33:44:55:66:77 172.16.64.7 up**

   b. For the corresponding interface on end B, set the IP to be in the same subnet:

      $ **ifconfig ethX 172.16.64.6 up**

   c. Follow the same steps for the interface eth(X+1). Change the MAC address at one end and assign the IP address to be in a different subnet as the subnet assigned for ethX.
4. Try ping between the machines.
5. Make one end a server. On a terminal, invoke netserver as shown:

   $ **netserver**

6. Make the other end a client. On a terminal, run netperf:

   $ **netperf -H <*IP-address*>**

This runs a ten second TCP_STREAM test by default and reports outbound performance.

Note: Connecting the two FMC channels at each end in 1:1 mode ensures that ethX on one machine connects to ethX on another machine. If the order of connection is changed, ethX of one machine gets connected to eth(X+1), which means setting up MAC and IP addresses has to be handled appropriately based on the connections made.

Appendix D

Troubleshooting

This section lists selected self-help tips for when things do not work as expected. This section is not an exhaustive troubleshooting guide. It is based on the following assumptions:

- The user has followed instructions as explained in Chapter 2, Getting Started.
- The user has ensured the PCIe link is up and that the endpoint device is discovered by the host and can be seen with **lspci**.
- The LEDs indicate various link status as described in Chapter 2, Getting Started.

Table D-1 lists troubleshooting tips and possible corrective actions.

Table D-1: **Troubleshooting Tips**

Problem	Possible Resolution
Performance is low.	Check if the design linked at x8 5 Gb/s rate
Link width change doesn't work.	Check the message log. It t is possible that the motherboard slot being used is not upconfigure capable.
Power numbers do not populate in the GUI.	Power cycle the board. The cause of this problem is PMBus signals get into an unknown state during FPGA configuration and the only way to bring PMBus back to a working state is to power cycle the board to reset the UCD9248 part.
Test does not start while using an Intel motherboard.	Check dmesg command if user is getting nommu_map_single then user can bring up by followings ways. • If OS is installed on the hard disk, the user can edit the `/etc/grub2.cfg` file and add **mem=2g** to kernel options. • While using LiveDVD, stop LiveDVD at the boot prompt and add **mem=2g** to kernel boot up options.
Performance numbers are very low and the system hangs upon un-installing the TRD driver.	This problem might be noticed in Intel motherboards. • If OS is installed on the hard disk, edit the `/etc/grab2.cfg` to add Add **IOMMU=pt64** to kernel boot up options.
Drivers cannot be installed.	An error message pops up when trying to install if there is a problem with the installation. The popup message mentions the reason, but the user can select the **`View Log`** option for a detailed analysis. This action creates an `open driver_log` file.

XILINX

Appendix E

Building the Windows Software

Required Tools

- Microsoft Visual Studio 2012
- Microsoft Windows Driver Kit Version 7.1.0
- Code signing certificate (required if creating and distributing your own version)
- InstallShield Professional Edition

Xilinx provides a framework to build the TRD Windows driver project using Microsoft Visual Studio 2012.

The Windows drivers are built using the Windows 7 Windows Driver Kit (WDK) version 7.1.0. The current WDK, version 7600.16385.1 *MUST* be used. It is available here. at no cost.

Note: It is an the WDK is an ISO image and must be burned to a CD/DVD in to install it.

It is necessary to code sign the drivers to allow the drivers to be installed on Vista and Windows 7 64-bit versions. This requirement can be disabled by pressing the **F8** key before windows starts and selecting **Disable Code Signing Enforcement**.

Note: Code signing enforcement must be disabled every time Windows is started when using drivers that are not code signed.

Code-signing the drivers requires a Software Publisher Certificate (SPC) from a Certification Authority (CA). Contact a CA to obtain the SPC and instructions for installing a private key on the signing computer.

To build a setup for distribution, InstallShield Professional Edition from Flexera Software is recommended.

Batch File Modifications

Note: If the WDK is installed to any location other than `C:\WinDDK\7600.16385.1` the batch files `build_XBlock.bat`, `build_XDMA.bat`, and `build_XNet.bat` must be modified as described in this section.

For each batch file `build_XBlock.bat`, `build_XDMA.bat`, and `build_XNet.bat`:

1. Open the batch file and locate the line:
   `Set DDKROOT=C:\WINDDK\7600.16385.1`
2. Change the path to point to the directory where the WDK is located.

