Spartan-6 FPGA Connectivity Targeted Reference Design with AXI4 Protocol

User Guide

UG399 (v2.1) January 24, 2013

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/05/10	1.0	Initial Xilinx release.
12/21/10	1.1	Removed pre-production nomenclature from document title page.
03/01/11	1.2	Starting with this release, Project Navigator flow is no longer supported. PlanAhead™ support is provided in its place. In Chapter 1, Introduction to the Reference Design, expanded The Targeted Reference Design.
01/18/12	2.0	Included references to Fedora 10 Linux and Windows XP throughout document. In Preface, updated Additional Documentation, page 10 and Additional Support Resources, page 10. In Chapter 1, updated Features, page 13 to include reference to Fedora 10 Linux and Windows XP operating systems. In Chapter 2, updated operating system details under Hardware Test Setup Requirements, page 15. Revised Hardware Bring Up, step 4, and added step 5. Added heading Testing with Linux Operating System, page 17. Updated Figure 2-6, Figure 2-7, Figure 2-8, Figure 2-16, and Figure 2-46. Added Testing with the Windows Operating System, page 32 through page 40. Added Launching Performance Monitor, page 41, Viewing Drivers, page 41, Configuration, page 42 through page 46. Added Windows configuration instructions under Autonegotiation on page 47, Promiscuous Mode on page 48, and Checksum Offload on page 49. Added heading Linux Flow, page 51. Revised PlanAhead-Based Flow on page 54 to include information about licensing and 12-hour evaluation netlist. Updated Figure 3-14, page 85. In Chapter 5, added reference to Appendix F, Compiling Windows Drivers in section Software-Only Modifications, page 99. Added Windows configuration instructions under Jumbo Frames on page 101 and PCIe Vendor and Device ID on page 102. In Appendix B, updated figure Figure B-1, page 119. In Appendix D, Added headings Linux Flow and Netperf, page 125 and Windows Flow and Iperf, page 126 through page 127. Added Appendix F, Compiling Windows Drivers.
01/24/13	2.1	Removed statement about a node-free hardware evaluation license from IP Cores with TRD, page 52 and deleted the license folder from the directory structure shown in Figure B-1, page 119.

Table of Contents

Chapter 3: Functional Description

About This Guide

The Spartan®-6 FPGA Connectivity Targeted Reference Design (TRD) delivers all the basic components of a targeted design platform for the connectivity domain in a single package. Targeted Design Platforms from Xilinx provide customers with simple, smart design platforms for the creation of FPGA-based solutions in a wide variety of industries.

This user guide details a targeted reference design developed for the connectivity domain on a Spartan-6 FPGA. The aim is to accelerate the design cycle and enable FPGA designers to spend less time developing the infrastructure of an application and more time creating a unique value-add design.

Guide Contents

This document contains the following chapters:

- Chapter 1, Introduction to the Reference Design provides an introduction to the targeted reference design and outlines its features.
- Chapter 2, Getting Started provides a quick-start guide to help the user get started with the hardware setup and simulation.
- Chapter 3, Functional Description provides a detailed architectural description of the hardware and software driver implemented. A separate software programming manual listing the software APIs is delivered with the software source.
- Chapter 4, Performance Estimation provides a detailed report on theoretical estimation and analysis of the design performance.
- Chapter 5, Designing with the TRD Platform provides suggested variations and add-ons to the platform to implement different functions.
- Appendix A, Register Description details certain registers programmed by the software driver. This section provides a comprehensive register programming manual.
- Appendix B, Directory Structure details the directory structure and the organization of various files and folders.
- Appendix C, AXI4 Interface Migration Considerations for TRD describes the migration considerations for users starting on AXI4.
- Appendix D, Setting Up a Private LAN provides instructions on setting up a private LAN connection to enable testing with Netperf.
- Appendix E, Troubleshooting provides tips on areas to look for when debugging a design.
- Appendix F, Compiling Windows Drivers, describes how to compile Windows drivers using the Windows Driver Kit.

Additional Documentation

The following documents have been used as a reference to develop this TRD.

- UG654, *Spartan-6 FPGA Integrated Endpoint Block for PCI Express User Guide*
- Northwest Logic Packet DMA Backend Core User Guide
  - Product Sheet
- DS759, *AXI Ethernet (v1.00.a) Data Sheet*
- UG388, *Spartan-6 FPGA Memory Controller User Guide*
- MARVELL PHY 88E1111 Data Sheet
- UG526, *SP605 Hardware User Guide*
- Fedora Project
  - Netperf v2.4 - Manual
  - GTK+ 2.0 - Manual
- Visual Studio - http://www.microsoft.com/visualstudio/en-us/home
- Windows Device Driver Kit - http://msdn.microsoft.com/en-us/windows/hardware/gg487421

Additional Support Resources

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

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

XILINX®

Chapter 1

Introduction to the Reference Design

The Targeted Reference Design

Targeted Reference Designs (TRD) provide the customers with a starting point for their application development. The TRDs accelerate the customers' design cycle and enable them to invest their resources to add their unique value to the product rather than building the entire infrastructure from scratch.

The Spartan®-6 FPGA Connectivity TRD simplifies customer application development for using:

- PCI Express® protocol
- Ethernet protocol
- Memory Interfaces

The TRD demonstrates the key integrated components in a Spartan-6 FPGA, namely the integrated Endpoint block for PCI Express, the transceivers, and the memory controller working together in an application along with additional IP cores including the third-party (Northwest Logic) Packet Direct Memory Access (DMA) engine, AXI_Ethernet IP, and the Xilinx® Memory Interface Generator (MIG) in the CORE Generator™ tool.

The TRD deliverables include source code deliverables for hardware RTL Design and software package — device drivers, application, and Graphical User Interface (GUI).

This chapter introduces the targeted reference design (with AXI4 protocol) for Spartan-6 FPGAs and outlines its features.

Introduction

The block diagram of the TRD with AXI4 protocol is shown in Figure 1-1, page 12. This design is a PCI Express v1.1 compliant x1 Endpoint block showcasing these independent applications.

- Network interface card (network path), providing either:
  - GMII mode using an external Ethernet PHY (typically used to connect to copper networks) or
  - 1000BASE-X mode using the GTP transceivers on the FPGA (typically used to connect to optical fiber Ethernet networks).

  This card allows the designer to connect to an external network and run networking applications including browsing web pages, telnet, and ftp sites.
- External memory interface over PCI Express—also referred to as the memory path.

  This application showcases data movement between system memory and DDR3 SDRAM through the Spartan-6 FPGA.

The TRD uses a bus-mastering packet Direct Memory Access (DMA) engine to offload processor data transfer overhead. The DMA works in conjunction with the integrated Endpoint for PCI Express and enables high-speed data movement between system memory and the FPGA.

The TRD framework is built with the integrated Endpoint and DMA blocks forming the foundation of an entire platform. The network path and memory path are two applications developed around this foundation.

The TRD uses the AXI_Ethernet IP core along with the AXI4 interface Lite master and slave on the network path. The host system is connected to the integrated Endpoint and uses the AXI4 Lite interface to configure the registers of the AXI_Ethernet IP. The Ethernet applications are demonstrated in two modes:

- GMII mode using the external PHY onboard
- 1000BASE-X mode using the Spartan-6 FPGA transceivers through an additional 1000BASE-X PCS-PMA IP core (inclusion of this IP is based on selection of a compile-time parameter for PHY)

The memory path showcases the integrated memory controller on a Spartan-6 FPGA. The controller communicates with onboard DDR3 SDRAM through the MIG. Using the connected components, data can be written to and read back from the external memory (MIG IP with an AXI4 interface is used to interface to the memory controller block).

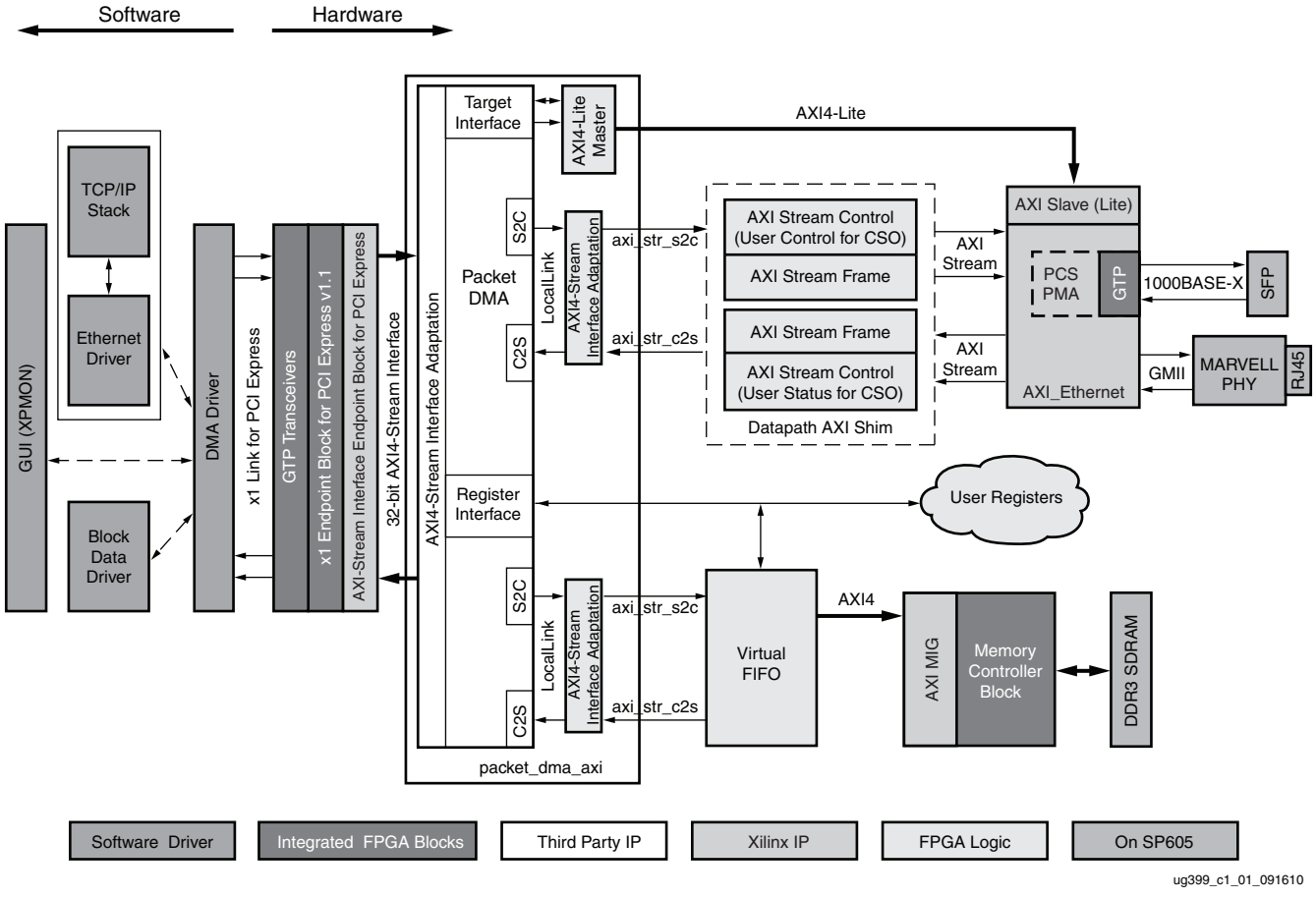


Figure 1-1: **Top-Level Design Overview**

Features

The design uses the AXI4 interface (in its various modes such as memory mapped, lite, and stream). The features provided by the targeted reference design are classified as either:

- Base Features: Features of the base components of the design including DMA and the integrated Endpoint block for PCI Express.
- Application Features: Features supported by each application developed on top of the base features.

The software driver source code for the Fedora 10 Linux operating system (32-bit) and the Windows XP operating system (32-bit, service pack 3) platforms is also provided. The layered implementation of the software driver enables the designer to choose any layer (pertaining to specific hardware functionality) and customize it to specific requirements.

Base Features

This section lists the base features of the integrated Endpoint block for PCI Express and Packet DMA which form the backbone of the entire design.

- PCI Express v1.1 compliant x1 Endpoint block
  - One lane operation with GTP transceiver at a line rate of 2.5 Gb/s/direction
  - MSI and Legacy interrupt support
  - Transaction Layer Packet (TLP) statistics engine for PCI Express
  - Optional high-performance mode utilizing extra block RAMs
- Bus Mastering Packet DMA
  - Multichannel operation
  - Packetized interface (similar to LocalLink)
  - Scatter Gather operation
  - DMA performance engine
  - Full-duplex operation (independent transmit and receive channels)

An AXI4 protocol adaptation layer is built around a Northwest Logic-provided Packet DMA. The AXI4 protocol adaptation layer provides an AXI4-Stream interface in place of a packetized user interface, and an AXI4-Lite interface in place of a target interface. The adaptation layer provides ease of plug and play to other AXI4 protocol compliant IP cores.

Application Features

The various application features are:

- Network Path Features (enabled by AXI_Ethernet IP)
  - Ethernet address filtering
  - Ethernet statistics engine
  - TCP/UDP checksum offload
  - Auto-negotiation
  - Jumbo frames
  - Optional 1000BASE-X mode

- Memory Path Features (enabled by MIG IP supporting AXI4 interface)
  - Integrated DDR3 controller providing up to 667 Mb/s performance
    - 16-bit single-component memory interface providing up to 10.672 Gb/s aggregate bandwidth
  - Reusable virtual FIFO interface with programmable depth

Resource Utilization

Table 1-1 and Table 1-2 list the resource utilization obtained from the map report during implementation phase. The XC6SLX45T-3-FGG484 is the target FPGA.

Note: The utilization numbers reported are obtained with the default options as provided in the top-level design. A change in default options will result in a change in utilization. The transceiver utilization is reported for the GTPA1_DUAL; one transceiver each is utilized from the GTPA1_DUAL (not both).

Table 1-1: **Resource Utilization—GMII Mode with External PHY**

Resource	Utilization	Total Available	Percentage Utilization
Slice registers	23,079	54,576	42
Slice LUTs	17,273	27,288	63
IOB	82	296	27
RAMB16BWER	35	116	30
DCM	1	8	12
PLL_ADV	2	4	50
BUFG	9	16	56
GTPA1_DUAL	1	2	50

Table 1-2: **Resource Utilization—1000BASE-X Mode**

Resource	Utilization	Total Available	Percentage Utilization
Slice registers	23,527	54,576	43
Slice LUTs	17,589	27,288	64
IOB	57	296	19
RAMB16BWER	35	116	30
DCM	1	8	12
PLL_ADV	2	4	50
BUFG	8	16	50
GTPA1_DUAL	2	2	100

The 1000BASE-X mode utilizes an additional IP (PCS-PMA) core, which increases the utilization of LUTs. Because the 1000BASE-X mode provides a serial interface as compared to a parallel interface with GMII mode, a corresponding reduction is seen in IOB usage. The 1000BASE-X mode uses an additional GTP transceiver.

XILINX®

Chapter 2

Getting Started

Introduction

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. It provides step-by-step instructions for testing the design in hardware.

Requirements

This section lists the minimum prerequisites of hardware testing and simulation.

Hardware Test Setup Requirements

The prerequisites for testing the design in hardware are:

- SP605 board with an XC6SLX45T-3 device.
- ISE® software design tools.
- USB JTAG for FPGA programming.
- RJ45 cable and network connectivity.
- PC with PCIe® v1.1 compliant slot, with at least 1GB of RAM and a 1MB cache.
- Operating system:
  - Fedora 10 Linux OS (kernel version 2.6.27), provided as LiveCD. Netperf v2.4 is optional.
  - Windows XP OS (Service Pack 3). iperf is optional.

Simulation Requirements

This section lists the prerequisites for simulation:

- ISE software design tools (system or embedded edition with EDK tools installed)
- ModelSim v6.5c or later

Hardware Test Setup

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

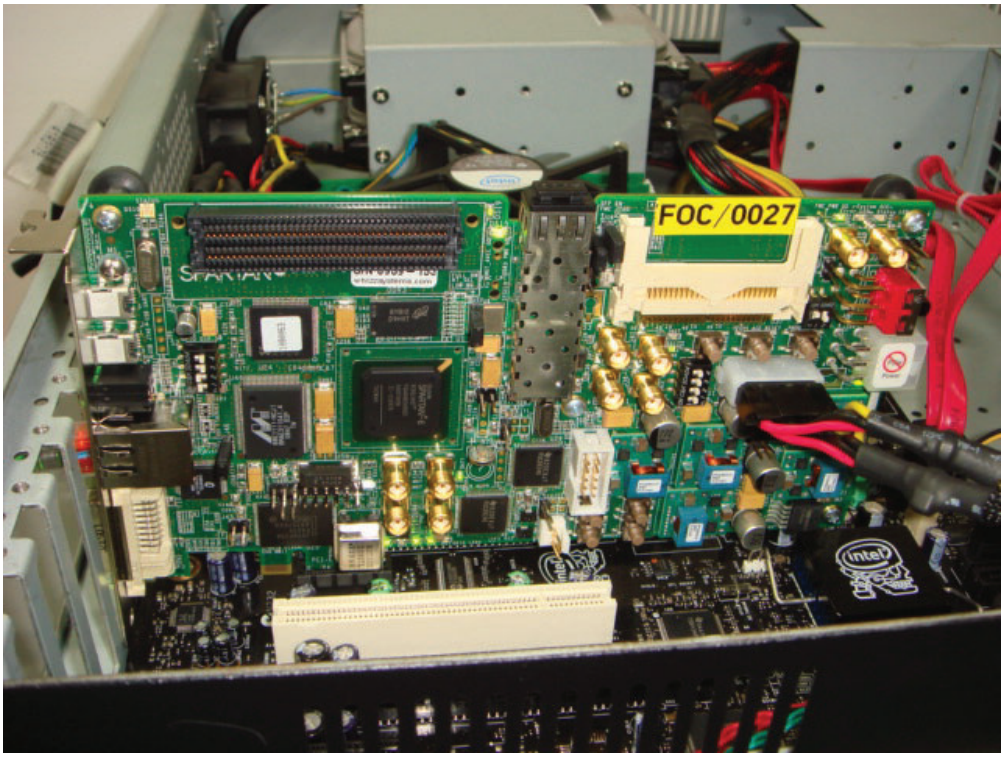
All procedures listed require super-user access on Linux machines. When using the Fedora 10 Linux OS LiveCD provided with the kit, super-user access is granted by default due to the way the kernel image is built; if not using LiveCD, contact your system administrator for super-user access.

Note: The changes made during LiveCD environment are not saved. For example, if the driver files were copied on the desktop after booting with LiveCD, they do not exist anymore after the system is shutdown and booted again with LiveCD. This is because the root file system for OS on LiveCD is created on system RAM and is not permanently on the hard disk.

Hardware Bring Up

This section lists the detailed steps for hardware bring-up.

1. With the host system switched to off, insert the SP605 board in the PCI Express® slot through the PCI Express x1 edge connector.
2. Connect the 12V ATX power supply 4-pin disk drive type connector to the board and also connect an Ethernet LAN cable in the RJ-45 slot provided. Do not power the board with both an external supply and the ATX supply at the same time.
3. Make sure the connections are secure so as to avoid loose contact problems. Power on the system.



UG399_c2_01_091510

Figure 2-1: **Board Setup**

4. Verify the hardware status through visual inspection. Make sure that the following indicators glow after powering on the system. Figure 2-2 indicates the location of the indicators.
   - PCI Express link up

- Ethernet link status (seen on the left-hand side in Figure 2-2, side-view of the board)
- Memory calibration complete

Figure 2-2 shows the entire design as mapped to hardware and the various components used on the SP605 board. The figure also highlights the visual indicators.

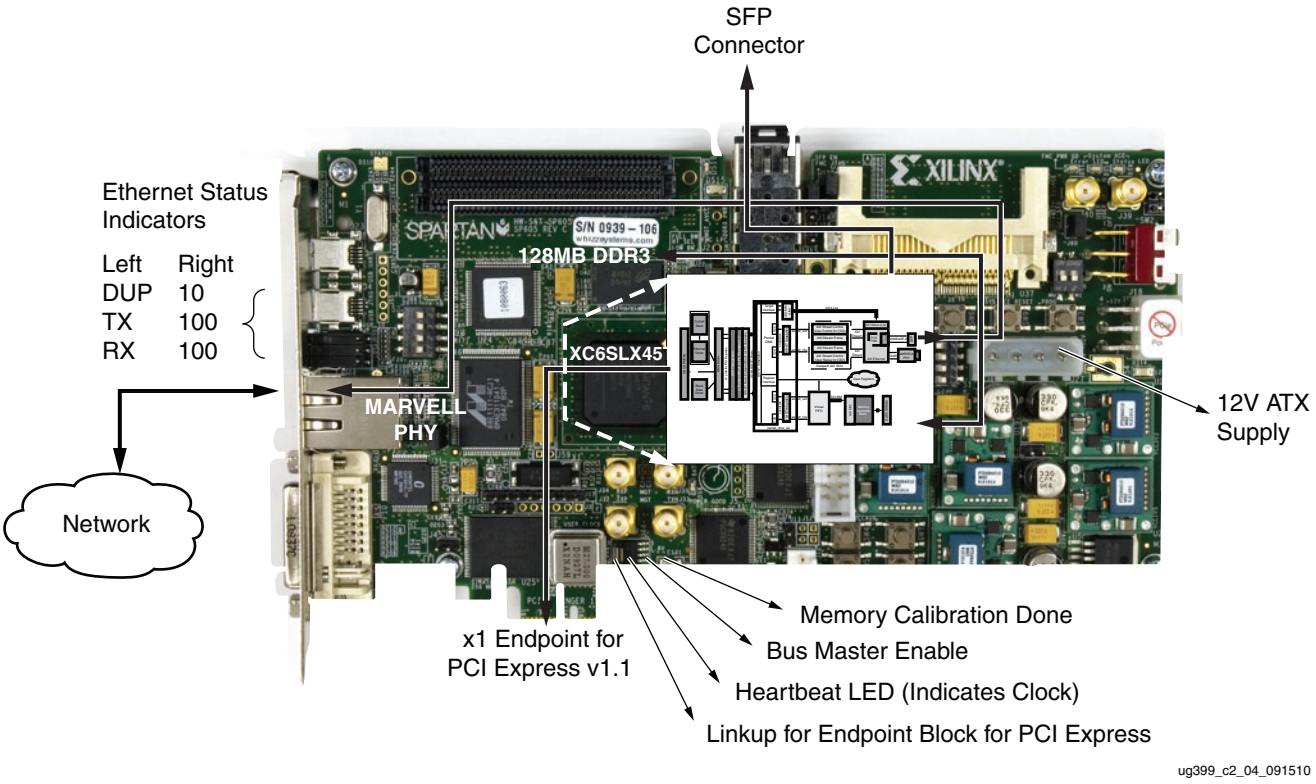


Figure 2-2: **Design on SP605 Board**

5. When hardware setup is complete, either the Windows or Linux OS platform can be used for testing. To test with the Linux OS, go to Testing with Linux Operating System. If using a PC with Fedora 10 installed, go to step 3, page 19. To test with Windows, Testing with the Windows Operating System, page 32.

Testing with Linux Operating System

1. When using the Fedora 10 LiveCD provided in the SP605 kit, configure the desktop PC to boot from the CD ROM in the BIOS while the system starts. The Fedora 10 LiveCD provided in the kit is for Intel compatible PCs. The CD contains a complete bootable Fedora 10 Live environment and packages required for the targeted reference design demonstration. While the system boots from the CD, the screen is as shown in Figure 2-3.

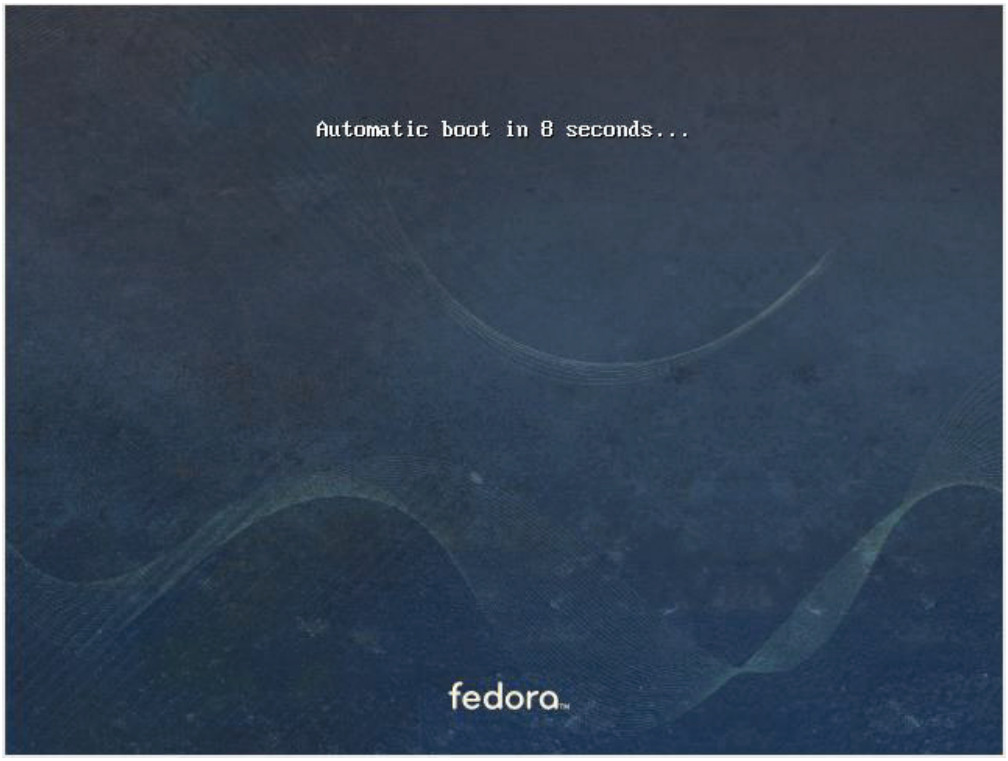


Figure 2-3: **Fedora 10 LiveCD Boot Screen**

2. On the login screen (Figure 2-4), follow the instructions and proceed.

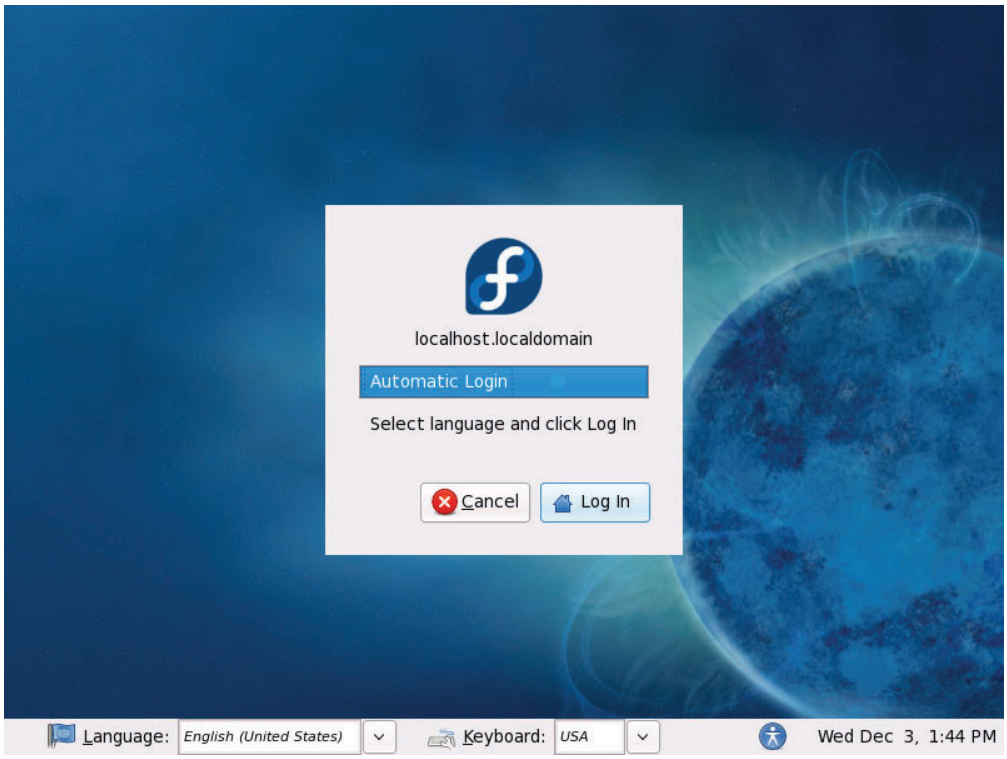


Figure 2-4: **Fedora 10 LiveCD Automatic Login**

3. Check if the Endpoint block for PCI Express is detected by the system after the system boots by opening a terminal window (go to **Application**→ **System Tools**→ **Terminal**) and perform the following:

   At the terminal command line:

   ```
   $ lspci
   ```

   One of the entries should show the following information:

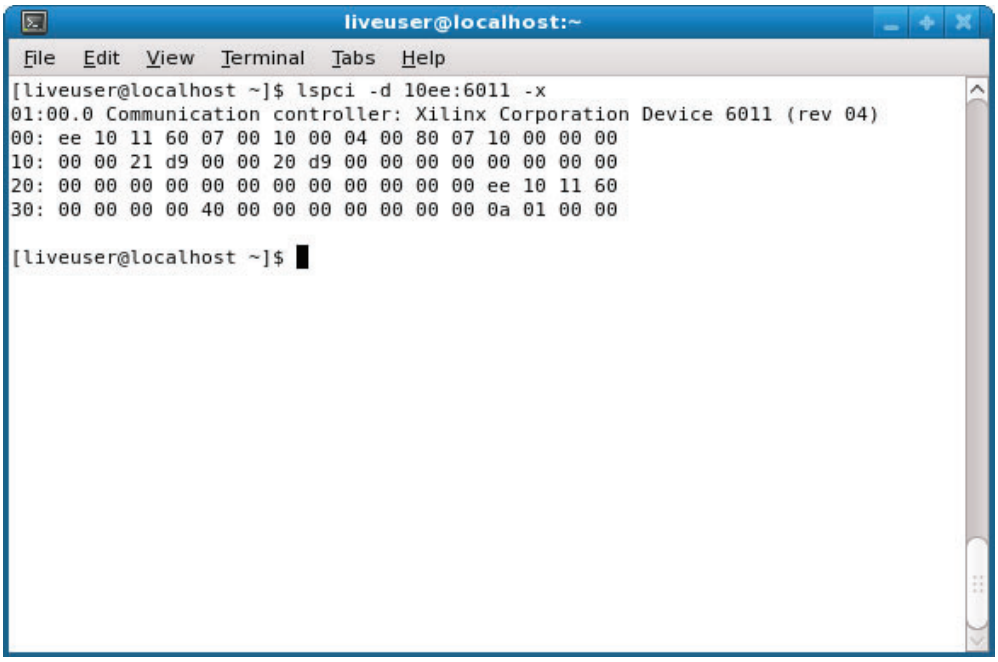
   ```
   01:00.0 Communication controller: Xilinx Corporation Device 6011 (rev 04)
   ```

   Where, `01:00.0` shows the PCI Express bus, device, and function number. As details in the output of the `lspci` command can vary from system to system, it is important that the Xilinx device is identified.

   To view details about the device configuration space, perform the following:

   ```
   $ lspci -d 10EE:6011 -v -x
   ```

   where, `10EE` is the vendor ID and `6011` is the device ID for the Endpoint. As details can vary from system to system, it is important to check that the vendor ID (`10EE`) and device ID (`6011`) match.



UG399_c2_05_091510

Figure 2-5: **Ispci Output**

To see more configuration space details type:

```
$sudo lspci -d 10EE:6011 -vvv -xxx
```

Software Bring Up

This section gives steps for bringing up the software driver, application, and using the various scripts provided.

1. After the device is detected, copy the `s6_pcie_dma_ddr3_gbe_axi` folder from the USB flash drive provided in the Connectivity kit to the desktop (or a folder of choice). After the folder is copied, unmount and disconnect the USB drive.

   Double-click on the copied `s6_pcie_dma_ddr3_gbe_axi` folder.

   The screen capture in Figure 2-6 shows the contents of the `s6_pcie_dma_ddr3_gbe_axi` TRD folder.

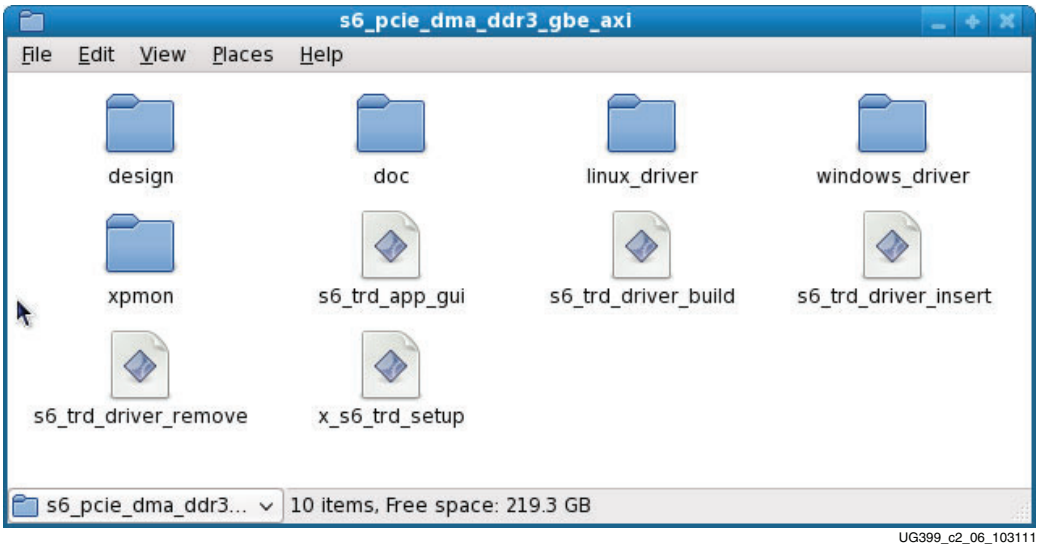


Figure 2-6: **TRD Folder on the USB Flash Drive**

2. This step involves driver compilation and insertion of the kernel modules. Separate steps are defined for a command line user conversant with Linux or for a user preferring button-click operations.

Command Line Mode using Makefile

To compile and insert the driver, type at the command line in the terminal in the driver folder. Navigate to the `s6_pcie_dma_ddr3_gbe_axi/linux_driver/driver` folder and follow the steps.

To clean the area:

```
$ make clean
```

To compile the files and build the kernel objects:

```
$ make
```

To insert the kernel object files:

```
$ make insert
```

Command Line Mode using Executable Scripts

The executable scripts provided can be run as-is on the command line:

For compilation of the driver modules:

```
$ ./s6_trd_driver_build
```

For insertion of the driver modules and to invoke the application GUI:

```
$ ./s6_trd_driver_insert
```

Mouse Click Driven Mode

To compile the driver, open the `s6_pcie_dma_ddr3_gbe_axi` folder.

Double-click on **s6_trd_driver_build**. This cleans the area and builds the kernel objects. A window prompt appears (shown in Figure 2-7), click on **Run in Terminal** and proceed.

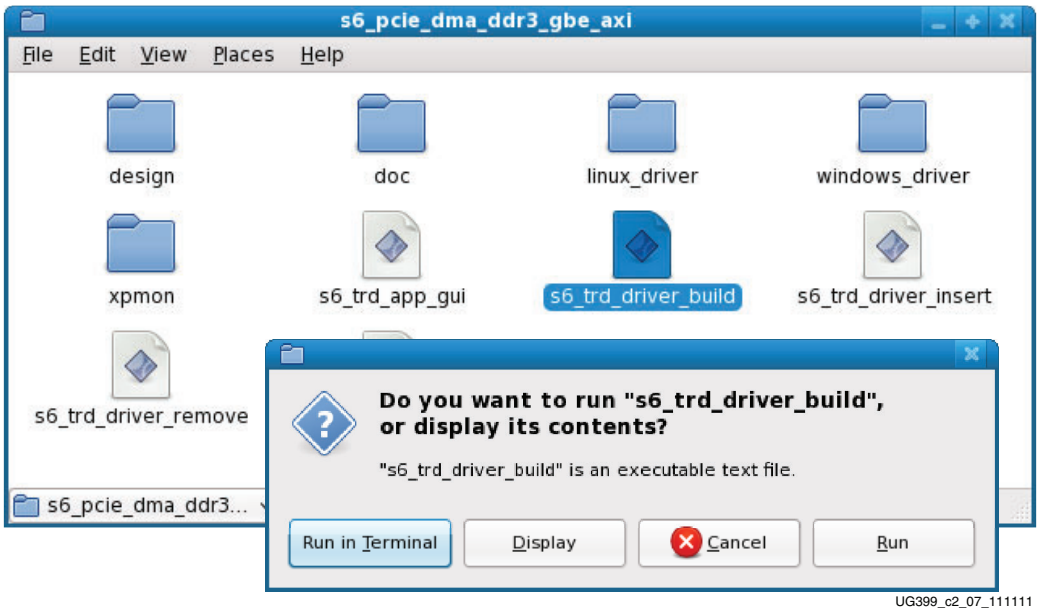


Figure 2-7: **Executing the Build Script**

Double-click on **s6_trd_driver_insert** (Figure 2-8). A Window prompt appears (shown in Figure 2-8), click on **Run in Terminal** and proceed. This inserts the driver modules into the kernel.

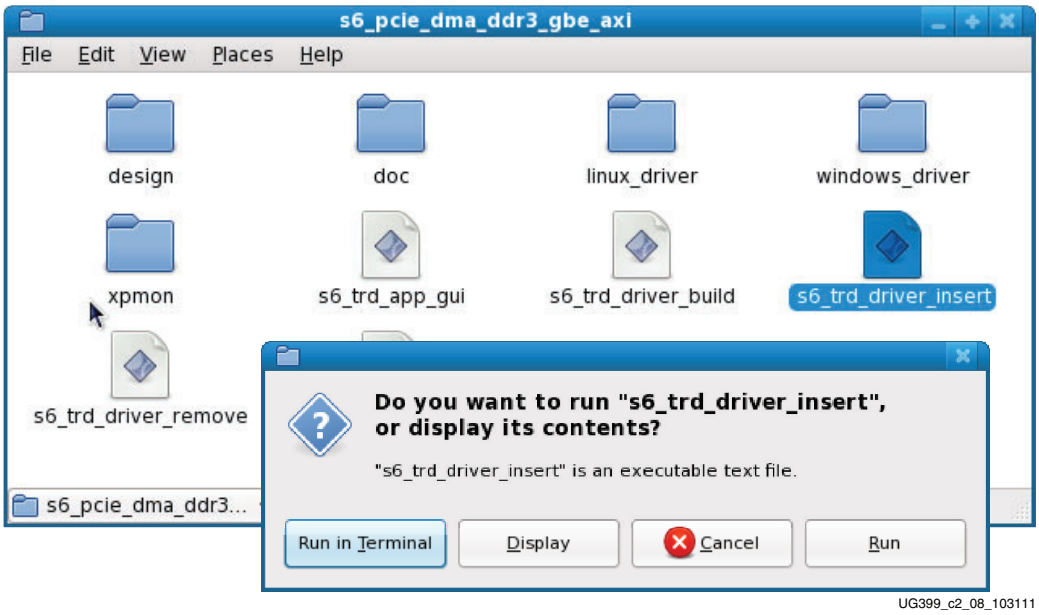


Figure 2-8: **Executing the Insert Script**

Proceed to Network Bring Up for network bring-up and GUI application usage.

3. To check if the driver modules are loaded into the kernel, try to use the `lsmod` utility in Linux. To see a list of kernel objects loaded:

   ```
   $lsmod | more
   ```

   `xblockdata_s6`, `xgbeth`, and `xdma_s6` modules are the Xilinx driver modules loaded as part of this TRD.

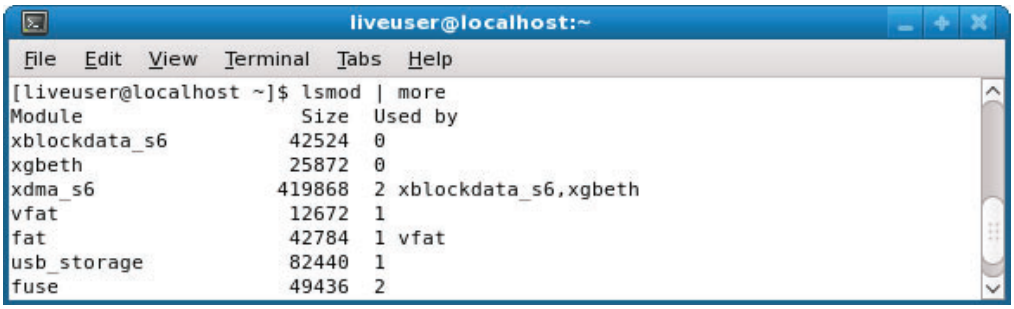


Figure 2-9: **lsmod Output**

Network Bring Up

After successful completion of the steps listed in Hardware Bring Up, page 16 and Software Bring Up, page 20, proceed to enable the Ethernet interface.

This section details Ethernet bring-up and uses the Network Configuration GUI on Linux.

1. To add a new network device, open the Network Configuration GUI (Figure 2-10)

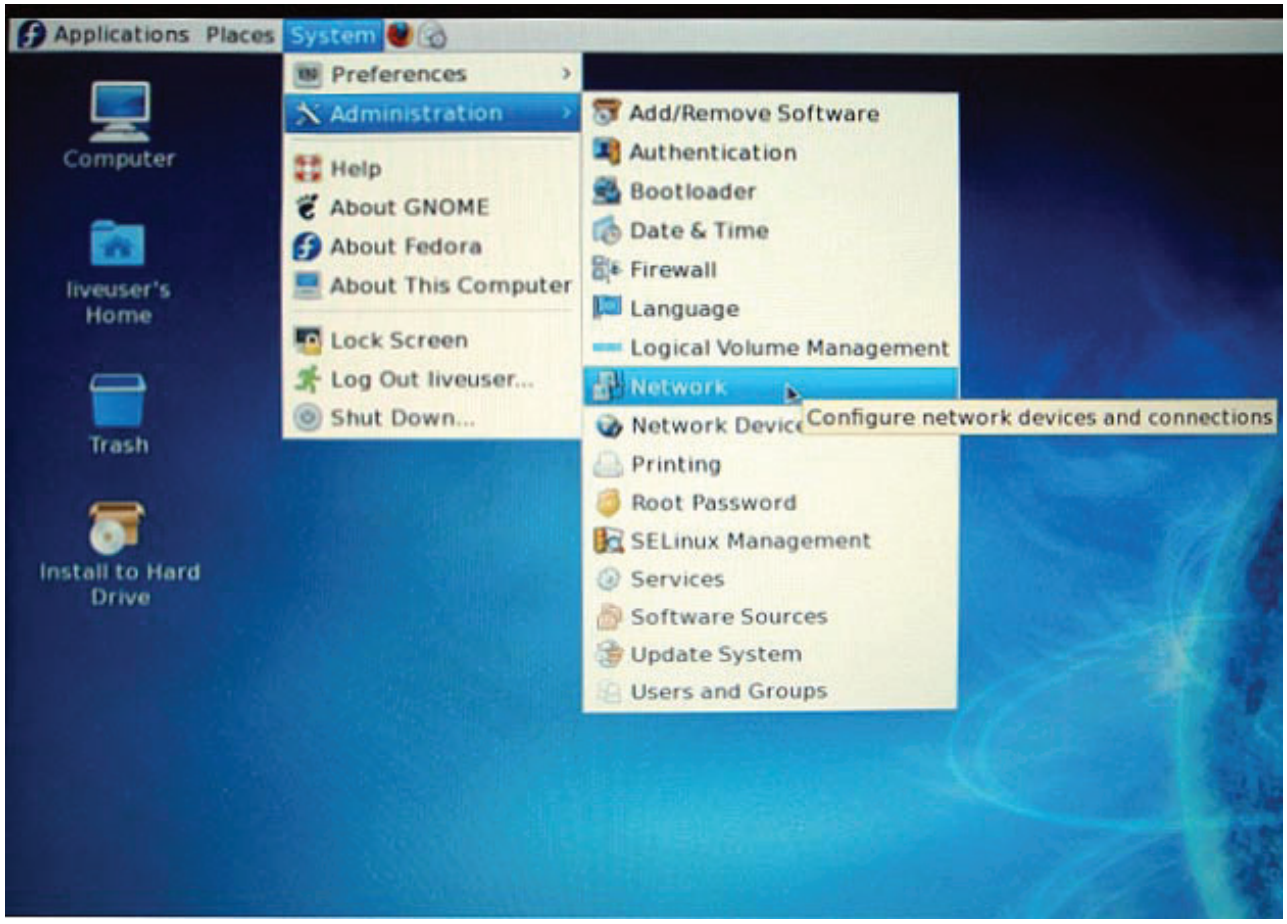


Figure 2-10: **Network Configuration GUI Path**

Note: If the LiveCD is not being used, invoking the Network Configuration GUI requires super-user password.

The GUI in Figure 2-11 appears.

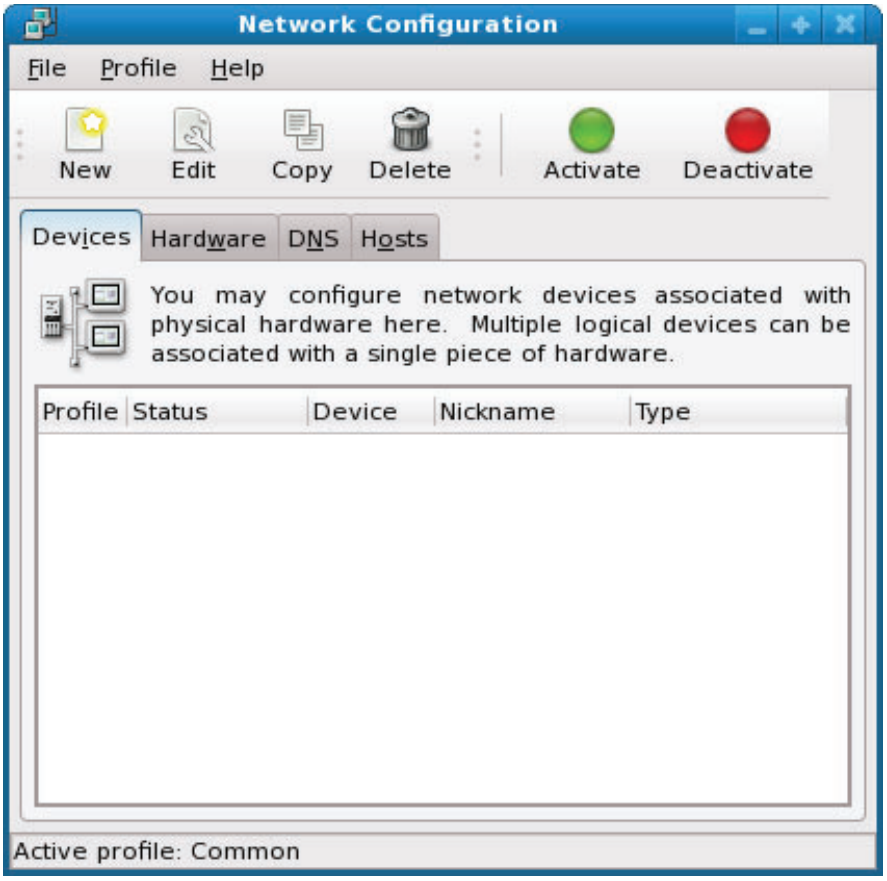


Figure 2-11: **Network Configuration GUI—Devices Tab**

With the LiveCD, the devices tab does not show anything as no interface is active. Make sure that the Ethernet cable is not connected to the existing LAN port on the PC.

Note: Activating multiple Ethernet ports on the same system requires each of the ports to be on different subnets. For this setup, do not connect any other network cable to the existing Ethernet ports on the PC. Connect the network cable only to the SP605 board RJ45 slot provided.

The **Hardware** tab shows the hardware devices present. As an example for this section, assume the NIC is detected as `eth1` as shown in Figure 2-12.

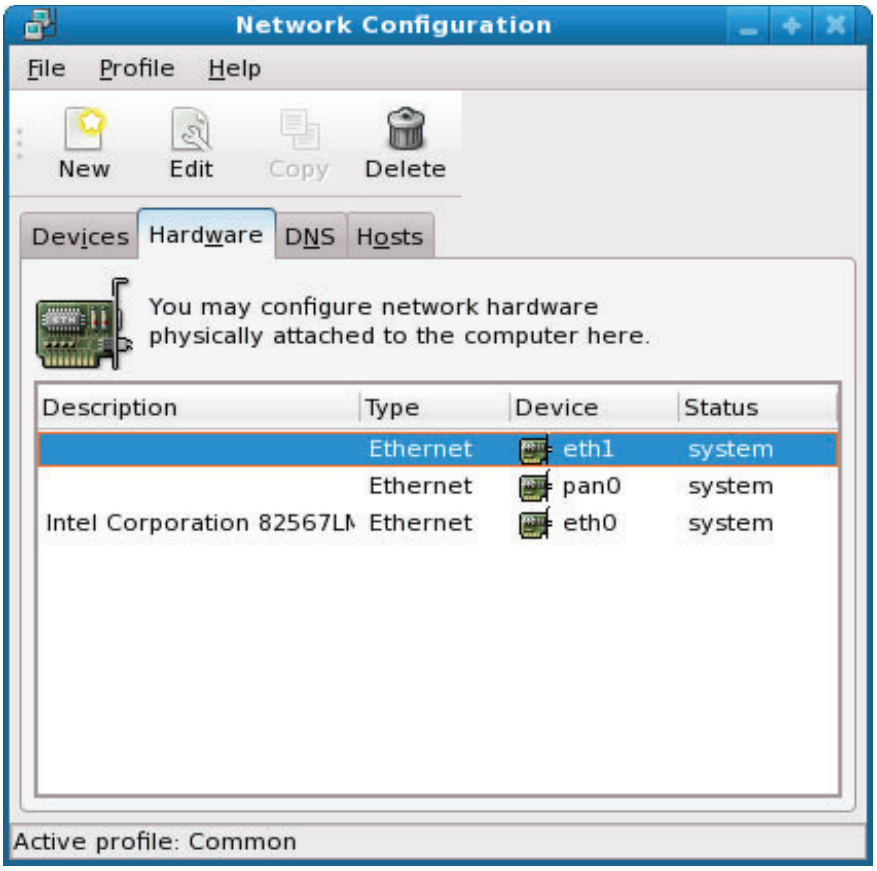


Figure 2-12: **Network Configuration GUI—Hardware Tab**

To have the SP605 NIC show up in the **Devices** tab, a new Ethernet connection needs to be created.

Click on **New→ Create New Ethernet Connection**.

Complete the setup process by following the instructions carefully on the screen. If an existing Ethernet connection(s) is available, this NIC is detected as an additional ethX interface where X would depend on the number of existing Ethernet interfaces.

The IP address assignment can be static or dynamic depending on the network setup. Contact the network administrator to understand IP address assignments on the network and also to obtain the necessary settings for network configuration.

Enter appropriate DNS settings for DHCP configuration as per the network administrator's instructions.

Once done, what is shown in Figure 2-13 should appear on the screen.