XILINX®

Appendix F

Enabling Debugging with the Windows Driver

To debug the Windows drivers, the free (release) build version of the drivers must be replaced by the checked (debug) version. This replacement is done by copying the checked build version over the already-installed free build version located in the `Windows\System32\Drivers` directory. After the copy is complete a reboot is necessary to reload the image.

Debug messages can be viewed by using the Kernel Debugger **WinDbg** or by using the application **DbgView**. Search the internet for DbgView. It is normally found in the Microsoft Sys internals website and is free to download.

To enable debug messages:

1. Open **Regedit** on the machine running the driver
2. Go to `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager`. If the **Debug Print Filter** key does not exist create it.
3. In the `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Debug` Print Filter section add a 32-bit ULONG value called IHVDRIVER for the XDMA driver and XBlock driver and/or IHVNETWORK for the XNet driver.

   Set the value to:

   - `1` for ERRORs only
   - `2` for ERRORs and WARNINGs
   - `3` for ERRORs, WARNINGs, and information
   - `4` for ERRORs, WARNINGs, information, and trace messages
   - `5` for ERRORs, WARNINGs, information, trace messages and verbose messages
   - `0xFFFFFFFF` for any/all messages

The Windows driver uses a macro for debug statements. The macro is called DEBUGP which consists of several DbgPrintEx statements. This macro is defined in `trace.h`. To use the Microsoft tracing facility, search and replace all DEBUGP macros with DEBUGP. This replacement is necessary because of the way the tracing is implemented. It cannot be used in a macro for easy substitution and tracing is difficult to setup and use. The debug print filter method is easier to use than tracing.

For example replace: **`DEBUGP(DEBUG_INFO`**
with: **`DEBUGP(TRACE_LEVEL_INFORMATION, DBG_INIT`**

XILINX

Appendix G

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website.

For continual updates, add the Answer Record to your myAlerts.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

The most up to date information related to the KC705 Evaluation Kit and its documentation is available on these websites:

Kintex-7 FPGA KC705 Evaluation Kit

Kintex-7 FPGA Connectivity Kit Documentation

Kintex-7 FPGA Connectivity Kit (AR 50555)

These Xilinx documents and sites provide supplemental material useful with this guide:

1. *Vivado Design Suite Migration Methodology Guide* (UG911)
2. *Vivado Design Suite Implementation User Guide* (UG904)
3. *7 Series FPGAs Integrated Block for PCI Express v2.2 Product Guide* (PG054)
4. *Synthesis and Simulation Design Guide* (UG626)
5. *Vivado Design Suite Logic Simulation User Guide* (UG900)
6. *LogiCORE IP 10-Gigabit Ethernet MAC Product Guide* (PG072)
7. *LogiCORE IP 10-Gigabit Ethernet PCS/PMA Product Guide* (PG068)
8. *Understanding Performance of PCI Express Systems* (WP350)
9. *KC705 Evaluation Board for the Kintex-7 FPGA User Guide* (UG810)
10. *7 Series FPGAs GTX Transceivers User Guide* (UG476)
11. *7 Series FPGAs Memory Interface Solutions User Guide* (UG586)
12. *Kintex-7 FPGA Base Targeted Reference Design Getting Started Guide* (UG883)
13. *LogiCORE IP AXI4-Stream Interconnect Product Guide* (PG035)

14. *LogiCORE IP AXI Virtual FIFO Controller Product Guide* (PG038)
15. *Kintex-7 FPGA Connectivity Kit Getting Started Guide (Vivado Design Suite)* (UG929)

These external websites provide supplemental material useful with this guide:

16. Northwest Logic
    PCI Express® Solution IP, including DMA Back-End Core
17. Fedora Project
    Fedora operating system information and downloads
18. Linux Kernel Organization
    The Linux kernel archives
19. Silicon Labs
    CP2103 VCP Drivers
20. Ayera Technologies
    TeraTerm Pro Enhanced Telnet/SSH2 Client terminal program
21. Faster Technology, LLC
    FM-S14 Quad SFP/SFP+ transceiver VITA 57 FMC module
22. Avago Technologies
    AFBR-703SDZ 10 Gb/s Ethernet, 850 nm, 10GBASE-SR, SFP+ Transceiver
23. Xilinx AXI Interconnect IP
    AXI Interconnect IP product page
24. Amphenol Corporation
    LC-LC Duplex 10 Gb/s Multimode 50/125 OM3 Fiber Optic Patch Cable - 2 x LC Male to 2 x LC Male