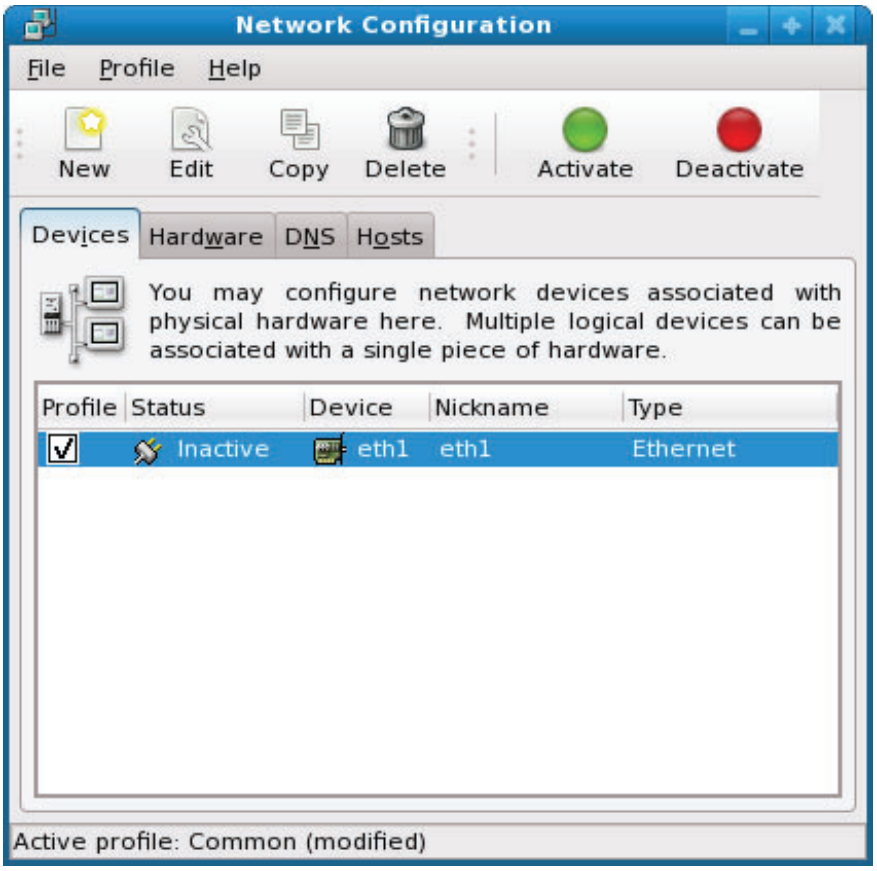


Figure 2-13: **Network Configuration after Adding Interface**

Save changes by clicking on the **File→ Save Changes** option. Do not close the network configuration GUI as it will be required later to activate the interface.

2. MAC Address Assignment

   Use the `ifconfig` utility to check the device:

   ```
   $ /sbin/ifconfig eth1
   ```

   Initially the NIC is assigned an Ethernet interface `eth1` with a default MAC address of `AA:BB:CC:DD:EE:FF`.

   Make note of the MAC ID assigned to the SP605 board with your kit (Figure 2-14).

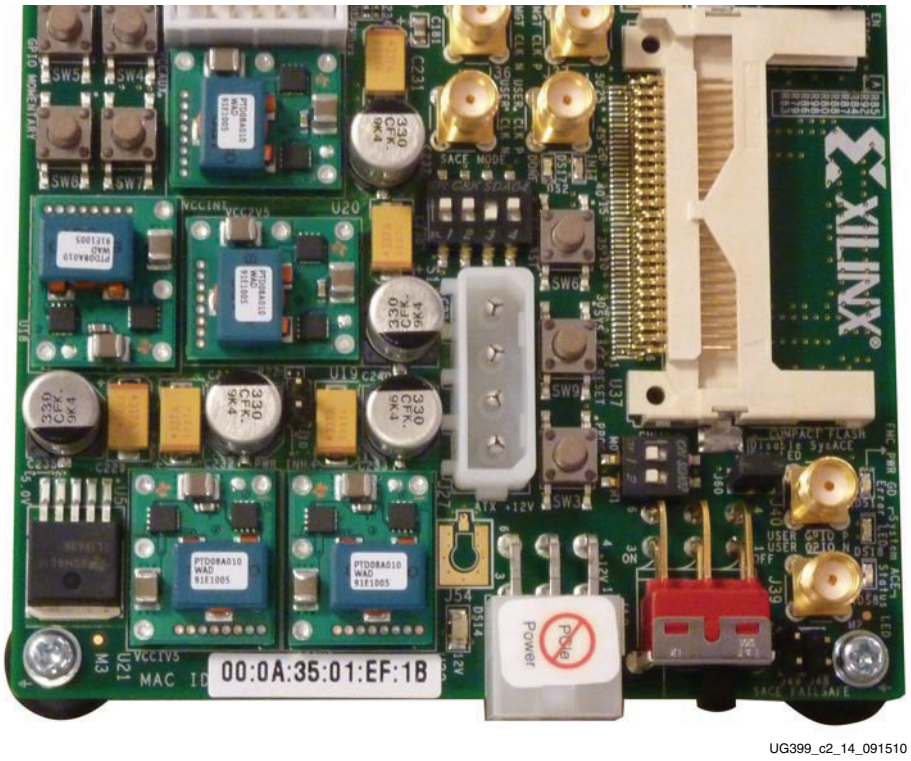


Figure 2-14: **MAC Address Label on SP605 Board**

Open a terminal and navigate to the `s6_pcie_dma_ddr3_gbe_axi` folder. Set the MAC address for the Ethernet MAC in the design using the script provided:

```
$ ./setmac_id ethX <SP605_MAC_ID>
```

For the MAC ID shown, the command would be:

```
$ ./setmac_id eth1 00:0A:35:01:EF:1B
```

After the MAC ID is assigned, try `ifconfig` again and the assigned MAC ID assigned to the NIC should appear.

Refer to the screen capture in Figure 2-15.

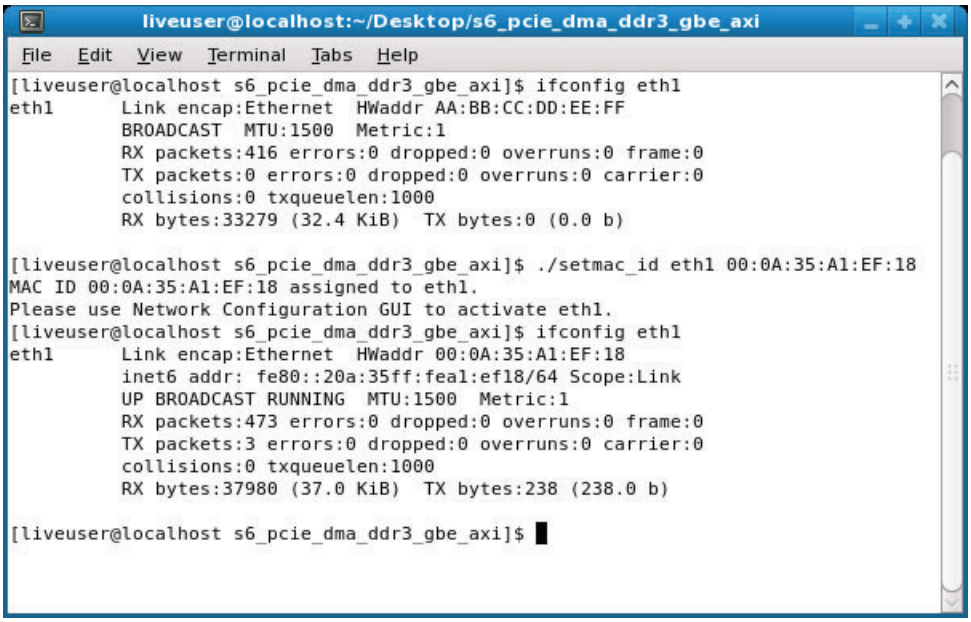


Figure 2-15: **MAC ID Assignment**

3. Invoke the Network Configuration GUI and activate the interface by clicking on the **Activate** button.
4. Invoke the browser and set the browser proxy settings as required for your network (contact your network administrator). Now online service should be available.

Using Application GUI

This section explains the application GUI provided and various features available.

1. This step involves GUI invocation. Separate steps are defined for a command line user conversant with Linux or for a user preferring button-click operations.

Command Line Mode Using Makefile

To compile and invoke the GUI navigate to the `s6_pcie_dma_ddr3_gbe_axi/xpmon` folder, follow these steps:

To clean the area.

```
$ make clean
```

To compile the files.

```
$ make
```

To invoke the GUI.

```
$./xpmon
```

Command Line Mode Using Executable Scripts

The executable scripts provided in `s6_pcie_dma_ddr3_gbe_axi` folder can be used as is on the command line:

```
$ ./s6_trd_app_gui
```

Mouse Click Driven Mode

Double click on **s6_trd_app_gui**. A window prompt appears (Figure 2-16), click on **Run in Terminal** and proceed.

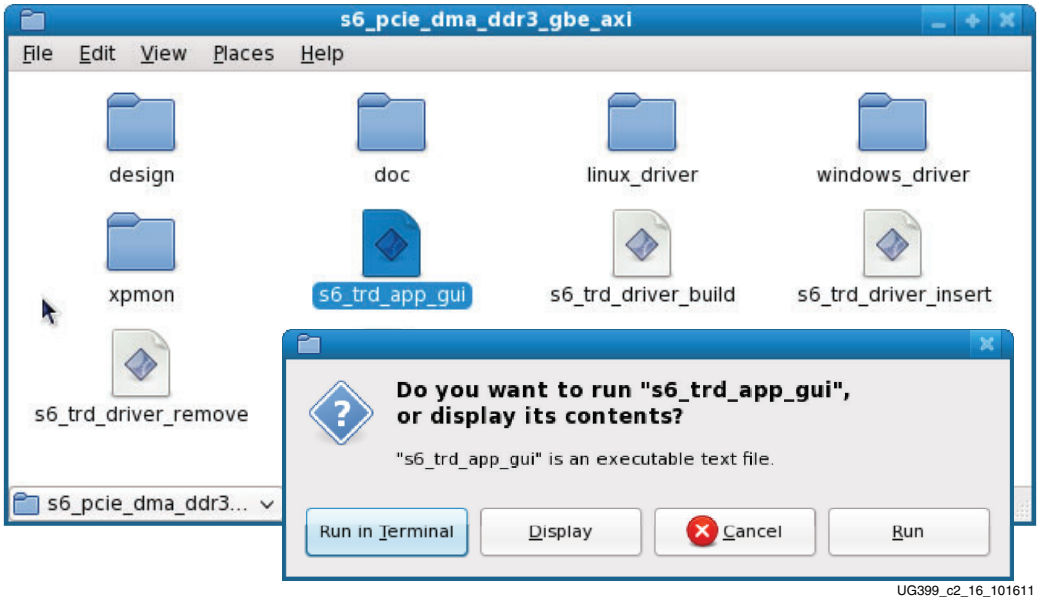


Figure 2-16: **Invoking GUI through Script**

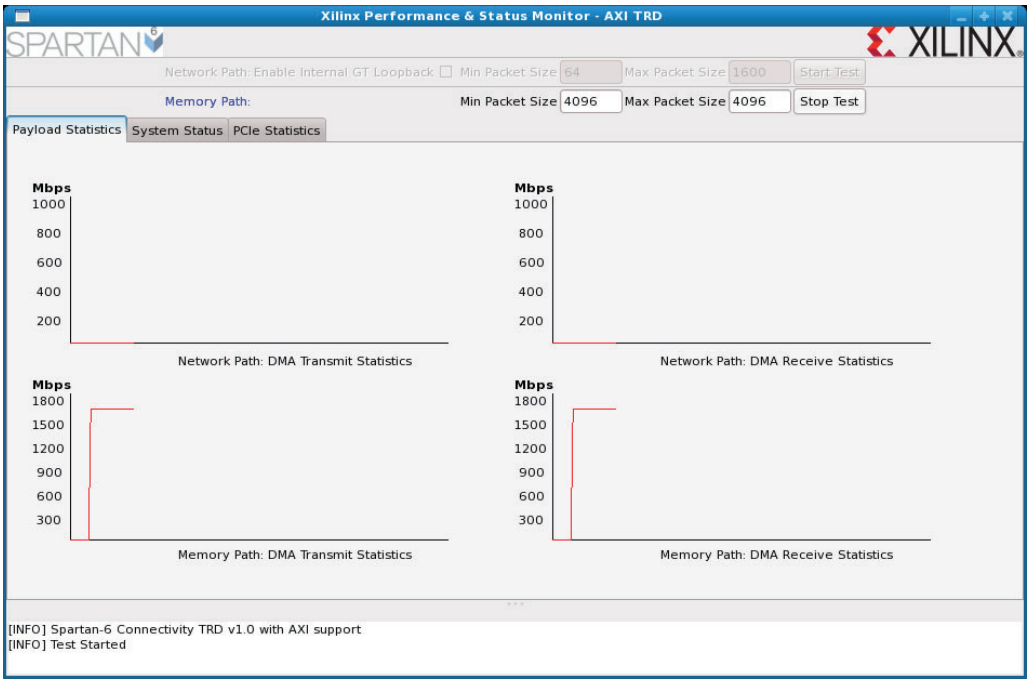
2. GUI walk-through screen-by-screen:

Test Setup and Payload Statistics

This screen defines the various test options provided for the memory path. The packet size for Ethernet can not be controlled by the application GUI as it is managed by the network (TCP/IP) stack.

There is an option to set the minimum and maximum packet size in bytes. While executing the test, the software driver builds packets of random length within the specified range. The range supported is 256-4096 bytes.

The screen in Figure 2-17 plots the data throughput obtained from the various DMA engines.

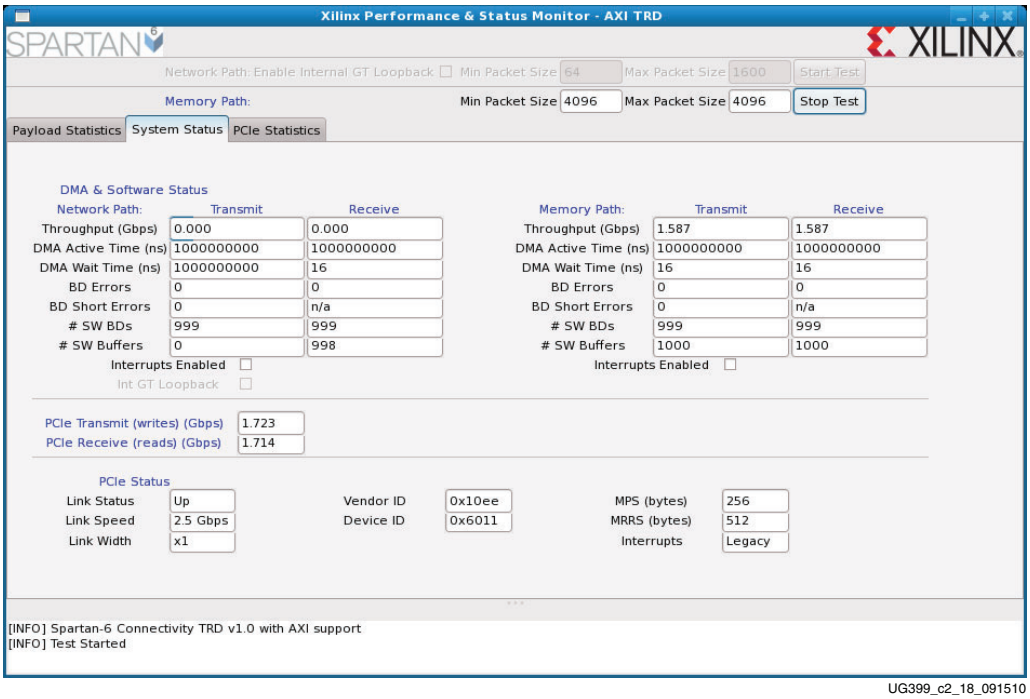


UG399_c2_17_091510

Figure 2-17: **Test Setup and Payload Statistics Screen**

System Status Screen

This screen gives the status of the PCI Express and DMA engine.



UG399_c2_18_091510

Figure 2-18: **System Status Screen**

Transaction Statistics

This screen plots the transaction utilization statistics on the PCI Express Transaction interface.

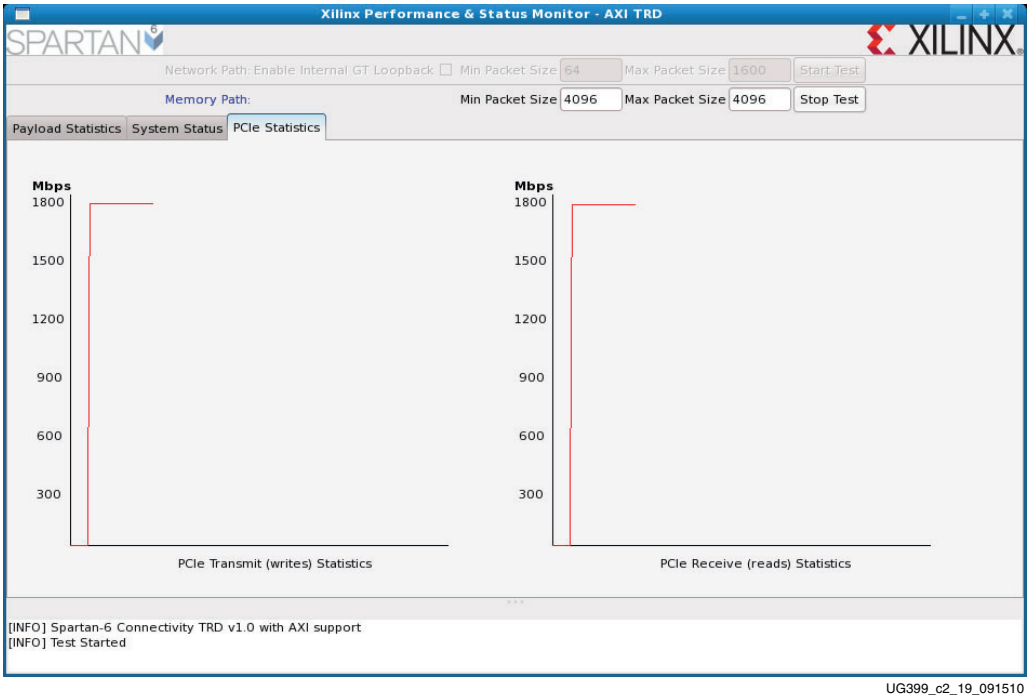


Figure 2-19: **Transaction Utilization Statistics**

Testing with the Windows Operating System

Copy the contents of the USB flash drive

The reference design files are provided on the USB flash drive delivered with the connectivity kit. Insert the flash drive into a USB connector on the PC system and copy the `s6_pcie_dma_ddr3_gbe_axi` directory to the PC system.

Note: Ensure that the path where the `s6_pcie_dma_ddr3_gbe_axi` directory is located does not have spaces.

Install the Drivers and GUI

1. Navigate to the `s6_pcie_dma_ddr3_gbe_axi` directory. Double click `x_s6_trd_setup.exe` (Figure 2-20).

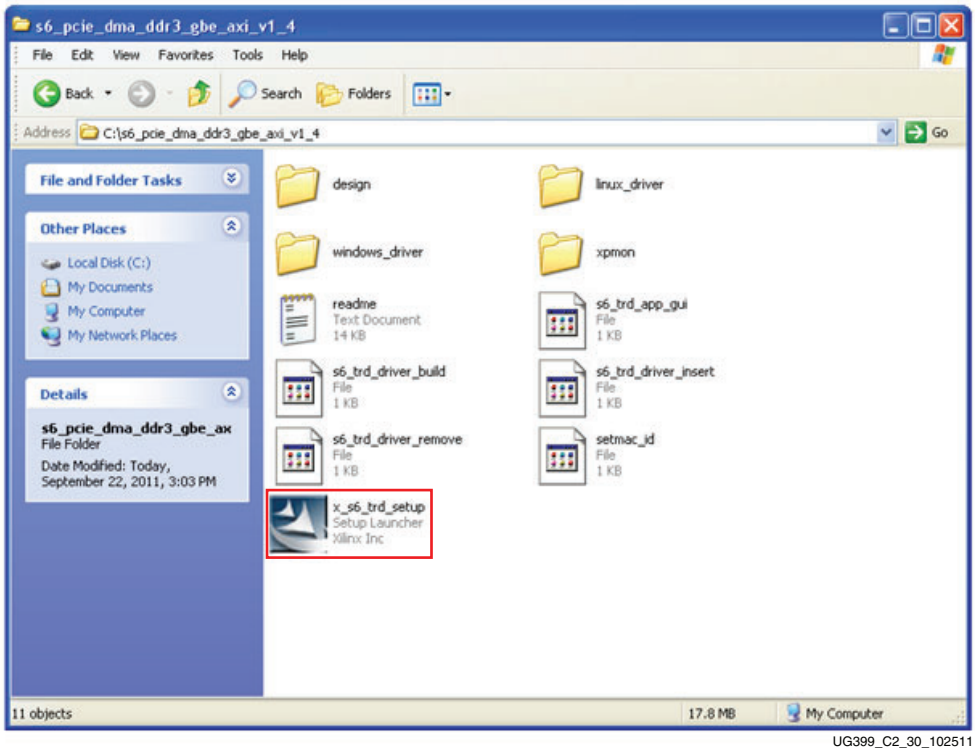


Figure 2-20: **Location of x_s6_trd_setup.exe**

2. When the InstallShield Wizard dialog box opens, click **Next** (Figure 2-21).

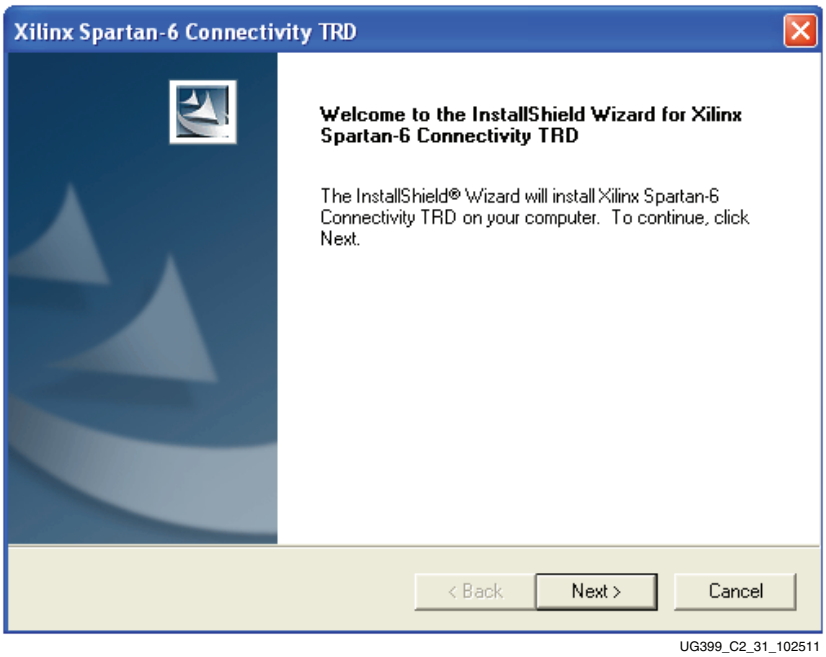


Figure 2-21: **InstallShield Wizard**

3. When the Setup Type dialog box opens, select **Typical** (Figure 2-22). This option installs the driver files and directories to the `C:\Program Files` directory. Click **Next**.

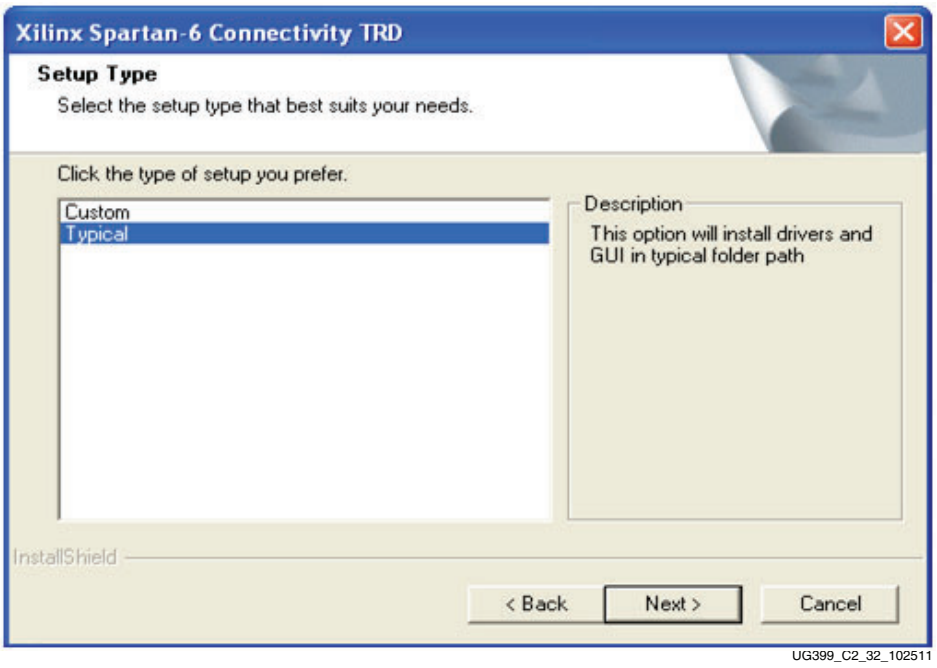


Figure 2-22: **Typical Setup Selected**

4. When the XNIC driver selection dialog box opens (Figure 2-23), select the driver type that matches the MAC to PHY interface used by the SP605 board. For example, if the bitfile programs the SP605 board to use GMII, select **GMII**. Click **Next**.

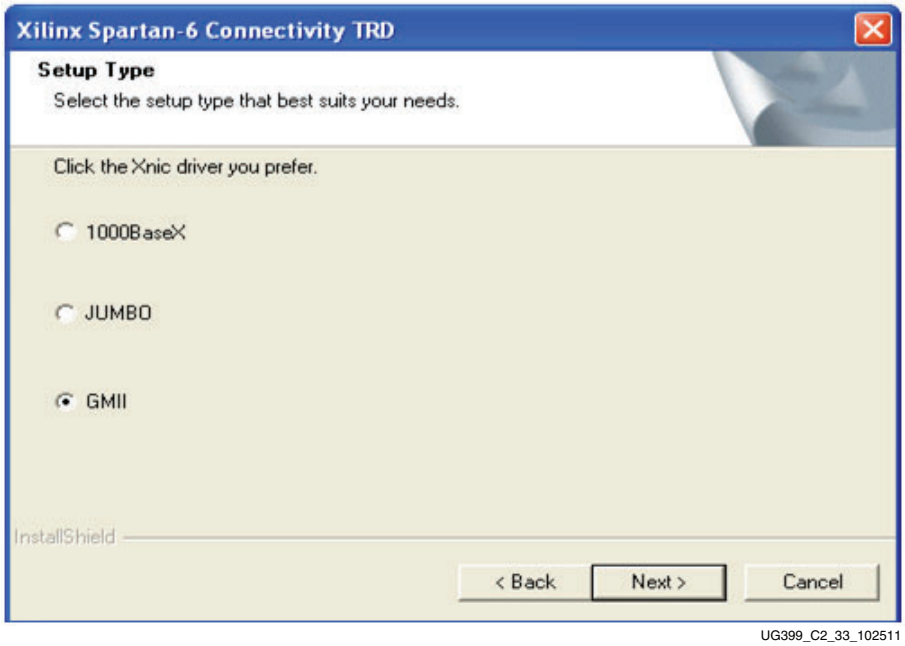


Figure 2-23: **XNIC Driver Selection**

5. The current settings show the destination directory where the program files will be located (Figure 2-24). Review and click **Next**.

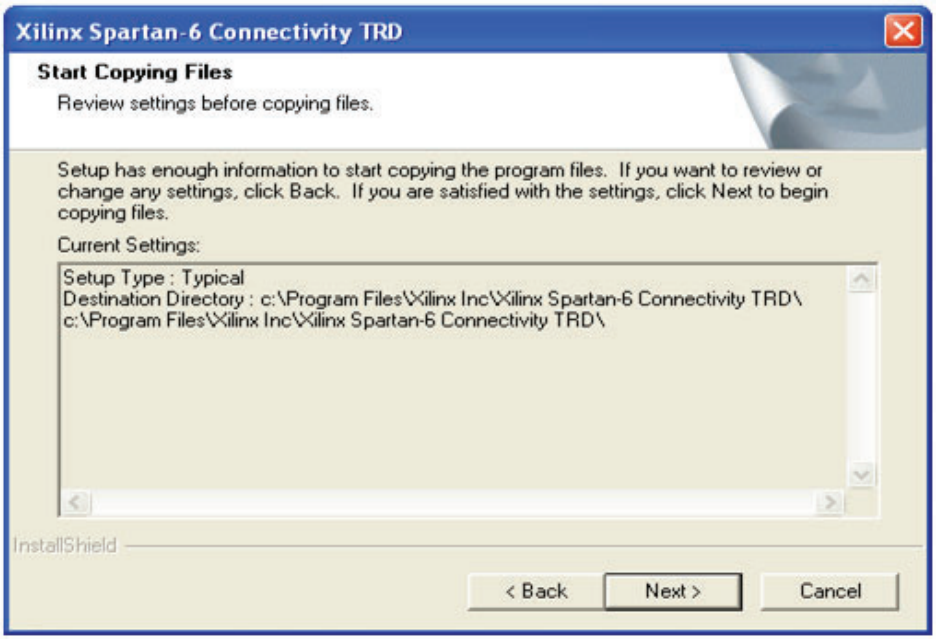


Figure 2-24: **Review Settings Window**

6. When the InstallShield Wizard Complete dialog box opens (Figure 2-25), click **Finish**.

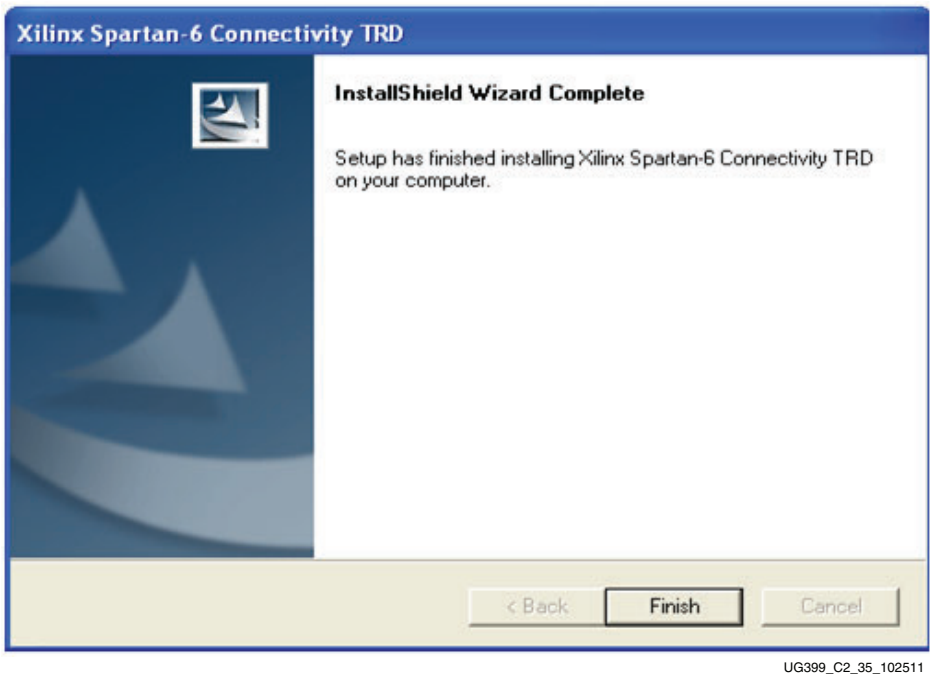


Figure 2-25: **Installation Complete**

Add Hardware Wizard

After the drivers are detected by Windows, the Add Hardware Wizard opens (Figure 2-26). Click **Next**.

Figure 2-26: **Add new Hardware Wizard**

Add Xilinx DMA Driver Device

1. When the Found New Hardware Wizard opens (Figure 2-27). Select **Install the software automatically** and click **Next**.

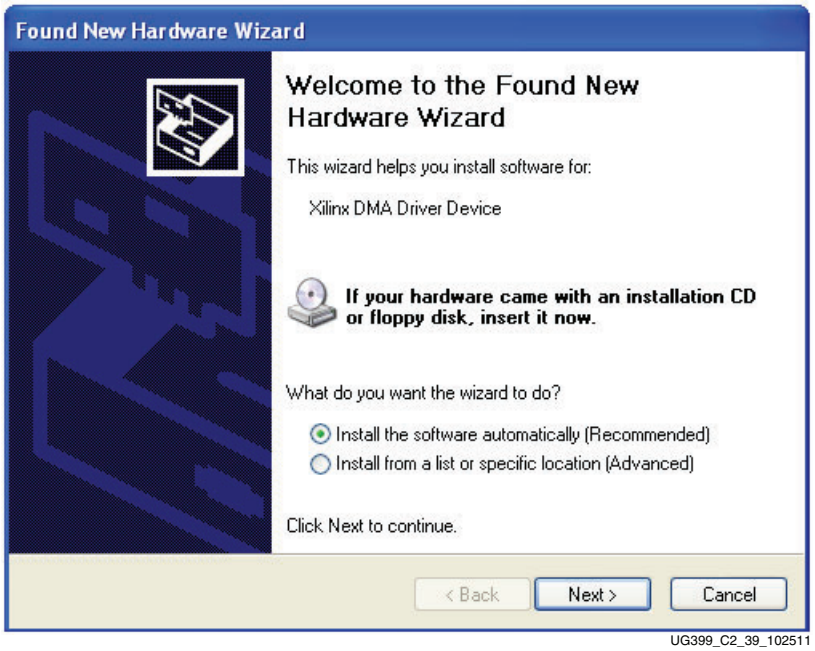


Figure 2-27: **Found XDMA Hardware**

2. When the XDMA driver is installed Figure 2-28 opens. Click **Finish**.

Figure 2-28: **XDMA Driver Installation Complete**

Add Xilinx Ethernet Adapter

1. Figure 2-29 indicates the Add Hardware Wizard detects the XNIC driver must be installed. Select **Install the software automatically** and click **Next**.

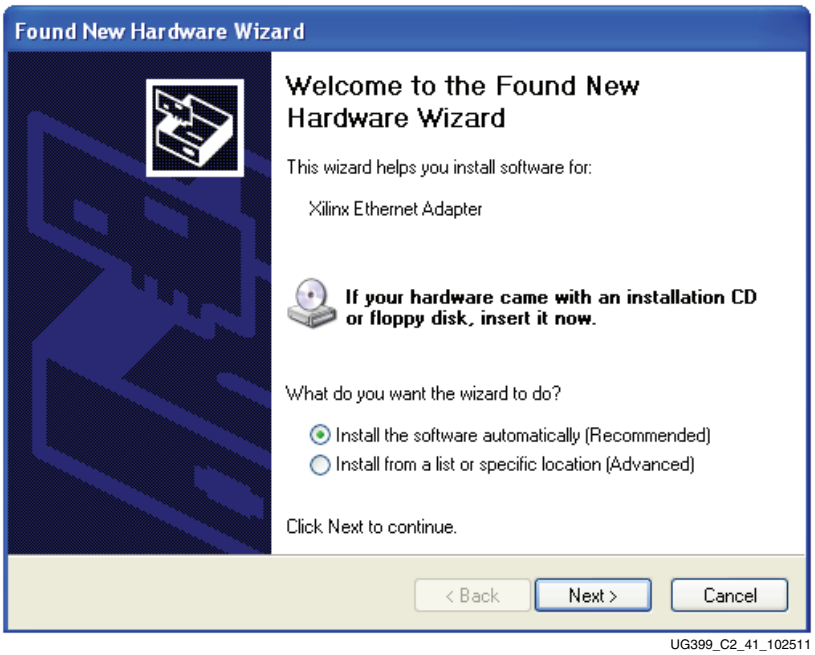


Figure 2-29: **Found XNIC Hardware**

2. Select the appropriate Xilinx Ethernet Adapter and click **Next** (Figure 2-30).

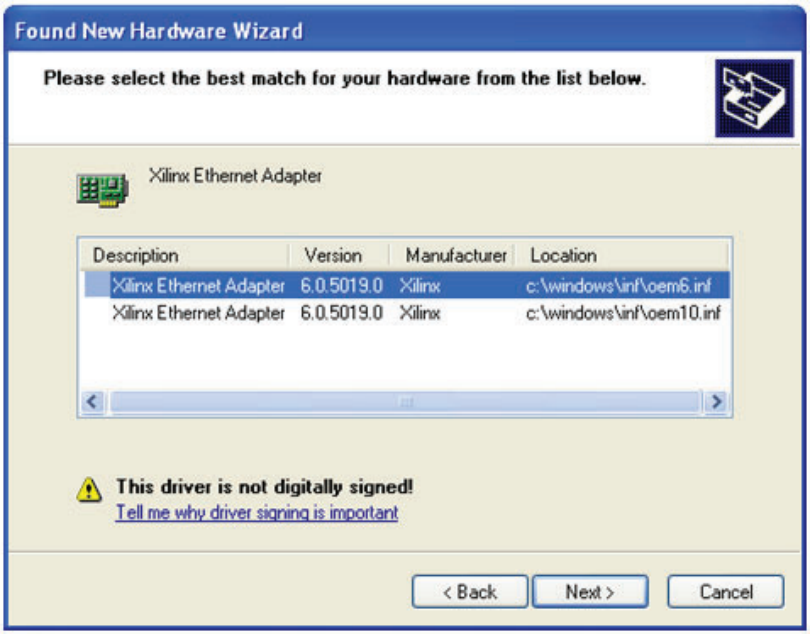


Figure 2-30: **Ethernet Adapter Selection**

3. If the dialog box shown in Figure 2-31 opens, click **Continue Anyway**.

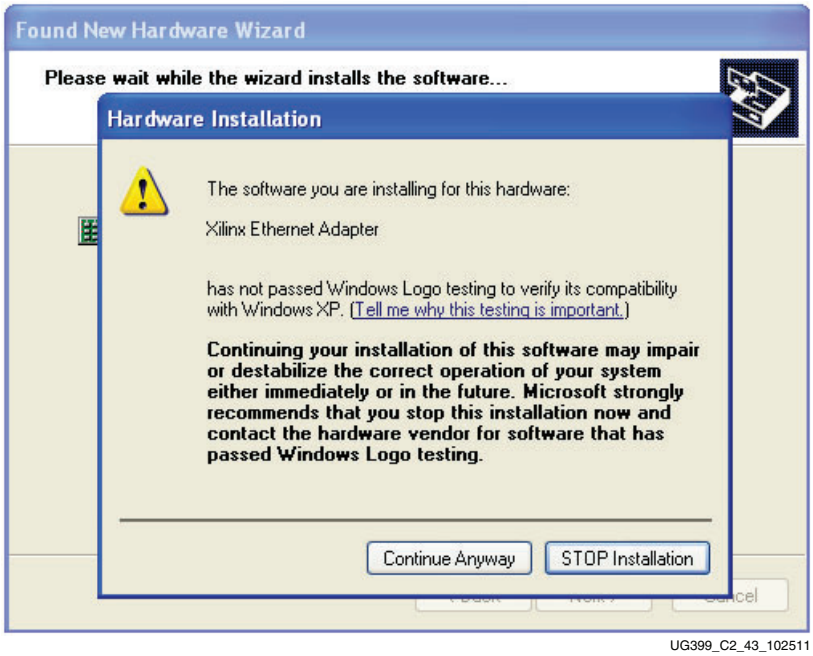


Figure 2-31: **Windows Compatibility Test Dialog Box**

4. When the XNIC driver is installed Figure 2-32 opens. Click **Finish**.

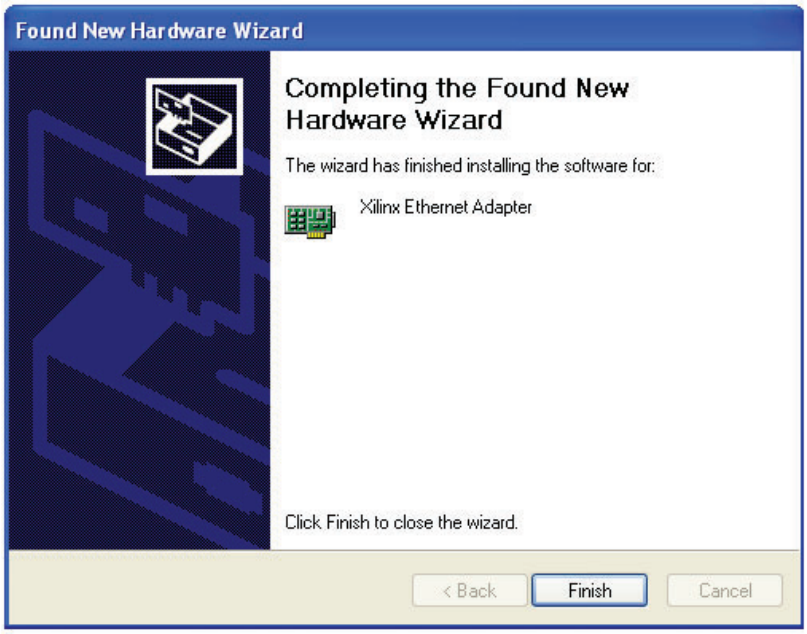


Figure 2-32: **XNIC Driver Installation Complete**

Add Xilinx Block Driver Device

1. Figure 2-33 indicates the Add Hardware Wizard detects the XBLOCK driver must be installed. Select **Install the software automatically** and click **Next**.

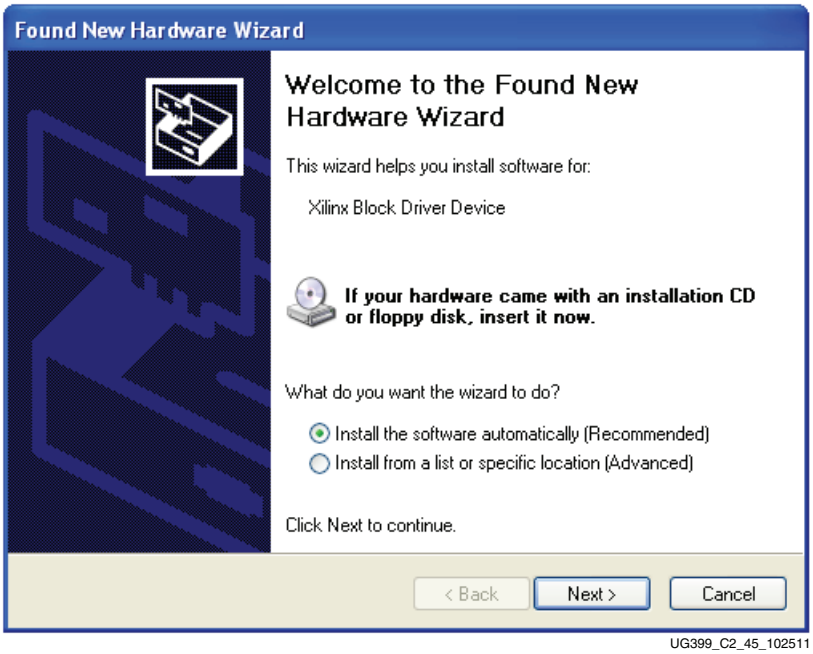


Figure 2-33: **Found XBLOCK Hardware**

2. When the XBLOCK driver is installed Figure 2-34 opens. Click **Finish**.

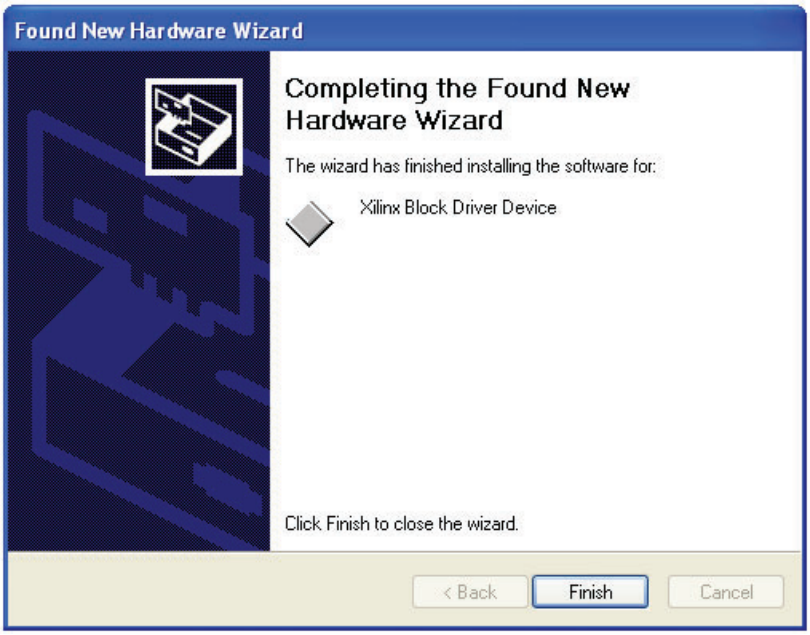


Figure 2-34: **XBLOCK Driver Installation Complete**

3. Click **Finish** (Figure 2-34).

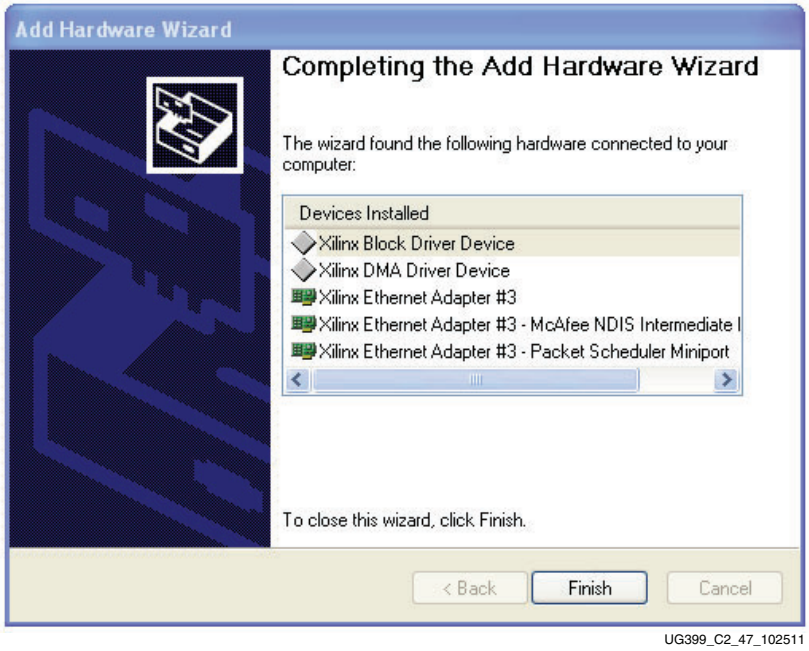


Figure 2-35: **List of Installed Drivers**

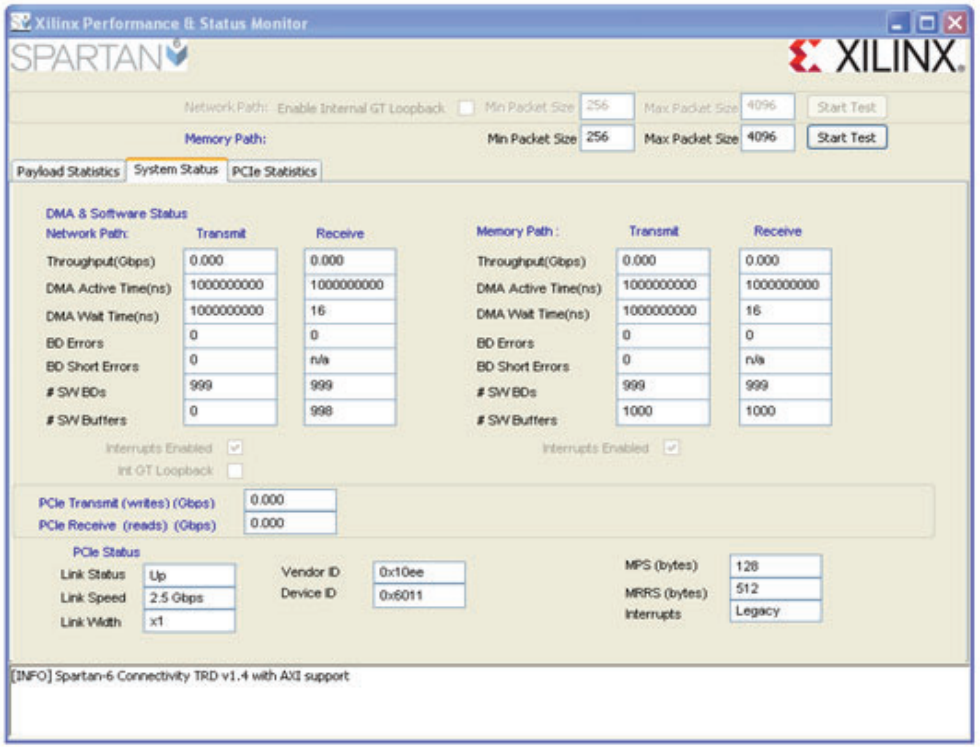
Note: Running `x_s6_trd_setup.exe` after the drivers are installed will uninstall the drivers and clean the install folders. Do not run `x_s6_trd_setup.exe` except when installing new versions of the drivers.

Launching Performance Monitor

To launch the Performance Monitor application (Figure 2-37) from the desktop, double-click the **xpmon icon** (Figure 2-36). Proceed to step 1 of Using Application GUI, page 29 to run the application GUI. The xpmon icon is also available in the `C:\ProgramFiles\XilinxInc\Xilinx Spartan-6 Connectivity Kit-SP605` directory.

UG399_C2_48_102511

Figure 2-36: **Xilinx Performance Monitor GUI**



UG399_c2_36_112811

Figure 2-37: **Xilinx Performance Monitor GUI**

Viewing Drivers

After the drivers are installed, they are accessible as shown in Figure 2-38 by right-clicking on the **My Computer** icon and selecting **Properties.** Select the **Hardware** tab and click **Device Manager**.

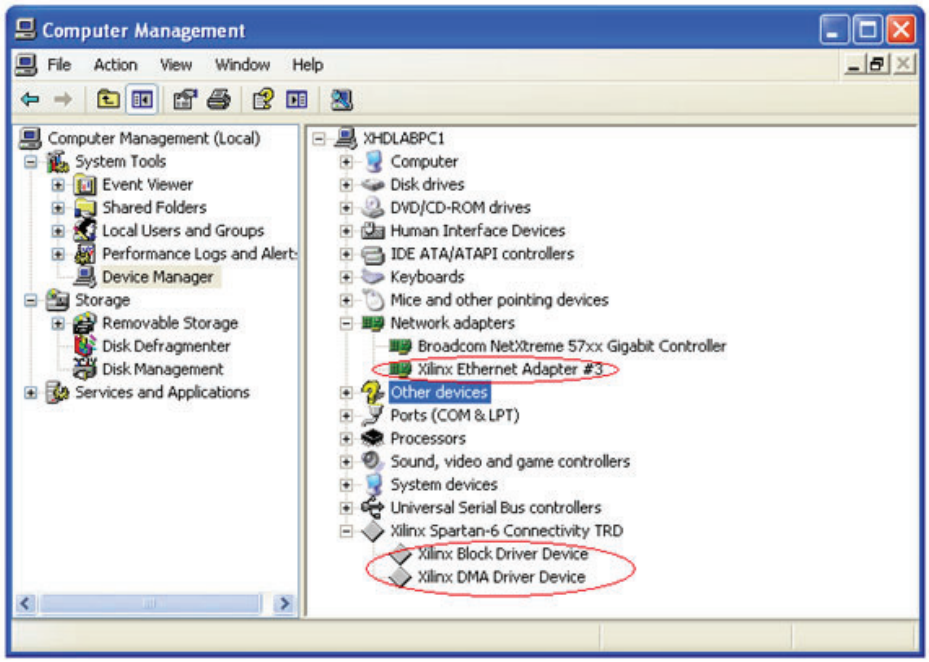


Figure 2-38: **Xilinx Ethernet Driver, DMA Driver and Block Data Driver in Device Manager**

Configuration

After the drivers are installed, Configure the Xilinx Ethernet Adapter as follows:

1. Browse to Network Connections and click **Set up a home or small office network**.
2. Right click on the **Local Area Connection *n***, where *n* depends on the number of LAN ports supported in the PC that has Xilinx Ethernet Adapter in the Device Name field.
3. Click Properties as shown in Figure 2-39.

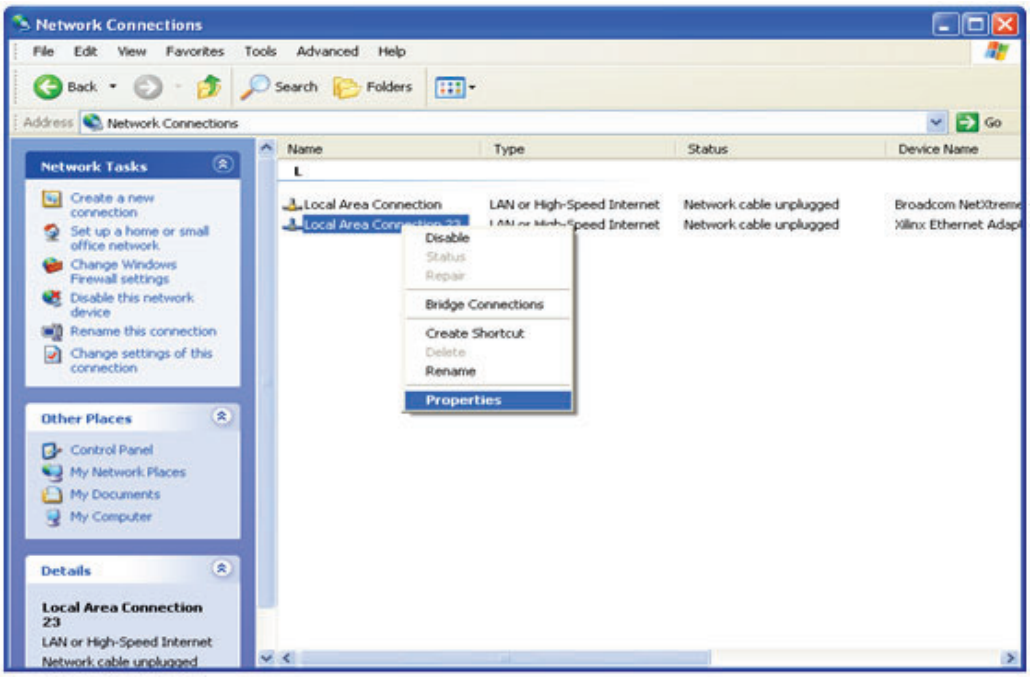


Figure 2-39: **Local Area Connection Properties**

4. When the Local Area Connection *n* properties window opens (Figure 2-40), select the **General** tab and click **Configure**.

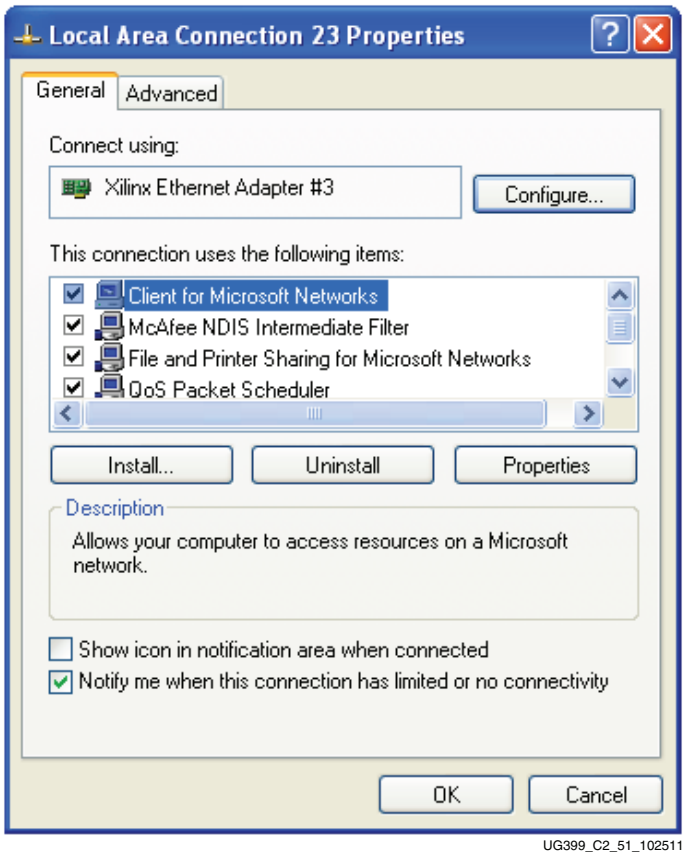


Figure 2-40: **Local Area Connection Properties**

Click the **Advanced** tab. Select **Locally Administered Address** as shown in Figure 2-41. Enter a value that matches with one written on the edge of your SP605 board (as shown in Figure 2-14, page 28). Click **OK**.

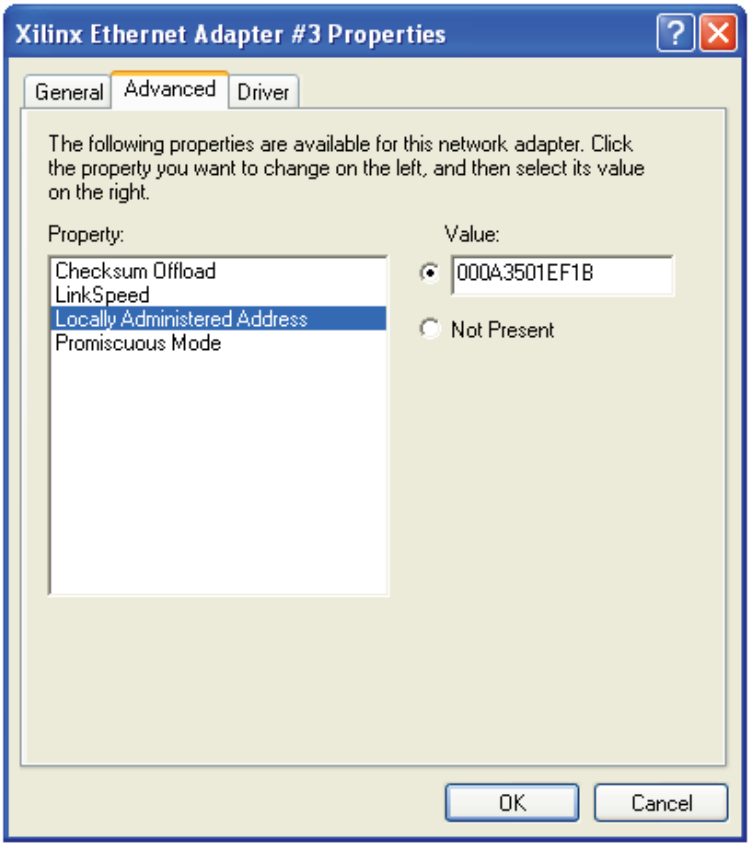


Figure 2-41: **Advanced Configuration Option**

5. On the Local Area Connection n Properties window, select **Internet Protocol (TCP/IP)** selection as shown in Figure 2-42 to configure the IP address of you Xilinx Ethernet Adapter.

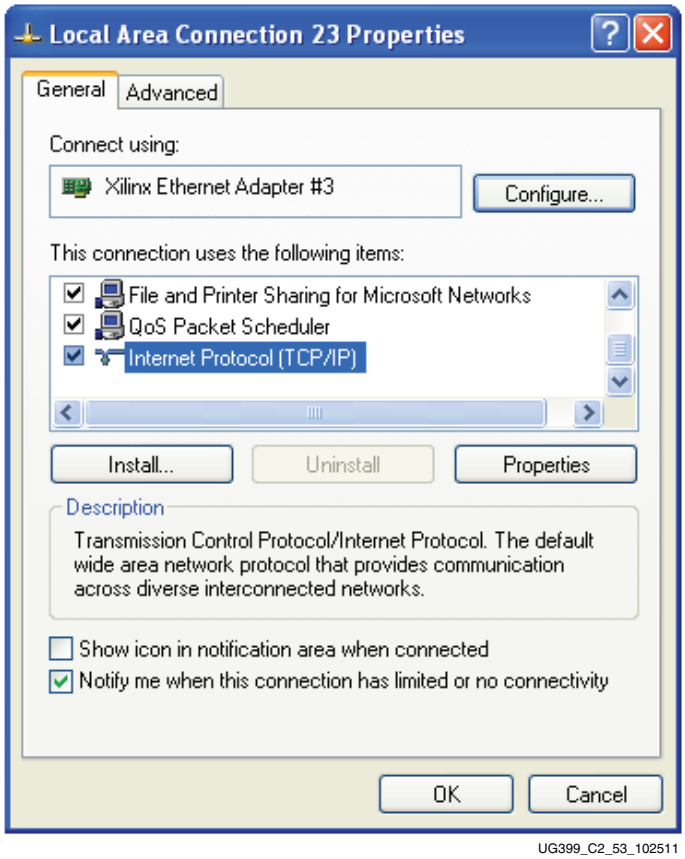


Figure 2-42: **Internet Protocol Properties**

Note: IP address allocation varies based on the network connection. Contact the network administrator to obtain network connection details on type of network connection (for example, whether IP address allocation is static or dynamic) and proxy settings requirements, if any.

Using Various Features

This section explains the various features of Ethernet configurable via standard tools or software driver macros. These features, when tested with Fedora 10 LiveCD, will require the Linux commands given to be preceded by `sudo`.

Ethernet Specific Features

The Ethernet specific features can be exercised by using command line utilities like `ifconfig` and `ethtool` present in Linux or through the network adapter properties GUI on Windows.

The Ethernet driver provides functions which are used by `ifconfig` and `ethtool` on Linux and Network properties GUI on Windows to report information about the NIC. For

reporting packet drops due to FCS errors, registers provided by the Ethernet Statistics IP are used.

On Linux, 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.

On Linux, 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 (On Linux)

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 (MARVELL PHY registers in case of GMII mode and PCS-PMA registers in case of 1000BASE-X mode) can be read using the following command:

```
$ ethtool –d ethX
```

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

```
$ ifconfig ethX
```

Autonegotiation

The tri-mode Ethernet MAC (TEMAC) is capable of operating at 10 Mb/s, 100 Mb/s or 1,000 Mb/s speeds. Autonegotiation is the process by which two devices capable of supporting different speeds choose a common speed and establish a link.

On Linux, the following command results in the information about the Ethernet interface selected, which also lists the speed of operation currently selected.

```
$ ethtool -S ethX
```

To change the speed, use the following command and select a speed. If the device is currently operating at 1000 Mb/s, select the 100 Mb/s or 10 Mb/s options to set a different speed. The speed change should succeed if the link partner supports the newly advertised speed.

```
$ ethtool -s ethX speed [10|100|1000
```

On Windows, autonegotiation can be configured by selecting **Link Speed** option. Select an appropriate value for the application. If the device is currently operating at 1,000 Mb/s, select the 100 Mb/s or 10 Mb/s options to set a different speed. The speed change should succeed if the link partner supports the newly advertised speed.

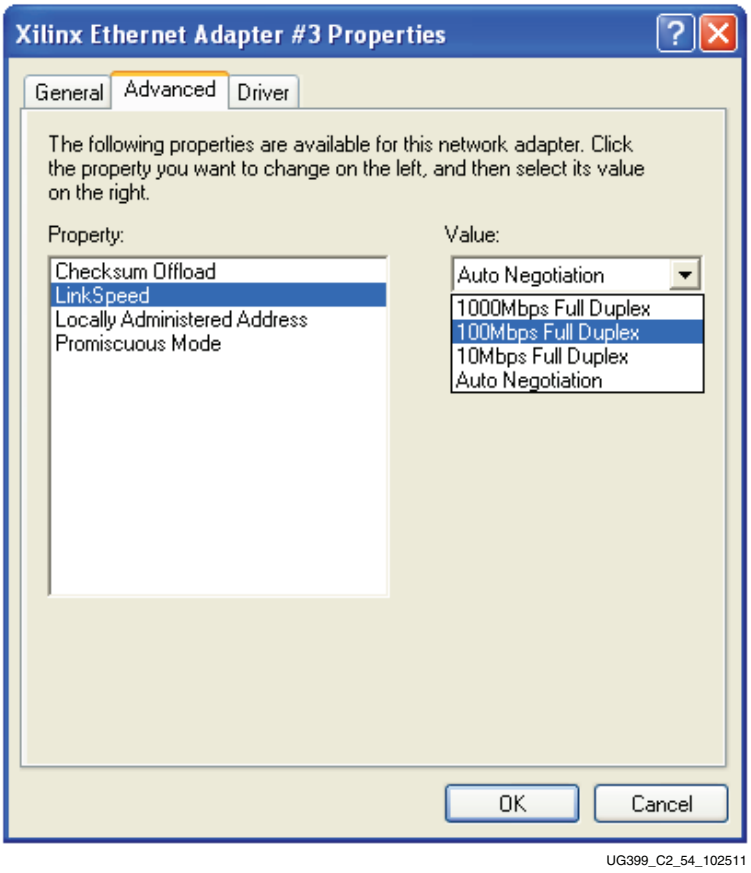


Figure 2-43: **Link Speed Property**

Promiscuous Mode

Enabling promiscuous mode disables address filtering in the TEMAC. This causes the driver to receive all the traffic which increases the CPU load.

On Linux, the following command can be used to enable promiscuous mode:

```
$ ifconfig ethX promisc
```

The following command disables the promiscuous mode:

```
$ ifconfig ethX -promisc
```

On Windows, in the Advanced properties for the card, select the Promiscuous Mode property. Value could be Enable or Disable. Select as appropriate.

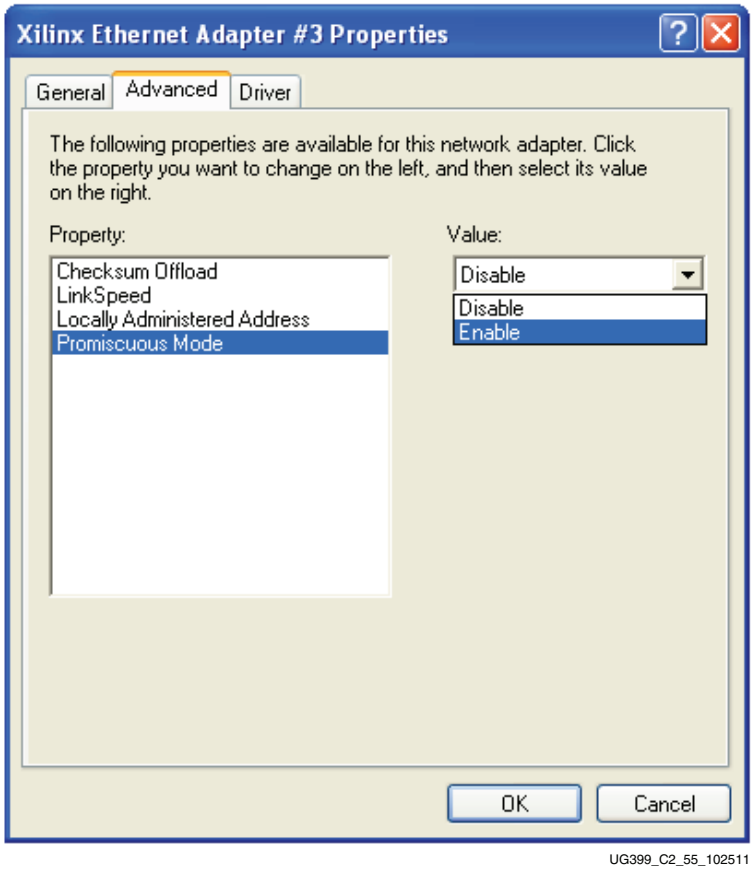


Figure 2-44: **Promiscuous Mode Property**

Using a packet sniffer like Wireshark in promiscuous mode puts the driver and in turn the device in promiscuous mode.

Checksum Offload

Enabling or disabling checksum offload can be controlled through ethtool for the TCP/IP stack.

On Linux, the following command gives the status of checksum offload:

```
$ ethtool -k ethX
```

In order to turn on or off checksum offload for either transmit or receive directions, use the following command:

```
$ ethtool -K ethX tx [on|off] rx [on|off]
```

This statistics command can be used to find out the number of packets for which checksum was offloaded to hardware, and the maximum number of fragments a packet was split into because of checksum offload:

```
$ ethtool -S ethX
```

On Windows, the Checksum option can be enabled by clicking on the Advanced tab. Use the appropriate Checksum Offload option as required.

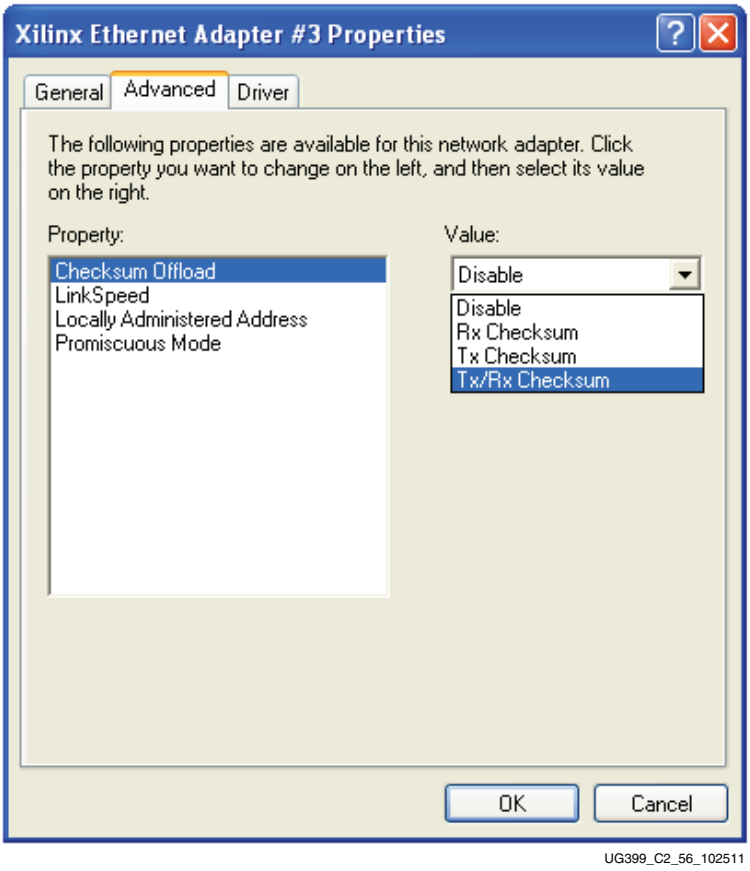


Figure 2-45: **Checksum Offload Property**

Memory Application Specific Features

This section describes the feature usage specific to the memory path connected to external DDR3 SDRAM through the Spartan-6 Memory Controller block. The packet size feature is available through the GUI.

Packet Size

As there are no sidebands to store the start-of-packet and end-of-packet information in DDR3, the design builds packets of a programmed size when sending data to system memory. This packet size can be programmed by writing to a specific register in the memory path user-space register. The application GUI programs this register with an average of the minimum and the maximum packet size value entered by the user.

AXI4 Interface Burst Size

The maximum burst size for transactions across the AXI4 interface can be set by means of the programming register for write and read burst size. By default, these are set to 256 which is the maximum AXI4 protocol supported burst size. This change must be done in the driver code (`xblockdata/user.c`) and cannot be changed through the GUI.

Shutting Down the System

On Windows, executing **x_s6_trd_setup** will uninstall the drivers and the application.

Linux Flow

The driver modules are automatically removed if the system is rebooted; however, the following steps are advised before shutting down the system for a graceful exit.

1. This step involves removal of the kernel modules. The steps are defined for both a command line user conversant with Linux and for a user preferring button-click operations.

Command Line Mode using Makefile

To remove the driver, type in the following command line in the terminal in the driver folder.

```
$ make remove
```

Command Line Mode using Executable Scripts

The executable scripts provided can be run as-is on the command line.

For removal of driver modules:

```
$./s6_trd_driver_remove
```

Mouse-Click Driven Mode

To compile the driver, navigate to the `s6_pcie_dma_ddr3_gbe_axi` folder.

To unload the driver modules (Figure 2-46), click on **s6_trd_driver_remove**.

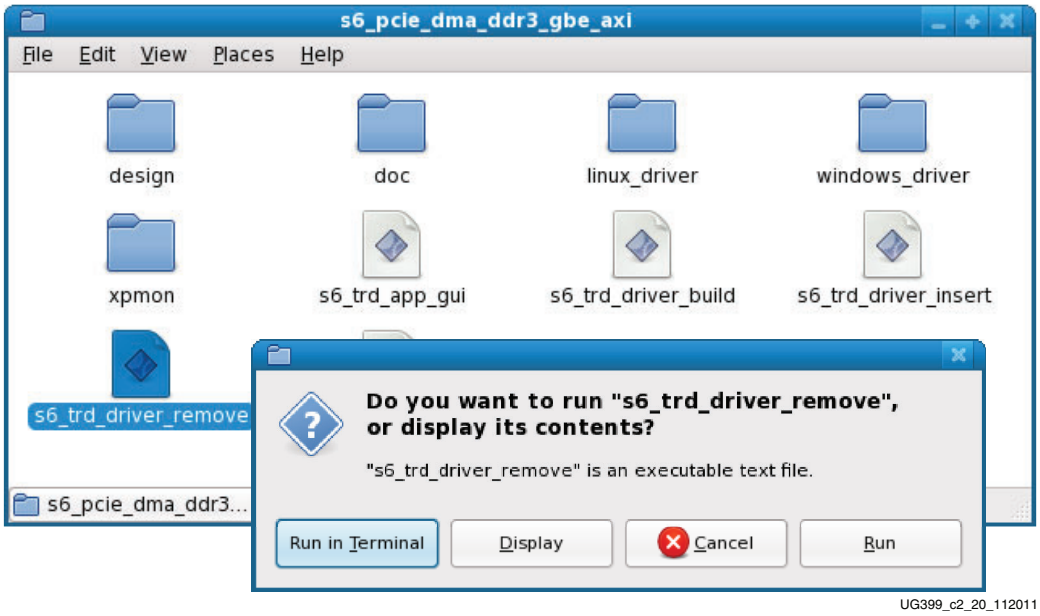


Figure 2-46: **Executing Driver Removal Script**

2. To shut down the system select the **System→ Shutdown** option. The LiveCD will be ejected. Follow the onscreen instructions. Any files saved during the LiveCD session are not accessible the next time LiveCD is run.

IP Cores with TRD

The Xilinx LogiCORE IPs required for the TRD are shipped with the TRD. The following cores/netlists are located in the `design/coregen_ip` directory:

- `s6_pcie_axi_st`
- Netlist for FIFO

For versions of the IP cores used, refer to the readme.txt (provided). The AXI_Ethernet IP is present in the EDK install directory and is used from there.

The MIG is generated from the CORE Generator tool. The scripts provided for implementation and simulation generate the MIG IP core as a first step. This is accomplished in the following manner (described for reference).

Open a terminal window (on Linux) or a DOS command (on Windows) and navigate to the `design/coregen_ip` directory. Type the following on the command line:

```
coregen -b mig_axi_mm.xco -p coregen.cgp
```

Additionally a golden set of XCO files are also provided under the `reference/xco_files` directory so that the cores can be regenerated.

Generating MIG cores overwrites the `mig_axi_mm.xco` provided. To regenerate the core, copy the `mig_axi_mm.xco` and `mig.prj` from the `design/reference/xco_files`.

Information on the version of IP cores used can be obtained from the `readme.txt` provided with the design directory.

Implementing the Design

This section explains how to implement the design after making certain modifications. The implementation flow is provided in two modes, script based and PlanAhead™ based (software GUI flow).

Script-Based Flow

Implementation scripts for both Linux and Windows operating systems are provided under the `design/implement` folder. Implementation requires the appropriate Xilinx tools environment to be set up.

Implementation on Linux

1. Navigate to the `design/implement/lin` folder.
2. To generate a design using GMII through a MARVELL PHY, execute the following on a command line:

   ```
   $ ./implement_gmii.sh
   ```

   To generate a design in the 1000BASE-X mode, execute the following on a command line:

   ```
   $ ./implement_1000basex.sh
   ```

3. After completion of the implementation process, a results folder is created with the bitstream, all the reports, and intermediate implementation results.

Implementation on Windows

1. Navigate to the `design/implement/nt` folder
2. To generate a design using GMII through a MARVELL PHY, execute `implement_gmii.bat`
3. To generate a design in the 1000BASE-X mode, execute `implement_1000basex.bat`

After completion of the implementation process, a results folder is created with the bitstream, all the reports, and intermediate implementation results.

PlanAhead-Based Flow

The Spartan-6 Connectivity TRD provides support for PlanAhead flow. Execution of this flow requires Xilinx ISE (System or Embedded edition) tools to be installed and environment variables set for XILINX, XILINX_PLANAHEAD, and XILINX_EDK.

- To implement a GMII design, navigate to `design/implement/planahead_flow_gmii.`
- To implement a 1000BASE-X design, navigate to `design/implement/planahead_flow_gmii.`

The scripts for GMII design flow `planahead_gmii.[bat|sh]` and for 1000BASE-X flow `planahead_1000basex.[bat|sh]` perform the following:

- MIG IP core generation
- License pre-processing for the Soft Tri-Mode Ethernet MAC and placement of the wrapper file in the `design/ip_cores/axi_ethernet` folder
- PlanAhead GUI opening with the project loaded

The scripts are to be executed on a Linux terminal or Windows command prompt after sourcing the Xilinx environment or in the ISE Design Suite command prompt.

The user can subsequently navigate through the files and design hierarchy. Implementation flow is performed with these steps:

1. Click on **Synthesize** in the Project Manager window. This runs the synthesis and on completion provides a synthesis completion status window.
2. Click on **Implement** in the Project Manager window. This runs the implementation flow and on completion provides an implementation completion status window.
3. Bitstream generation can be commenced by clicking on the option provided in the Implementation window or through Project Manager (**Program & Debug** → **Generate Bitstream**). The options for bitstream generation can be specified in the window which opens.

The results of PlanAhead flow are available in sub-folders under `planAhead_run1`. The bitstream can be found in the folder `<project_name>.runs/impl_1`.

The TRD by default uses the evaluation version of the DMA IP which times out after 12 hours. To use the full version of the DMA IP:

1. Obtain a full DMA license as documented in UG665: *Spartan-6 FPGA Connectivity Kit Getting Started Guide*. Licensing requires registration.
2. After a full license is obtained, depending on the flow being used (script based or PlanAhead) modify the respective scripts to use the DMA netlist from `ip_cores/dma/netlist/full` folder instead of `ip_cores/dma/netlist/eval`. This requires path change and search directory path change in the scripts.

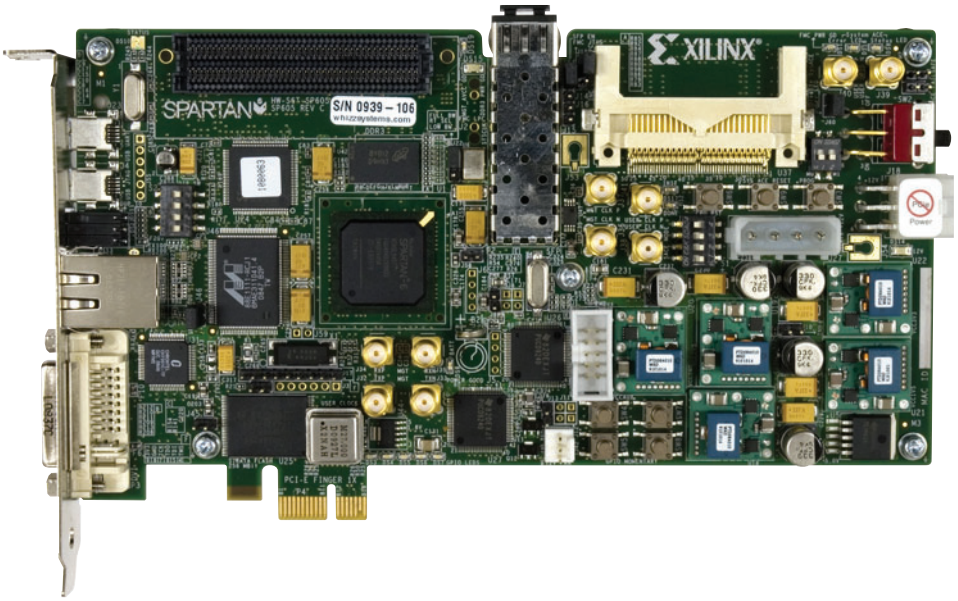
Programming the SP605

The SPI flash on the SP605 board is pre-programmed with the TRD file. This section provides a check on the on-board jumper settings after modification or use by another user.

Board Settings

This section shows the jumper settings required on the board (Figure 2-47) for the TRD to work when programmed in SPI x4 Flash mode.

1. Set the mode switch SW1 to `01` (M1 = 0 and M0 = 1).
2. Jumper J46 is required to be ON for the FPGA to be programmed with the onboard SPI x4 flash.
3. All other jumpers and switches are required to be in default position. Refer to UG526, *SP605 Board User Guide* for default jumper/switch settings.



UG399_c2_21_091510

Figure 2-47: **Board Setup**

Board Programming

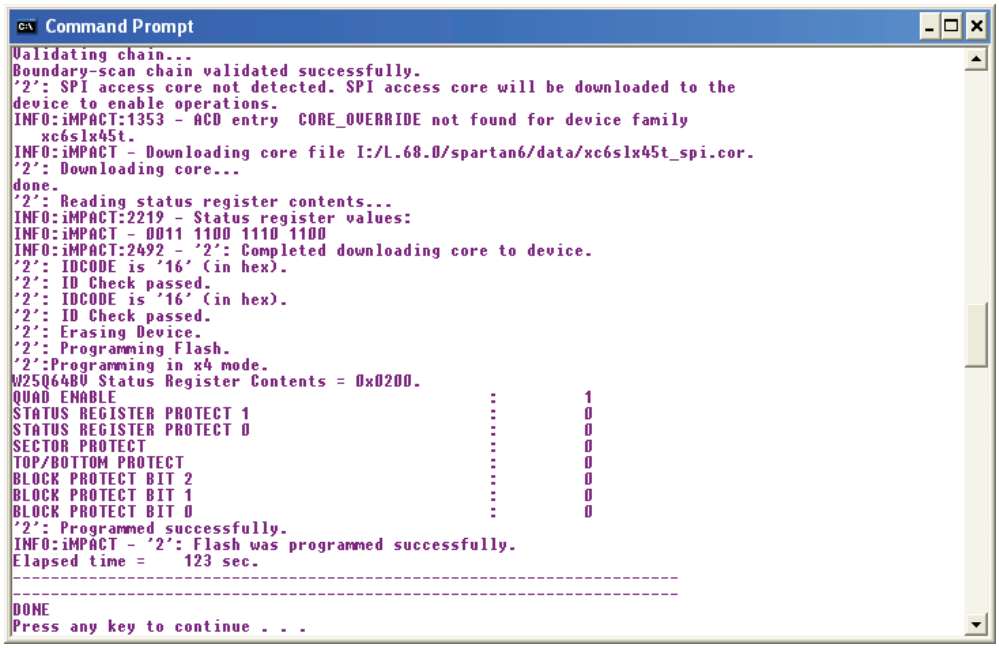
This section explains the programming of the SP605's onboard SPI x4 Flash.

1. Connect the USB download cable between the SP605 board and the PC. The SP605 board is required to be powered ON for programming. Use the power-supply brick provided with the connectivity kit for this purpose.
2. Navigate to the `design/reference/configuration` folder and copy the flash programming file to this folder.
3. Open the file `spi_program.cmd` and modify the name of the MCS file to be programmed to be the same as the one copied.

   On Linux, execute the following on the command line of a terminal window:

   ```
   $ impact -batch spi_program.cmd
   ```

On Windows, double-click on the `spi_program.bat` file. Programming on Windows opens the DOS command window as shown in Figure 2-48.



UG399_c2_22_091510

Figure 2-48: **Programming Status on Windows**

Testing 1000BASE-X Mode

The design supports the 1000BASE-X mode of operation. To test this, the additional requirements are to use an SFP to RJ45 adapter (for a copper cable) or an SFP optical adapter (for an optical cable). These items are not part of the SP605 connectivity kit.

To test the 1000BASE-X mode of operation, program the device with either `sp605_use_1000basex.bit` or `sp605_use_1000basex.mcs` provided under the `reference/configuration` folder.

Jumpers J44 and J22 are required to be in default position. Refer to UG526, *SP605 Board User Guide* for default jumper settings.

To test the 1000BASE-X mode of operation, the Ethernet driver needs to be compiled by enabling an additional macro.

- Navigate to the `driver/xgbeth` folder
- Open the **Makefile** and add **-DUSE_1000BASEX** to the end of the line defining **EXTRA_CFLAGS**

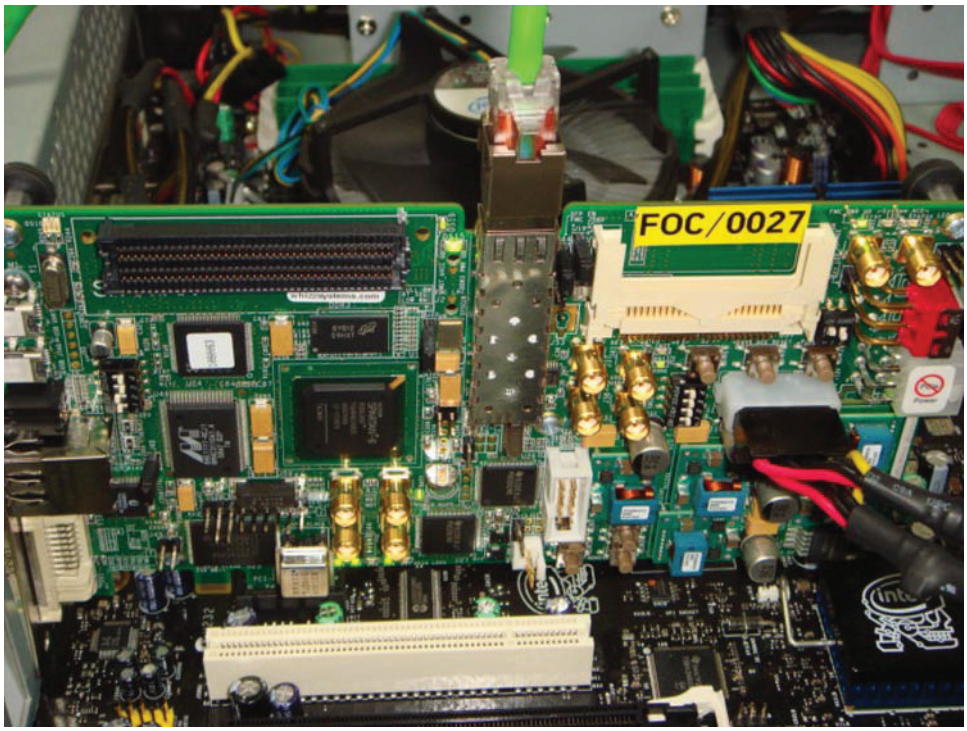
Hardware programming of the 1000BASE-X design and recompilation of drivers with this change works in the same flow as described in the Hardware Bring Up, Software Bring Up, and Network Bring Up sections.

The LEDs indicated as Ethernet status indicators in Figure 2-4 do not apply when using a 1000BASE-X design.

The setup with 1000BASE-X using an SFP to RJ45 adapter is shown in Figure 2-49.

The 1000BASE-X mode in this TRD was tested with the HP 378928-B21 Cisco Gigabit Ethernet RJ45 SFP module. This is a RJ45 SFP module that can connect to copper media.

Note: The 1000BASE-X mode of operation requires a 1 Gb/s Ethernet connection.



UG399_c2_23_091510

Figure 2-49: **Board Setup with an SFP to RJ45 Adapter**

Simulation

This section details the simulation environment provided with the design. This simulation environment is provided to show the functionality of the design. The simulation environment showcases basic traffic movement and demonstrates the functionality of the various components.

Overview

The simulation environment (Figure 2-50) consists of the design (commonly referred to as the design under test or DUT) connected to a Virtex-6 FPGA root-port model for PCI Express.

The root-port model for PCI Express is a limited test-bench environment that provides a test-program interface. The root-port model provides a source mechanism for generating downstream PCI Express traffic to stimulate the DUT and a destination mechanism for receiving upstream PCI Express traffic from the DUT in a simulation environment.

This simulation environment is built on top of the simulation environment generated by Virtex-6 FPGA integrated block for PCI Express.

The simulation environment consists of the following:

- A root-port model connected to DUT
- Transaction Layer Packet (TLP) generation tasks for various programming operations
- Test cases to generate different traffic scenarios

The simulation environment demonstrates the basic functionality of the design through various test-cases. The simulation environment shows the following:

- Configuration transactions for PCI Express
- DMA initialization
- AXI_Ethernet initialization
- Virtual FIFO initialization
- Traffic movement over Ethernet
- Traffic movement over virtual FIFO
- Depending on the test case selected, it shows:
  - Interrupt handling (MSI or legacy interrupt)
  - DMA disable operation
  - Packet spanning across multiple descriptors

The simulation test bench sets up the buffer descriptors and corresponding buffers based on the compilation macros explained in detail in User Controlled Macros. The length of the packets in a network path is selected randomly and the packet size is fixed for a memory path.

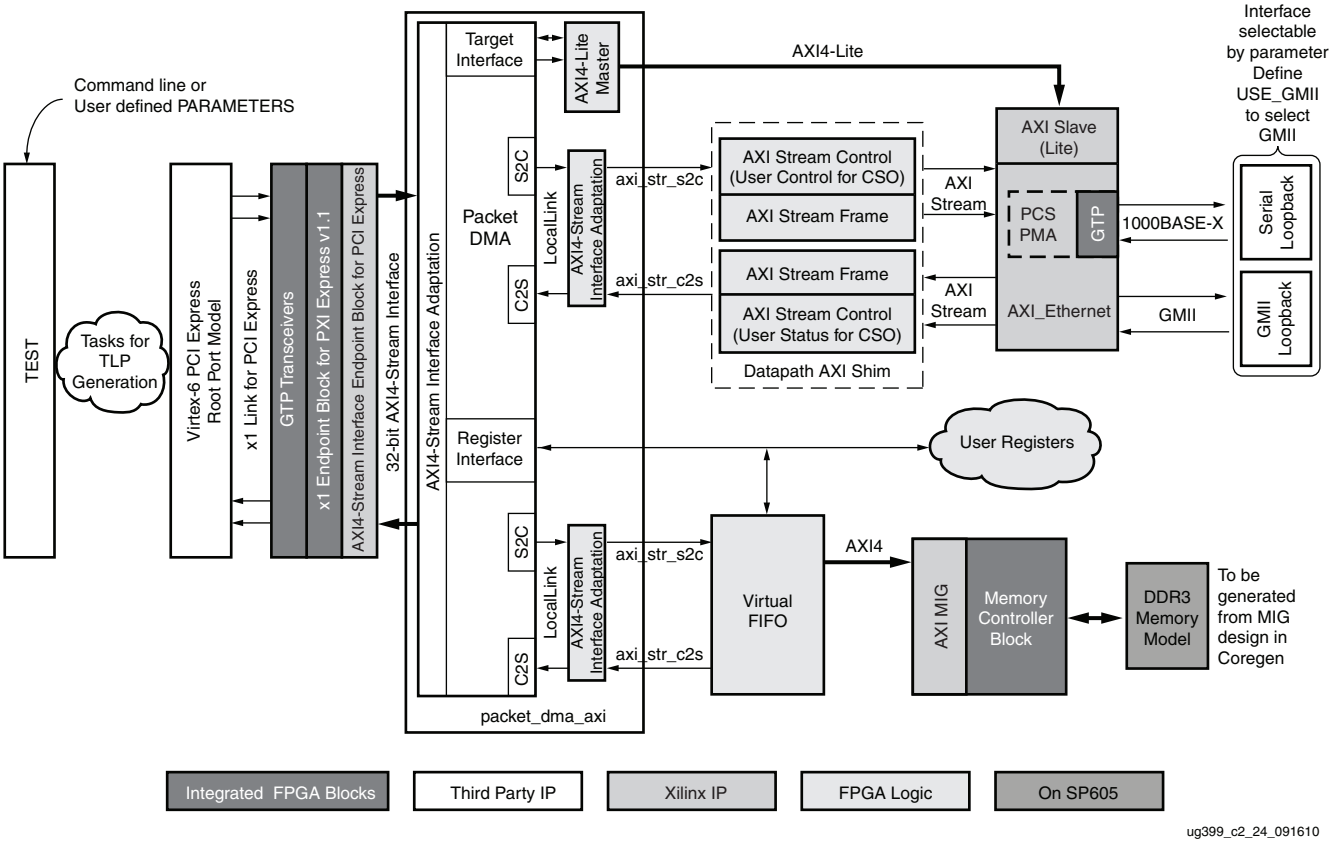


Figure 2-50: **Simulation Overview**

The simulation environment uses the Micron DDR3 memory model and connects the Ethernet interface in loopback mode (either GMII or serial-interface loopback depending on the mode selected).

The simulation environment creates log files during simulation. These log files contain a detailed record of every TLP that was received and transmitted respectively, by the root port model.

User Controlled Macros

The simulation environment allows definition of some macros controlling test bench configuration. These values can be changed in the `user_defines.v` file under the `design/sim/include` folder.

Table 2-1 describes the parameters that can be modified.

Table 2-1: **User Controlled Macro Description**

Macro Name	Default Value	Description
CH0	Defined	Enables network path initialization and traffic.
CH1	Not defined	Enables memory path initialization and traffic.
CH0_S2C_BD_COUNT	6	Number of S2C descriptors set up for network path. For basic_test, this is also the total number of packets to be transmitted. This value can be increased to a maximum of 25 for one packet per descriptor.

Table 2-1: **User Controlled Macro Description *(Cont'd)***

Macro Name	Default Value	Description
CH1_S2C_BD_COUNT	6	Number of S2C descriptors set up for memory path. For basic_test, this is also the total number of packets to be transmitted. This value can be increased to a maximum of 25 for one packet per descriptor.
USE_GMII	Not defined	Enables GMII loopback for Network Path. By default, 1000BASE-X mode is enabled.
LEGACY_INTR	Not defined	Enables legacy interrupts for PCI Express when defined, otherwise, MSI is enabled.
DETAILED_LOG	Not Defined	Enables a detailed log of each transaction.

Test Selection

The test environment generates packets of random length for network path and builds appropriate Ethernet frames. For memory path, packets of fixed length (1024 bytes) are generated.

Table 2-2 describes the various tests provided by the simulation environment.

Table 2-2: **Test Description**

Test Name	Description
basic_test	Basic Test: This test runs a defined number of packets per application. 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.
test_interrupts	Interrupt Test: Sets the interrupt bit in descriptor and enables interrupt registers. This test also shows interrupt handling by acknowledging relevant registers.
dma_disable	DMA Disable Test: Shows a DMA disable operation sequence on a channel.

Simulating the Design with ISim

The relevant script for simulation with ISim is provided. On Linux, open a terminal and source the Xilinx environment.

- Navigate to the `design/sim/isim_lin` folder.
- Execute this script:

```
$ ./simulate_isim.sh
```

On Windows, open a command prompt and source the Xilinx environment.

- Navigate to the `design/sim/isim_nt` folder.
- Execute the `simulate_isim.bat` file.

Execution of these scripts results in compilation, elaboration, opening of the ISim GUI, running simulation, and tracing of the waveform. By default, simulation runs the Basic Test. The test name can be changed by using the following syntax while running the ISIM generated executable (board.exe) through the script:

```
"-testplusarg TESTNAME=basic_test"
```

Simulating the Design with ModelSim

The simulation environment provides scripts for simulation with ModelSim. Simulation of the TRD requires compilation of the Xilinx EDK Simulation Libraries for ModelSim. Xilinx provides a tool called *compxlib* for this purpose. UG628: *Command Line Tools User Guide* (Chapter 25) contains details on options specific to the ModelSim simulator version.

To run the simulation, execute the following scripts at the command prompt after setting the required environment.

ModelSim (`design/sim/mti` folder)

- On Linux

```
$ ./simulate_mti.bat
```

- On Windows

  Execute `simulate_mti.bat`. This invokes the ModelSim GUI and runs simulation.

By default, the simulation script file specifies the Basic Test to be run using the following syntax:

```
"+TESTNAME=basic_test"
```

The test selection can be changed by specifying a different test-case argument as specified in Table 2-2 in the scripts provided.

XILINX®

Chapter 3

Functional Description

This chapter describes the hardware design and software driver architecture in detail.

Hardware Design Description

This section discusses the hardware design architecture in detail. Figure 3-1 shows a detailed view of the TRD with AXI4 interface. The network application is denoted by USER_APP0 and the external memory application is denoted by USER_APP1. The hardware design contains several key IP components which are stitched together with additional FPGA logic to create the entire TRD framework.

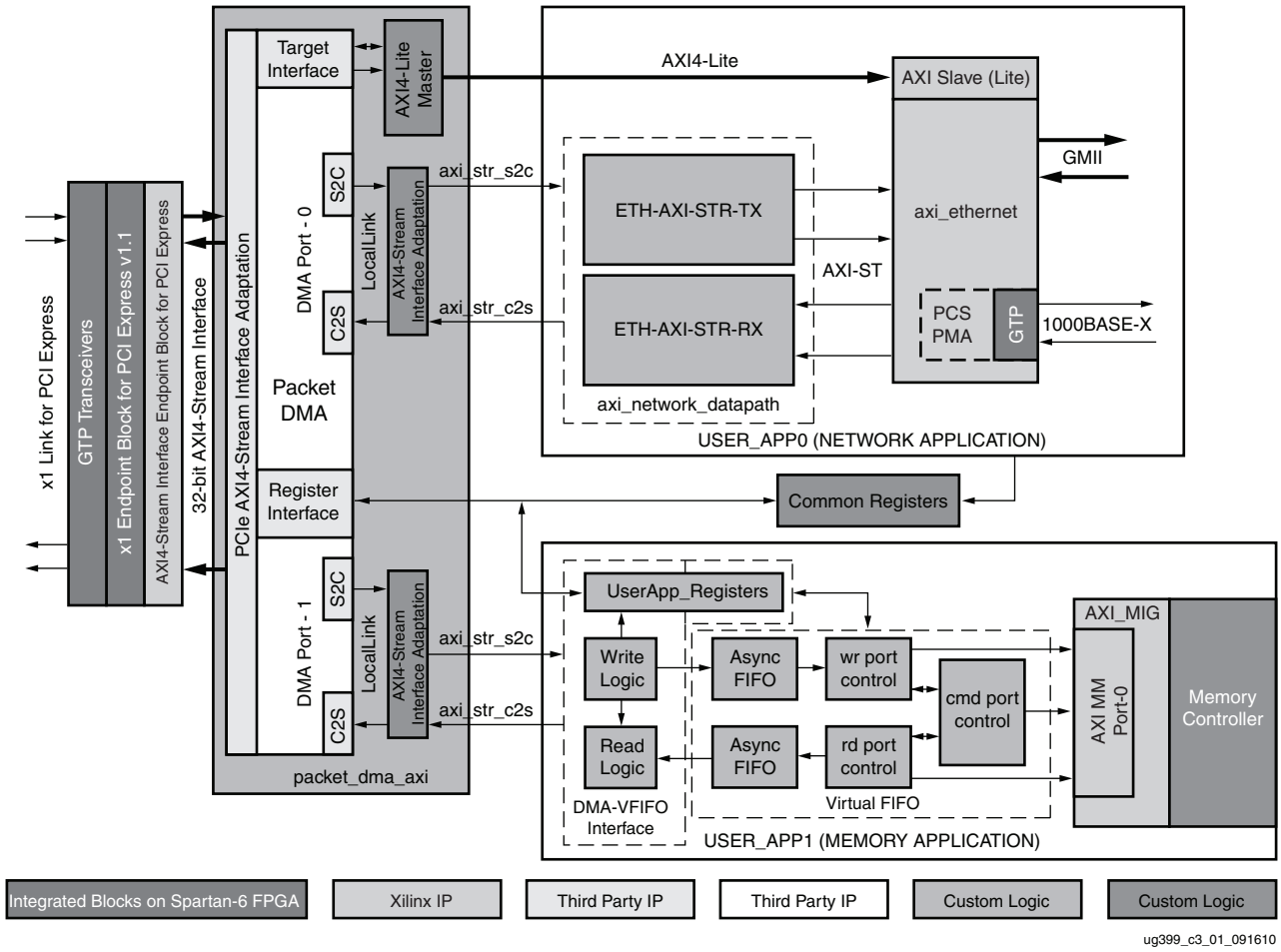


Figure 3-1: **Detailed Design Block Diagram**

The hardware architecture is described in detail under the following sections:

- Base Design Architecture: Describes the Endpoint block for PCI Express and the Packet DMA block with AXI4 protocol adaptation layer
- Network Path Architecture: Details the network path, use of various IPs, and developed FPGA logic
- Memory Path Architecture: Details the memory path, use of various IPs, and developed FPGA logic

Base Design Architecture

The TRD is made up of two key base building blocks: the integrated Endpoint for PCI Express (PCIe) and the Packet DMA (including the adaptation layer designed for AXI4). The Endpoint provides a high-speed serial bus interface between the Spartan®-6 FPGA and a host system. Any movement of data between system memory and hardware over the PCI Express interface involves huge memory transfers, which could potentially consume a large amount of processor bandwidth if entirely managed by the processor. This transfer overhead is reduced by using a bus mastering Packet DMA controller to transfer data with minimal processor intervention. These building blocks together form the basis of the network application and external memory application detailed in this chapter.

Integrated Endpoint Block for PCI Express

The LogiCORE IP Spartan-6 FPGA Integrated Endpoint Block for PCI Express provides wrappers around the integrated Endpoint block for PCI Express. It is a PCI Express v1.1 compliant core supporting x1 lane width operating at a 2.5 Gb/s line rate per direction. The wrapper combines the integrated Endpoint block with GTP transceivers, clocking, and reset logic to provide a simplified AXI4-Stream user interface. The user interface data width is 32-bits operating at 62.5 MHz.

For details on this core, refer to UG672, *Spartan-6 FPGA Integrated Endpoint Block for PCI Express User Guide.*

This core is generated with two 32-bit base-address registers (BAR) for this design. One BAR maps to DMA registers and another BAR maps to the AXI_Ethernet and external PHY registers.

A device ID value of `6011` is used and class code is set to `07_80_00` to indicate a communication controller.

By selecting the SP605 board, a 125 MHz reference clocking scheme is automatically picked up. All other settings remain at the default values.

Packet DMA with AXI4 Protocol Adaptation Layer

This block is a four-channel bus-mastering packet direct memory access (DMA) controller.

This controller helps move high-speed data between the system memory and FPGA using PCI Express. It enables the simultaneous operation of two different user applications through the four channels provided. Each channel of DMA is either a system-to-card (S2C or transmit) or a card-to-system (C2S or receive). The four-channel DMA used in this design has two system-to-card (S2C) or transmit channels and two card-to-system (C2S) or receive channels. The DMA interfaces to the Integrated Endpoint Block for PCI Express through the AXI-Stream transaction interface. The DMA provides an AXI4-Stream interface for datapath and an AXI4 Lite interface for control plane.

Each DMA channel has a set of independent registers. Registers specific to this TRD are described in DMA Registers in Appendix A. Further details of various registers can be obtained from the NorthWest Logic (NWL) Packet DMA user guide.

An AXI4 protocol adaptation layer is designed around the Northwest Logic-provided Packet DMA core. The AXI4 protocol adaptation layer performs these functions, as shown in Figure 3-2:

- Provides the AXI4-Stream user interface in place of the LocalLink-like interface
- Provides the AXI4-Lite Master interface in place of the target interface

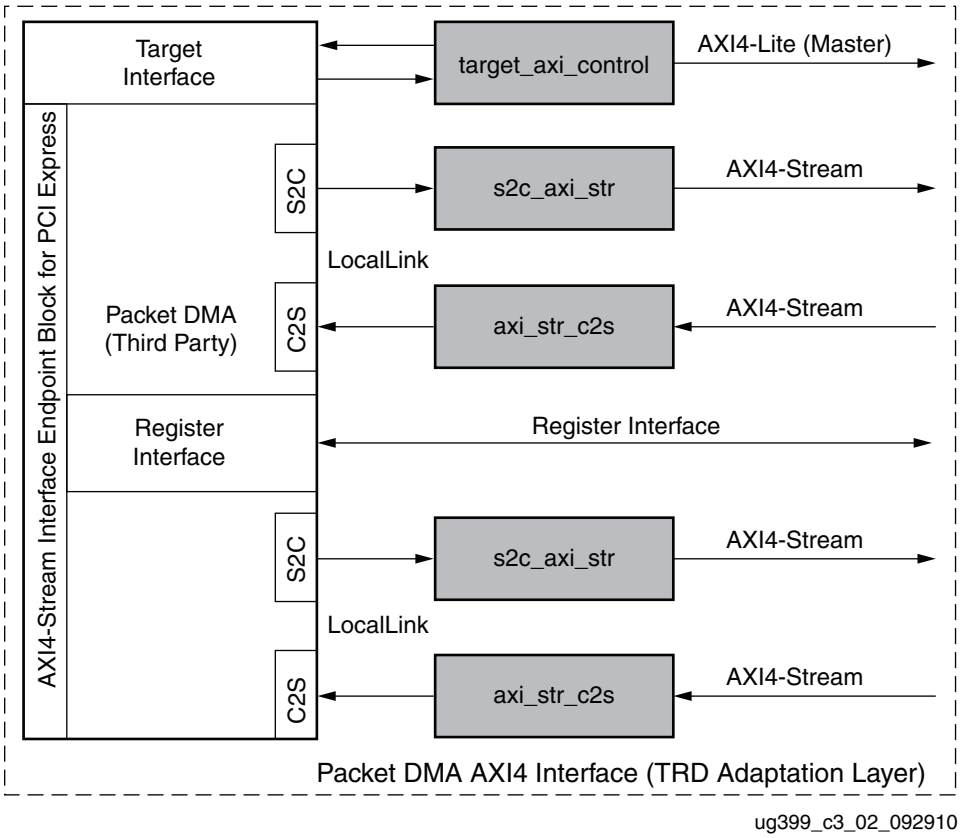


Figure 3-2: **Packet DMA AXI4 Interface (TRD Adaptation Layer)**

Control Plane Interface

The DMA controller requires a 64-KB register space mapped to BAR0. Certain address ranges in BAR0 are used by the DMA, the rest are available to the user to implement registers for user logic. All other BARs are also available to the user, as required.

The user is required to perform read or write operations on the user-implemented registers. All other aspects, including building of TLPs or extracting data from memory write TLPs, are managed by the DMA controller.

The Packet DMA provides these interfaces for register space mapped to BARs in the configuration space for PCI Express:

- Register interface for registers 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 through this interface.

  The register interface requires that the register read data be available immediately one cycle after the read request is issued. Therefore the register interface is retained "as is" and no AXI4-Lite wrapper is built over it.

- Target interface for registers mapped to BAR other than BAR0

  Although the target interface can support multiple DWORD transactions, it is only used for one DWORD transactions in this TRD. Therefore an AXI4-Lite Master interface is provided in place of target interface.

AXI4-Lite Master Interface

An AXI4-Lite interface is built around the target interface. With the AXI4-Lite interface, the control interface can be branched out using the AXI4 Interconnect, if required.

For details on target interface signals, refer the Packet DMA user guide. Table 3-1 provides descriptions of the AXI4-Lite interface signals.

Table 3-1: **AXI4-Lite Interface Signal Description**

Signal Name	Direction	Description
m_axi_aclk	Output	Output Clock
m_axi_awid	Output	Write address ID
m_axi_awaddr[31:0]	Output	Write address
m_axi_awvalid	Output	Write address valid
m_axi_awready	Input	Write address ready
m_axi_wdata[31:0]	Output	Write data
m_axi_wstrb[3:0]	Output	Write data strobe
m_axi_wvalid	Output	Write data valid
m_axi_wready	Input	Write data ready
m_axi_bresp[1:0]	Input	Write response
m_axi_bvalid	Input	Write response valid
m_axi_bready	Output	Write response ready
m_axi_araddr[31:0]	Output	Read address
m_axi_arvalid	Output	Read address valid
m_axi_arready	Input	Read address ready
m_axi_rdata[31:0]	Input	Read data
m_axi_rvalid	Input	Read data valid
m_axi_rready	Output	Read data ready
m_axi_rresp[1:0]	Input	Read data response

In the design, because this master interface connects to just one slave, the AXI4-Lite Interconnect instance is not used and one-to-one connections are done.

For scalability requirements when connecting to multiple slaves, the AXI4-Lite interconnect can be used.

Data Plane Interface

This section describes the data plane interface and corresponding AXI4-Stream interface signals. An AXI4-Stream protocol adaptation layer is defined to convert the DMA user interface to an AXI4-Stream interface. Table 3-2 defines the mapping of DMA user interface signals to AXI4-Stream interface signals.

Table 3-2: **Data Plane Interface and Corresponding AXI4-Stream Interface Signals**

DMA User Interface Signal	AXI4-Stream Interface Counterpart	Comments
Transmit Path		
s2c_sop	N/A	No concept of SOP in AXI4-Stream interface
s2c_eop	axi_str_s2c_tlast	Signal end of packet
s2c_src_rdy	axi_str_s2c_tvalid	
s2c_dst_rdy	axi_str_s2c_tready	
s2c_valid[1:0]	axi_str_s2c_tstrb[3:0]	2-bit valid from DMA converted to 4-bit AXI strobe. • 00: 1111 • 01: 0001 • 10: 0011 • 11: 0111
s2c_data[31:0]	axi_str_s2c_tdata[31:0]	
s2c_user_control[63:0]	axi_str_s2c_tuser[63:0]	Sideband signal, special care must be taken when using width converters
Receive Path		
c2s_sop	N/A	Because the AXI4-Stream interface has no SOP counterpart, the AXI4-Stream protocol adaptation layer needs to build SOP
c2s_eop	axi_str_c2s_tlast	
c2s_src_rdy	axi_str_c2s_tvalid	
c2s_dst_rdy	axi_str_c2s_tready	
c2s_data[31:0]	axi_str_c2s_tdata[31:0]	
c2s_data[31:0]	axi_str_c2s_tstrb[3:0]	Mapping from AXI strobe to DMA: • 0001: 01 • 0011: 10 • 0111: 11 • 1111: 00
c2s_user_status[63:0]	axi_str_c2s_tuser[63:0]	

Scatter Gather Operation

A scatter gather scheme 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.

The term *scatter* refers to the ability to write the data into different memory locations (basically scattering data in memory). The term *gather* refers to the capability to gather data from different locations in memory and build a packet out of it.

Buffer descriptors, as the name suggests, describe the data buffer. Each buffer descriptor is eight double words (DW) in size. One DW has four bytes. Eight DWs 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.

The buffer descriptor layout is slightly different for S2C and C2S directions as highlighted in Figure 3-3. The various fields are described in Table 3-3.

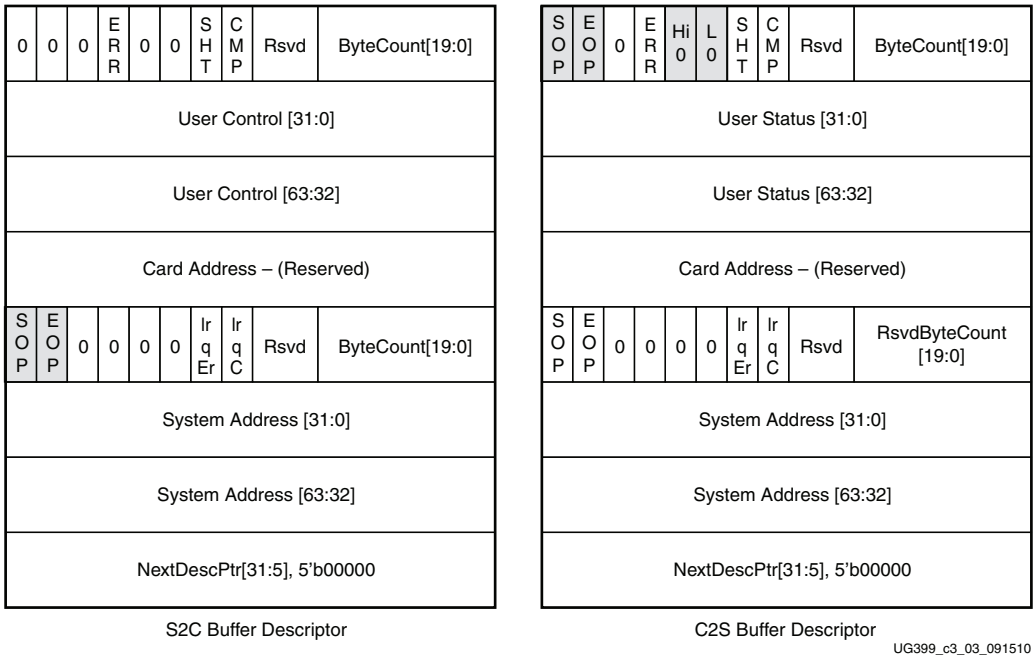


Figure 3-3: **Buffer Descriptor Layout**

Table 3-3: **Buffer Descriptor Field Description**

Descriptor Fields	Functional Description
SOP	Start of Packet: In the S2C direction, indicates the start of a new packet. In the C2S direction, DMA updates this field to indicate to software the start of a new packet.
EOP	End of Packet: In the S2C direction, indicates the end of current packet. In the C2S direction, DMA updates this field to indicate to software the end of the current packet.
ERR	Error: Set by the DMA on a descriptor update to indicate an error while executing that descriptor.
SHT	Short: Set when the descriptor is completed with a byte count less than the requested byte count. It is common for C2S descriptors having an EOP status set and should not be treated as an error in the C2S direction, but should be analyzed when set for S2C descriptors.
CMP	Complete: This field is updated by the DMA to indicate to the software the completion of an operation associated with that descriptor.
Hi 0	User status High is zero: Applicable only to C2S descriptors. This is set to indicate User Status [63:32] = 0. This bit is also used to ensure coherency to make sure that the descriptor status is valid.

Table 3-3: **Buffer Descriptor Field Description *(Cont'd)***

Descriptor Fields	Functional Description
L 0	User status Low is zero: Applicable only to C2S descriptors. This is set to indicate User Status [31:0] = 0. This bit is also used to ensure coherency to make sure that the descriptor status is valid.
Irq Er	Interrupt on Error: This bit indicates to the DMA to issue an interrupt when the descriptor results in error.
Irq C	Interrupt on Completion: This bit indicates to the DMA to issue an interrupt when an operation associated with the descriptor is completed.
ByteCount[19:0]	Byte Count: In S2C direction, this indicates to the DMA the byte count queued up for transmission. In C2S direction, the 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. Depending on the packet size, the DMA might or might not utilize the entire buffer.
User Control/User Status	User Control or Status Field: In S2C direction, this is used to transport application specific data to the DMA. In C2S direction, the DMA can update application specific data in this field.
Card Address	Card Address Field: Reserved for packet DMA.
System Address	System Address: Defines the system memory address where the buffer is to be fetched from (for S2C direction) or written to (for C2S direction).
NextDescPtr	Next Descriptor Pointer: This field points to the next descriptor in the linked list. All descriptors are required to be 32-byte aligned.

Packet Transmission

Initially, the software driver prepares a ring of descriptors in system memory and writes the starting and ending addresses of the descriptors in the relevant channel registers in the DMA. Once enabled, the DMA fetches the descriptor followed by the buffer it points to. Data is fetched from the host memory and made available to the user application at the DMA S2C streaming interface. The packet interface signals (for example, start-of-packet, end-of-packet) are built from the control fields in the descriptor. The information present in the user-control field is made available during `s2c_sop`.

To indicate completion of the data fetch corresponding to this descriptor, the DMA engine updates the first DW of the descriptor by setting a completed bit by highlighting the Status and ByteCount field as shown in Figure 3-4. Figure 3-4 also shows the corresponding AXI4-Stream interface signals for S2C direction. The software driver analyzes the completed field to free up the buffer memory and also move the descriptor back under software ownership.

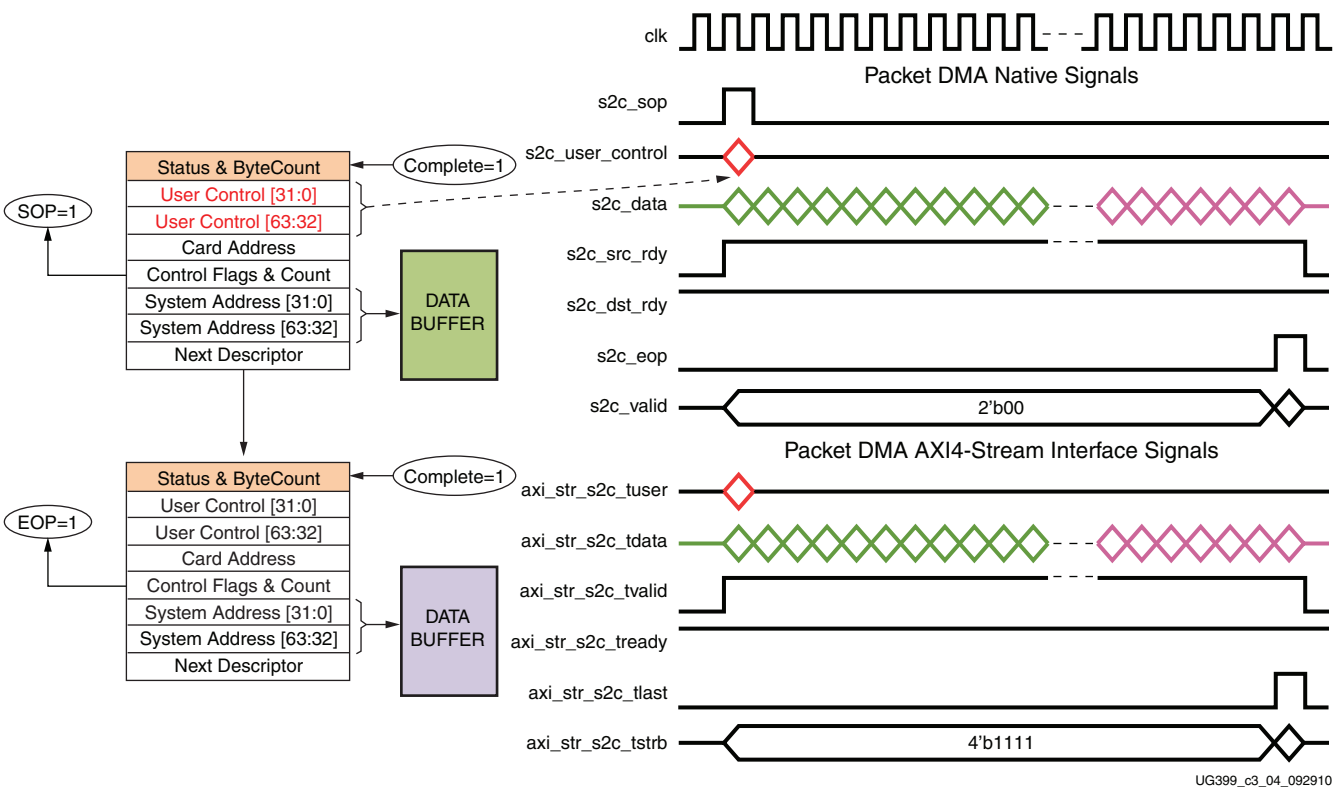


Figure 3-4: **Data Movement from System to Card**

Packet Reception

The software driver prepares a ring of descriptors with each descriptor pointing to an empty buffer and programs the starting and ending addresses in relevant DMA channel registers. The DMA reads the descriptors and waits for the user application to provide data on the C2S streaming interface (Figure 3-5). When the user application provides data, the DMA writes the data into one or more empty buffers pointed to by the pre-fetched descriptors. After the packet is written to host memory, the DMA updates the status in the descriptor. The user status field is considered valid only during `c2s_eop`. When updating EOP status, the DMA engine updates three DWs in descriptor otherwise, it only updates one DW. The completed bit in the updated status field indicates to the software driver the availability of data received from the hardware. Figure 3-5 also shows the corresponding AXI4-Stream interface signals.

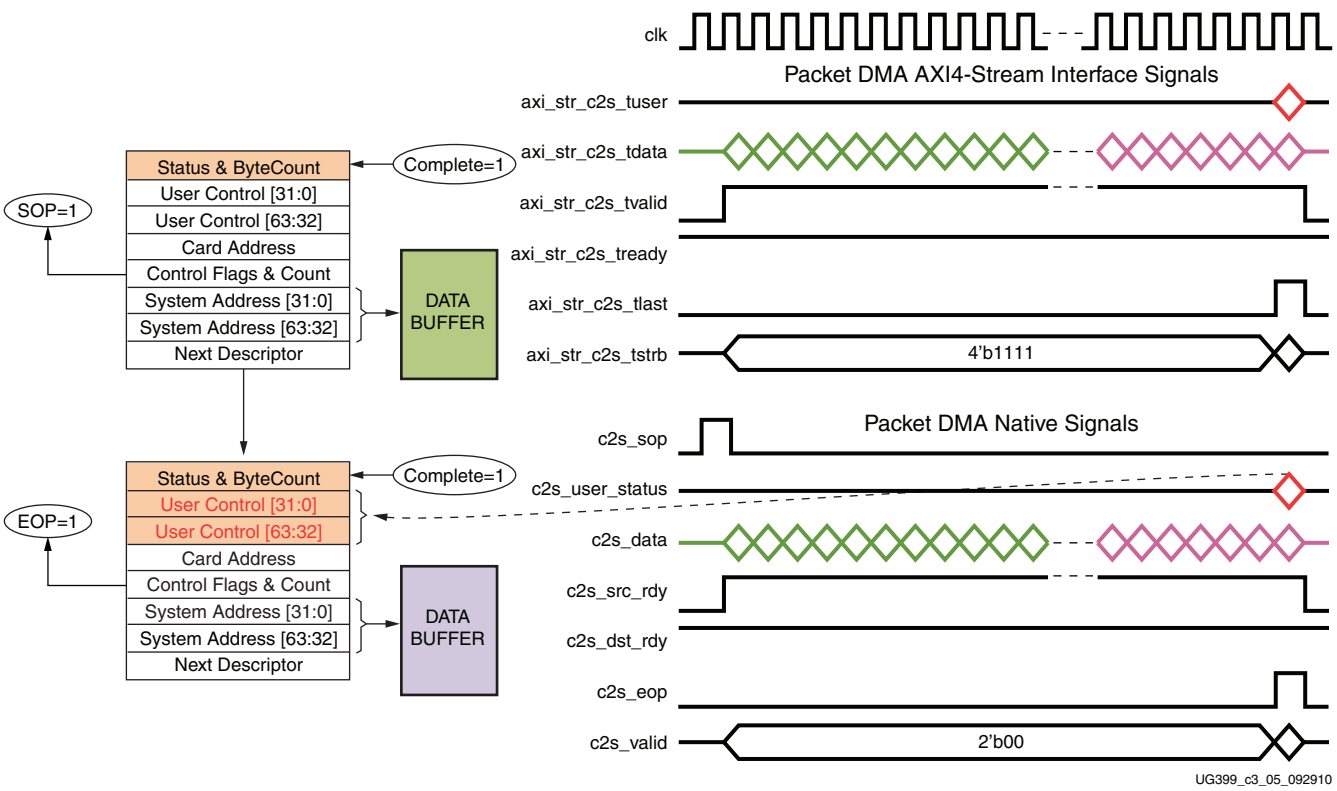


Figure 3-5: **Data Movement from Card to System**

Transmit and receive channel registers are updated in between (dependent on data availability in the transmit direction and dependent on periodic time-out or interrupts in the receive direction) to ensure uninterrupted data flow to the DMA. By analyzing the buffer descriptor, the software driver can read the status of the associated DMA transfers, fetch user information on receive channels, and determine the completion of transfer.

Network Path Architecture

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 through the base design.

The NIC application supports the following modes of operation:

- GMII mode interfacing to an external Ethernet PHY (typically used to connect to copper Ethernet networks)
- 1000BASE-X mode using the Spartan-6 FPGA GTP transceivers (typically used to connect to optical-fiber Ethernet networks)

The following sections describe the various IP cores and relevant features used.

AXI_Ethernet IP

The AXI_Ethernet IP is used with a soft TEMAC option for Spartan-6 FPGAs. Options for these features are provided:

- Partial checksum offload

- Ethernet statistics monitoring
- Choice of physical interface - either GMII or 1000BASE-X
- Jumbo frame

The AXI_Ethernet IP provides an AXI4-Lite slave interface to enable access to its registers. The external PHY registers are accessed through an MDIO interface. Certain registers are described in Appendix A, Register Description. For more details, refer to DS759, *AXI_Ethernet Data Sheet*. This slave interface is connected to the AXI4-Lite master adaptation layer built around the Packet DMA-provided target interface.

The AXI4-Stream interface model used on the datapath is called "Packetized and Aligned Strobe". The legacy LocalLink interface used with header and footer is split into dual channel AXI4-Stream interfaces (one for data and one for control/status) in both the transmit and receive directions (see Figure 3-6).

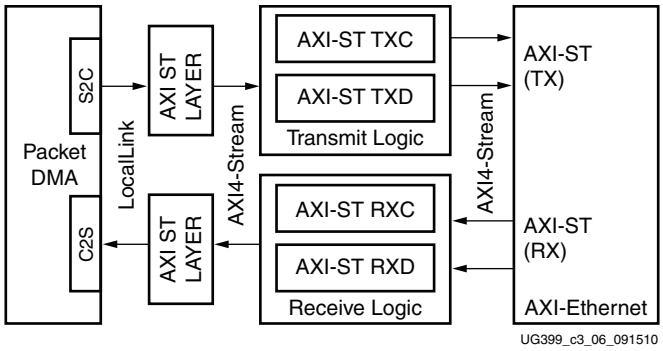


Figure 3-6: **Connection to AXI_Ethernet IP**

The logic built in the data plane performs these functions:

- The transmit logic interfaces the S2C AXI4-Stream interface to AXI_Ethernet transmit datapath. It provides AXI4-Stream control and data interface signals to the transmit portion of the AXI_Ethernet interface.
- The receive logic interfaces the AXI_Ethernet receive datapath to the C2S AXI4-Stream interface. It drains out the AXI-Stream interface data and status signals from the AXI_Ethernet receive interface.

Checksum Offload

For TCP or UDP Ethernet protocols, data integrity is maintained by calculating and verifying checksum values over the frame data. Checksum calculation in software can be relatively slow and uses significant processor utilization for large frames at high Ethernet data rates.

This design demonstrates hardware acceleration by offloading the TCP/UDP checksum calculation to hardware. Checksum information is passed between the software and hardware (AXI_Ethernet IP) by user control and user status fields in the buffer descriptors which are subsequently mapped to AXI4-Stream interface control in transmit and AXI-Stream interface status in receive directions.

This section defines the field mappings for checksum offload (see Table 3-4). In the transmit direction, the user control information is available as `axi_str_s2c_tuser` (see Figure 3-7).

Table 3-4: **Field Mappings for Checksum Offload (Transmit Direction)**

S2C Descriptor User Control Field	Checksum Field Mapping	Description
S2CDescUserControl[31:0]	{TX_CSCNTRL,14'd0, TX_CSBEGIN}	TX_CSCNTRL controls insertion of the checksum into data frame: • 01: Partial CSUM offload (C_TXCSUM=1) • 10: Full CSUM offload (C_TXCSUM=2) TX_CSBEGIN is the beginning offset which points to the first data byte to be included in checksum calculation
S2CDescUserControl[63:32]	{TX_CSINSERT, TX_CSINIT}	TX_CSINSERT is the offset which points to the location where the checksum should be written into the TCP/UDP segment header. TX_CSINIT is the seed used to insert the pseudo header into the checksum calculation. If the stack inserts the pseudo header checksum into the packet, this field should be zeroed.

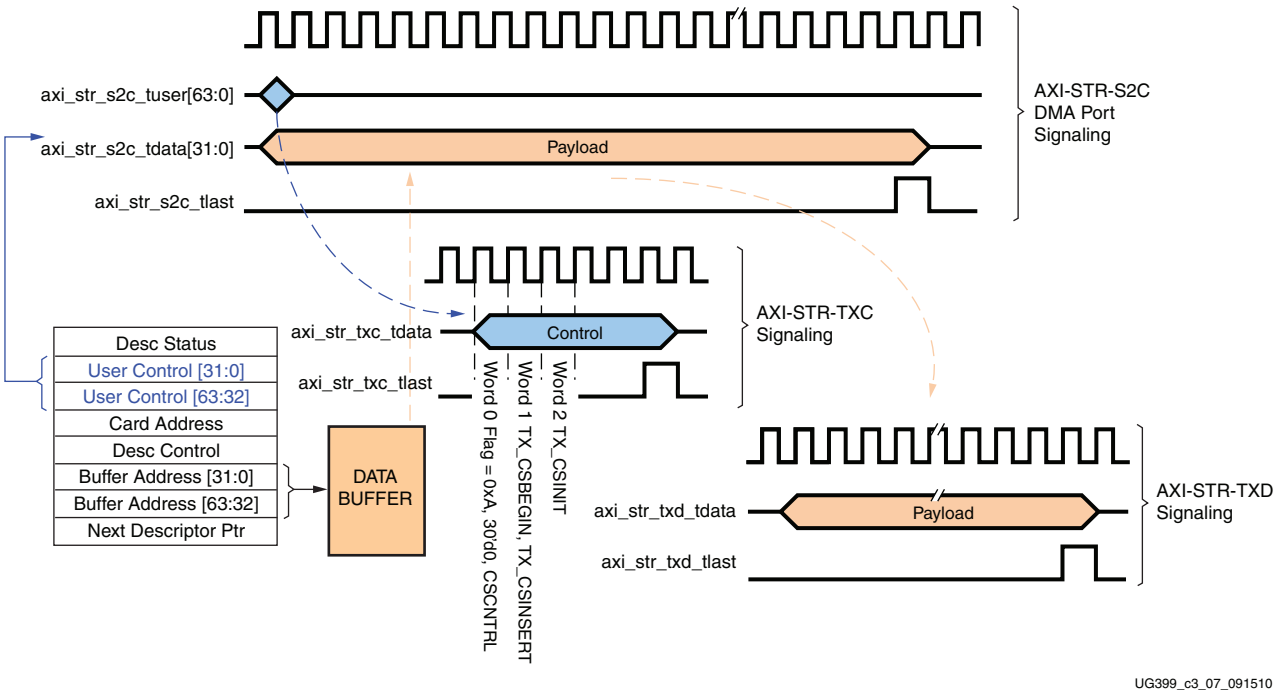


Figure 3-7: **User Control Information (Transmit Direction)**

In the receive direction, the checksum value obtained is transferred to the system through the `axi_str_c2s_tuser` signal over to user status field in the buffer descriptor (See Table 3-5 and Figure 3-8.

Table 3-5: **Field Mappings for Checksum Offload (Receive Direction)**

C2S Descriptor User Status Field	Checksum Field Mapping	Description
C2SDescUserStatus[31:0]	{RX_CSRAW,13'd0, RX_CS_STS}	RX_CSRAW is the raw checksum calculated over the entire Ethernet payload starting from the 14th byte. RX_CS_STS is the receive CSUM status: • 000: Neither IP header nor TCP/UDP checksums checked. • 001: IP header CSUM checked and is correct. TCP/UDP CSUM unchecked. • 010: Both IP header and TCP CSUM checked and correct. • 011: Both IP header and UDP CSUM checked and correct. • 100: Reserved • 101: IP header CSUM incorrect. TCP/UDP CSUM unchecked. • 110: IP header CSUM correct but TCP CSUM incorrect. • 111: IP header CSUM correct but UDP CSUM incorrect.
C2SDescUserStatus[63:32]	{RX_BYTECNT,18'd0}	RX_BYTECNT is the length of the received frame in bytes (the number of bytes in the Ethernet frame).

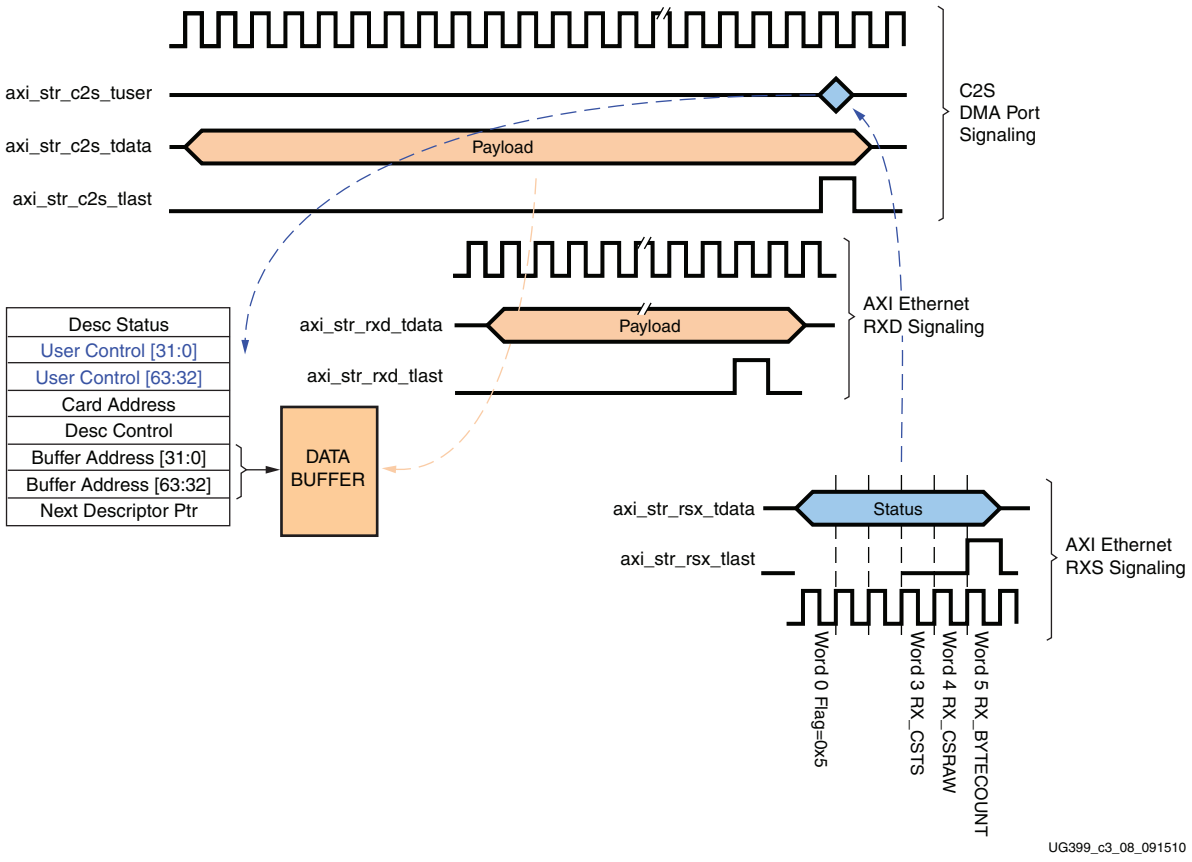


Figure 3-8: **User Status Information (Receive Direction)**

The AXI_Ethernet core supports various options for transmit and receive buffer size. Higher values of the Block RAM size support jumbo frames up to 9 KB in size.

Support for 1000BASE-X Mode

The AXI_Ethernet IP also provides an option to use 1000BASE-X mode by configuring the parameter C_PHY_TYPE = 5. This setting enables and integrates the 1000BASE-X PCS-PMA core within the AXI_Ethernet IP.

The 1000BASE-X IP, implements the PCS and PMA specific functionality as defined in IEEE 802.3 specific to 1000BASE-X operation. The core implements logic for clause 36 and 37 of the specification.

In 1000BASE-X mode, only the serial interface signals are exposed out of the FPGA when compared to multiple GMII signals in GMII mode. The serial interface is routed to the SFP connector on the SP605 board where an optical module (using a fiber optic connection) or SFP to RJ45 converter module (using a copper connection) can be plugged in.

Memory Path Architecture

This section describes the external memory connection through the integrated memory controller block on the Spartan-6 FPGA.

Additional logic designed into the system implements a FIFO layer around the AXI4 interface delivered by the MIG wrapper such that the entire DDR3 SDRAM appears as a virtual FIFO.

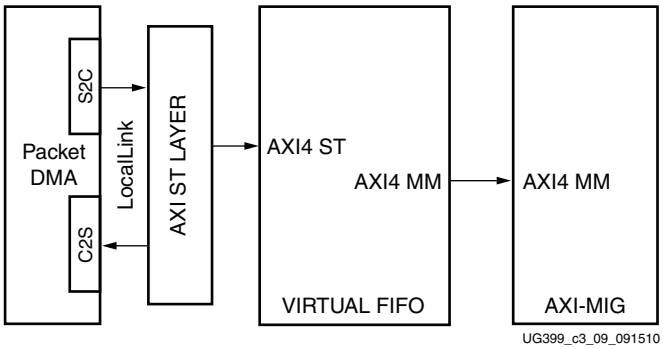


Figure 3-9: **Connection to MIG using AXI**

Memory Interface Generator

The MIG provides a wrapper around the integrated memory controller and physical interface block. The user interface is AXI4-MM. The MIG defines all the attributes required to implement a memory interface. When selecting the memory component as MT41J64M16LA-187E DDR3 (which is the available DDR3 SDRAM component on the SP605 board), the MIG wrapper automatically programs the appropriate memory attributes and timing parameters. This DDR3 SDRAM component is connected to the MCB in bank-3 of the FPGA. Details of the AXI4-MM interface can be obtained from UG388, *Spartan-6 FPGA Memory Controller User Guide*.

MIG Wrapper Modification for Clocking Support

The MIG core generated from the CORE Generator tool requires a 333.33 MHz differential clock input (3000 ps time period). As the SP605 board provides only a 200 MHz differential clock, the following changes are made to the MIG files to facilitate the use of the 200 MHz clock available.

In the file `infrastructure.v`, the following changes are made:

- The input clock period to the PLL is changed to 5 ns.
- The PLL settings are changed to generate a 667 MHz clock and other relevant clocks for the memory controller.

The user constraints file is also modified appropriately for locations and constraints. The MIG core-generated files, changed for this purpose, are provided with the design.

Virtual FIFO

The DDR3 SDRAM requires an address for data access. The Packet DMA provides the AXI4-Stream interface but no destination address information.

To enable a packetized, streaming interface to direct data into memory, a virtual FIFO wrapper is designed around the MIG core provided in the AXI4-MM interface (see

Figure 3-10). The virtual FIFO wrapper makes the DDR3 SDRAM appear as a FIFO and the logic implemented takes care of address generation for read and write transactions. The virtual FIFO requires the packet size to be a multiple of four bytes so that all data in the packet is valid and stored completely in memory.

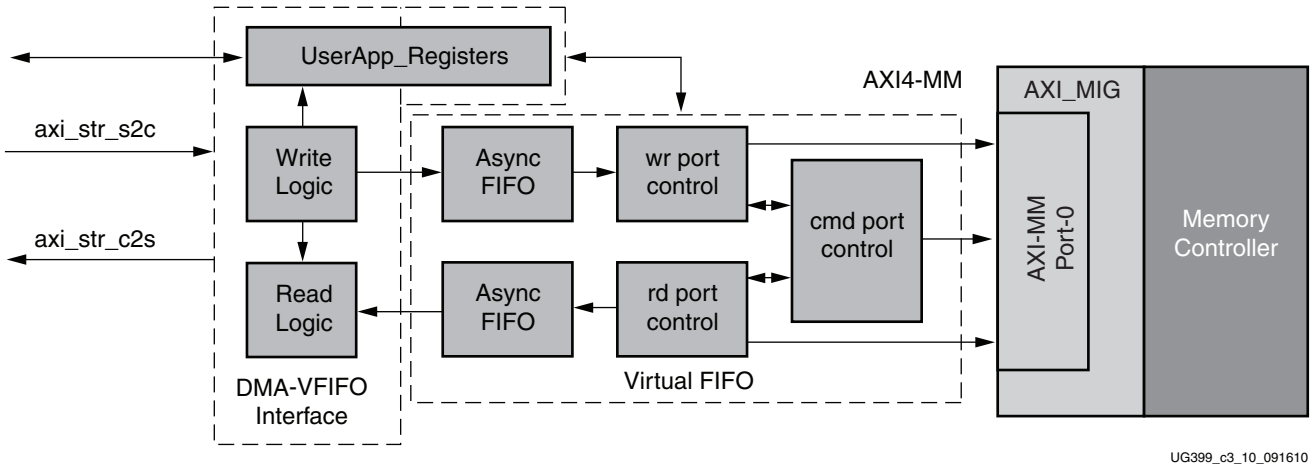


Figure 3-10: **Virtual FIFO Wrapper**

The virtual FIFO design provides an AXI4-Stream FIFO-based user interface. The read and write logic blocks map the DMA-provided AXI4-Stream interface to the FIFO interface. The FIFO interface signals are described in Table 3-6

Table 3-6: **FIFO Interface Signal Description**

Signal Name	Direction	Description
p0_axi_str_wr_tdata[31:0]	Input	Write data input to FIFO
p0_axi_str_wr_tready	Output	Write ready status from FIFO
p0_axi_str_wr_tvalid	Input	Write data valid input to FIFO
p0_axi_str_rd_tdata[31:0]	Output	Read data output from FIFO
p0_axi_str_rd_tvalid	Output	Read data valid output from FIFO
p0_axi_str_rd_tready	Input	Read ready status to FIFO
start_address[31:0]	Input	Start address in DDR3 SDRAM for the FIFO
end_address[31:0]	Input	End address in DDR3 SDRAM for the FIFO
wrburst_size[8:0]	Input	Burst size for write transactions
rdburst_size[8:0]	Input	Burst size for read transactions

Data is considered valid only when both tready and tvalid signals are asserted. The AXI4-Stream FIFO is responsible for managing the data writes to and reads from FIFO, based on these sideband signals. The other signals, tstrb and tlast, are optional and are not used in the design as the entire data in a packet is considered valid.

The asynchronous AXI4-Stream FIFOs also provide a means of crossing over from user clock domain to MIG IP clock domain. This FIFO acts as a preview FIFO which buffers the data so that the appropriate burst length can be driven across the AXI4 interface. For maximum efficiency, a burst size of 256 (which is also the maximum supported by AXI4) is

used. The burst size value can be varied by programming the Write and Read burst size registers (see Appendix A, Register Description).

A brief functional description of the various modules follows:

- User Register Block

  This block manages the registers for start address, end address, write burst size, and read burst size. This interfaces to the register interface of the Packet DMA.

- Write Port Control

  This block buffers incoming data from DMA into the FIFO and provides write transaction size based on the availability of data. When the write transaction is granted, it also reads the data from the FIFO and provides it to the AXI4 interface write data channel.

- Read Port Control

  This block reads data from the AXI4 interface read data channel and stores it into FIFO. The read burst size is dependent on the space available in the preview FIFO. However, the aim is always to be able to issue the maximum size transactions for better performance.

- Command Port Control

  This block generates the addresses required for writing/reading the data to/from the DDR3 SDRAM memory. It also generates the control signals such as write burst size and read burst size, etc., on the AXI4 interface of the AXI protocol based wrapper. This block is responsible for address generation for read and write and also tracking FIFO full or empty conditions. The FIFO in external memory is implemented between the start and end address ranges as programmed by the software (see Appendix A, Register Description). This module also handles the address wrap-around for FIFO.

The memory path design provided uses one 32-bit bidirectional port on the memory controller block. The design is scalable so multiple instances of the virtual FIFO logic can be used for up to four bidirectional memory controller ports (four is the maximum number allowed). To generate the MIG IP with four 32-bit bidirectional ports, the MCB_PORTS and NUM_PORTS parameters in the top level file should be modified to 4. The connections are currently implemented to daisy chain the virtual FIFO, that is, data read from one virtual FIFO is provided as write data to the next virtual FIFO. The user can modify the daisy chain logic for a custom design, as required. For future scalability considerations, control registers, when using multiple ports of the controller, are implemented with relevant placeholders.

Common Registers

A set of common registers are defined and shown in Figure 3-1. These registers include the following:

PCI Express Transaction Monitor Registers

The PCIe transaction monitor registers provide transaction interface utilization for PCI Express. A transaction layer packet utilization monitor is designed to help analyze the transaction layer utilization. This monitor provides registers to count the following:

- Double words transmitted upstream, including packet headers
- Double words received downstream, including packet headers
- Transmitted memory write transactions payload count

- Received completion transactions payload count

Details of these register bits are found in PCI Express Transaction Layer Monitor Registers in Appendix A.

Clocking and Reset

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

Clocking Strategy

An overview of the clocking strategy is shown in Figure 3-11. Apart from various clock requirements of the cores, the FPGA logic in the design runs at a uniform clock rate of 62.5 MHz.

The Endpoint for PCI Express receives a 100 MHz differential clock from host system. This is converted to 125 MHz by an on-board PLL and provided to the FPGA. The Endpoint for PCI Express outputs a 62.5 MHz clock. DMA, FPGA logic, and data and control interfaces of the network and memory paths operate at the same speed.

Additionally, the AXI_Ethernet IP requires a 125 MHz clock for driving GMII signals and a 200 MHz reference clock for clocking the IODELAY controllers for GMII signals. These clocks are derived from the 200 MHz differential clock available on the SP605 board through a DCM. The AXI4-Stream datapath and AXI4 slave interface of the AXI_Ethernet IP runs at the 62.5 MHz clock provided by Endpoint for PCI Express.

The differential clock required by the MCB and the DDR3 SDRAM is also derived from the 200 MHz differential clock source available onboard.

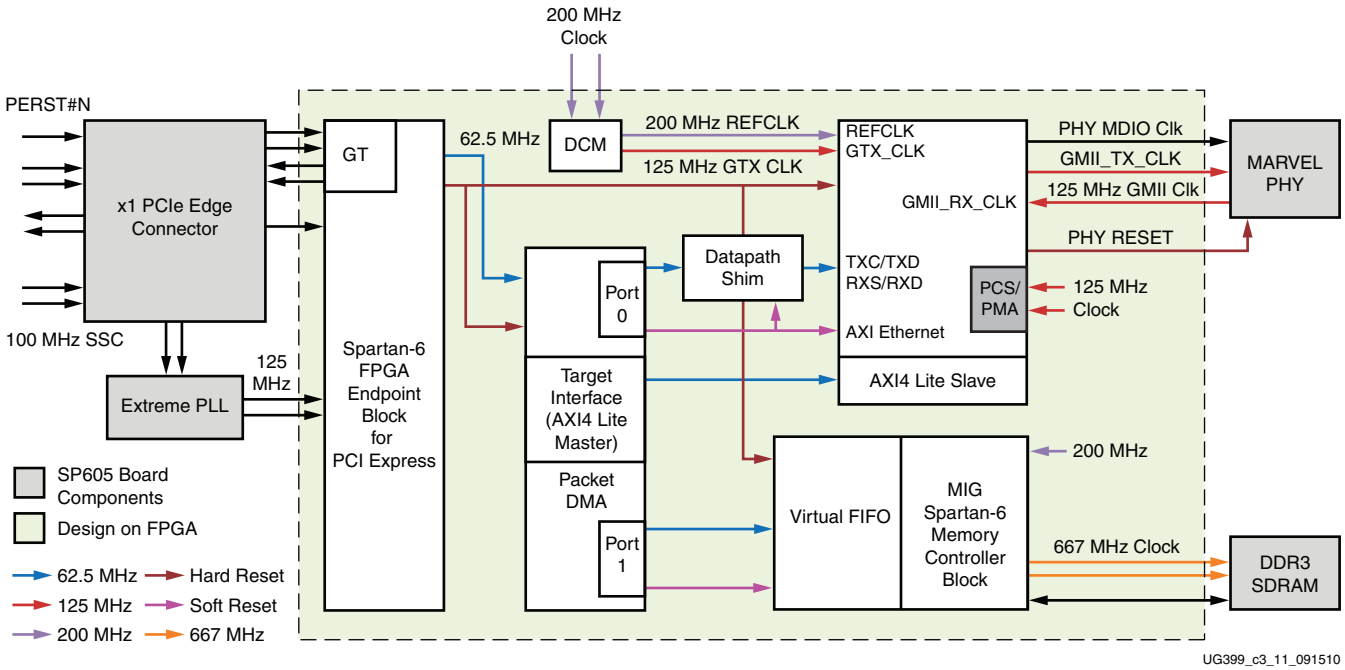


Figure 3-11: **Clocking and Reset Strategy**

For the 1000BASE-X mode, a 125 MHz differential clock is provided to the AXI_Ethernet core. All other clock connections remain the same as in the GMII mode.

Reset Scheme

This section outlines the reset scheme of the design. An overview of the reset connection is shown in the Figure 3-11.

The system reset for PCI Express (PERST#N) driven by the downstream port through the edge connector is the only hard reset available to the entire design. This reset is provided as an output by the Endpoint core which is connected to the various blocks used.

In addition, a software driver programmed reset, also called a soft reset, is provided by the DMA. This reset is per DMA channel and used to drive the user logic blocks connected to the specific port being reset. The soft reset feature is used when the software driver is loading and unloading to flush out all pending transactions and prevent any stale transactions from interrupting the CPU when the hardware is not in use.

Software Design Description

The software component of the TRD framework is comprised of one or more Linux/Windows kernel-space driver modules with one user-space application, which controls the design operation.

> ***Note:*** For details on available user APIs, refer to the documentation supplied with the driver sources.

The software is designed using building blocks for scalability. Additional user-space applications can be designed with the existing blocks.

The software is designed to meet the following requirements:

- Generate adequate data to showcase the high-performance capabilities of the hardware design with specific focus on throughput on the PCI Express and Ethernet links.
  - Hardware design must demonstrate the use of PCI Express compliant DMA, 1 Gb/s Ethernet, and DDR3, while achieving best possible data throughputs in the process.
- Effectively showcase the use of the multichannel DMA transfers
- Provide an easy to use and intuitive user interface
- Provide an extensible, reusable, and customizable modular design.

The feature list of the User Application and Linux kernel-space drivers that enables these requirements is described in User-Space Application Features.

User-Space Application Features

The user-space application (which is a graphical user interface or 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 PCI Express transaction interface and DMA engine

Standard Linux/Windows OS tools should be used as described in Ethernet Specific Features in Chapter 2 for control of Ethernet specific features.

Kernel-Space Driver Features

The kernel-space application provides these features:

- Configuration of the DMA engine to achieve data transfer between the hardware and main system memory
- Transfer of Ethernet packets from TCP/IP stack to the network path in hardware for transmission into the LAN, and from the network path in hardware to the TCP/IP stack for handling by networking applications. This is the Ethernet data flow.
- Transfer of a block data stream from the driver to the memory path in hardware for storing in DDR3, and loopback into system. This is the memory data flow.

Data Flow Model

This section provides an overview of the datapath flows in both software and hardware.

Ethernet Data Flow

Figure 3-12 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 the packet DMA. The DMA fetches the packet through the Endpoint for PCI Express and transfers it to the AXI_Ethernet where it is transmitted through the Ethernet link into the LAN.

On the receive side, packets arriving on the AXI_Ethernet are collected by the packet DMA. The DMA pushes the packet to the driver through the Endpoint for PCI Express. The driver hands off the packet to the upper layers for further processing.

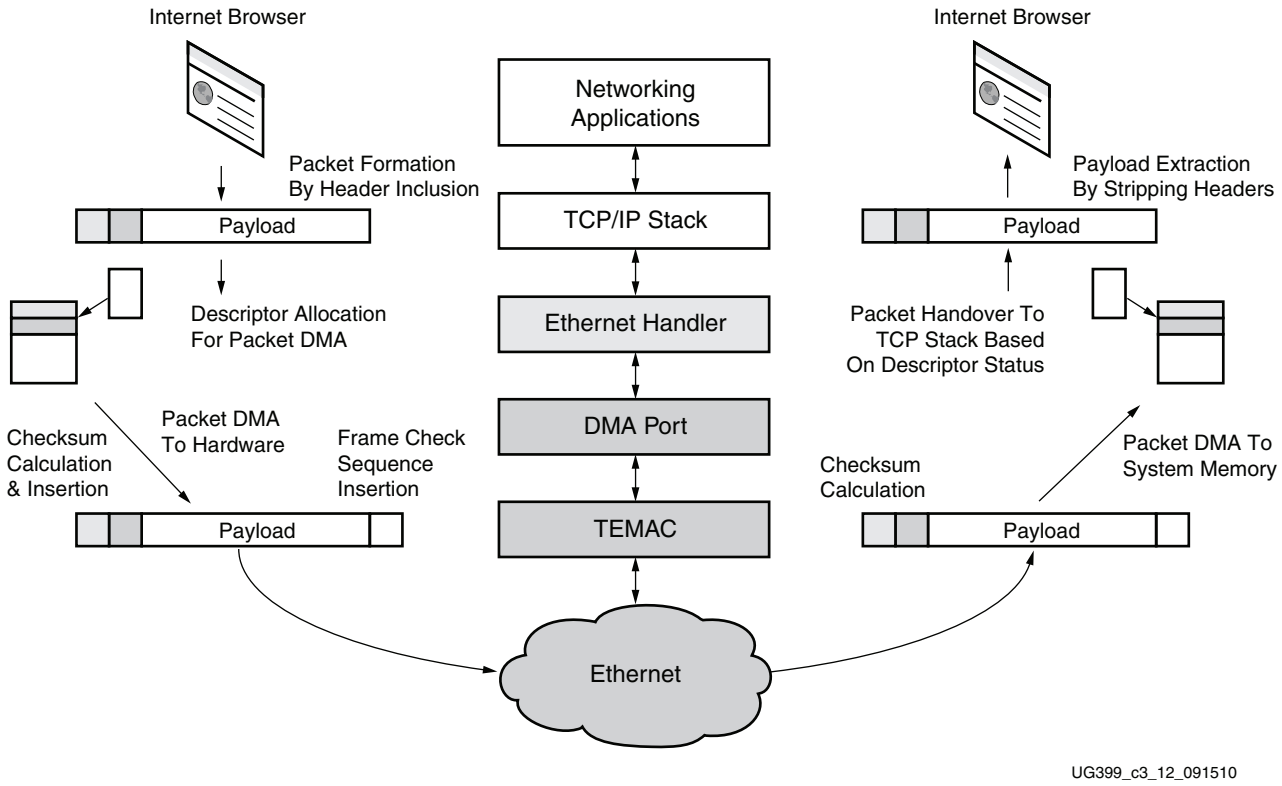


Figure 3-12: **Ethernet Data Flow**

In a typical use scenario, the user runs an application, such as a web browser, and packets are transmitted to and received from the network.

Memory Data Flow

Figure 3-13 illustrates the memory data flow. A streaming block data flow is implemented on the DDR3 side.

On the transmit side, data buffers are generated in the block data handler and queued up for transmission. The packet DMA fetches the packets through the Endpoint for PCI Express and transfers them to the Virtual FIFO which writes them into the DDR3 memory. The data written to the DDR3 is read and transferred back to the DMA creating a loopback scenario. On the receive side, the DMA pushes the packets to the software driver through the Endpoint for PCI Express. The driver receives the packets in its data buffers, verifies the data, and discards the buffers.

The driver also reads the performance statistics collected by the hardware, and makes these available to the `xpmon` GUI.

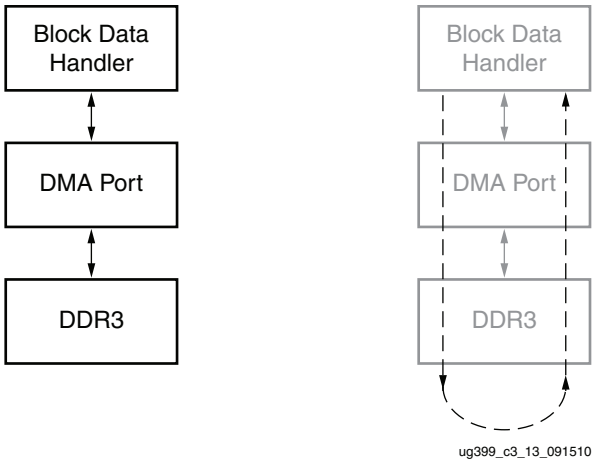


Figure 3-13: **DDR3 Memory Data Flow**

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 architecture has two basic data flow paths. One is the Ethernet flow and the other is the streaming block data flow. The Ethernet flow carries typical networking packets into and out of the system through a DMA port, destined for the AXI_Ethernet. The block data flow carries blocks of raw data into and out of the system through another DMA port, destined for the DDR3.

Figure 3-14 shows the architecture overview. The standard software blocks already exist and the driver designed interacts with them. No modifications are done to the standard blocks; they are used as is.

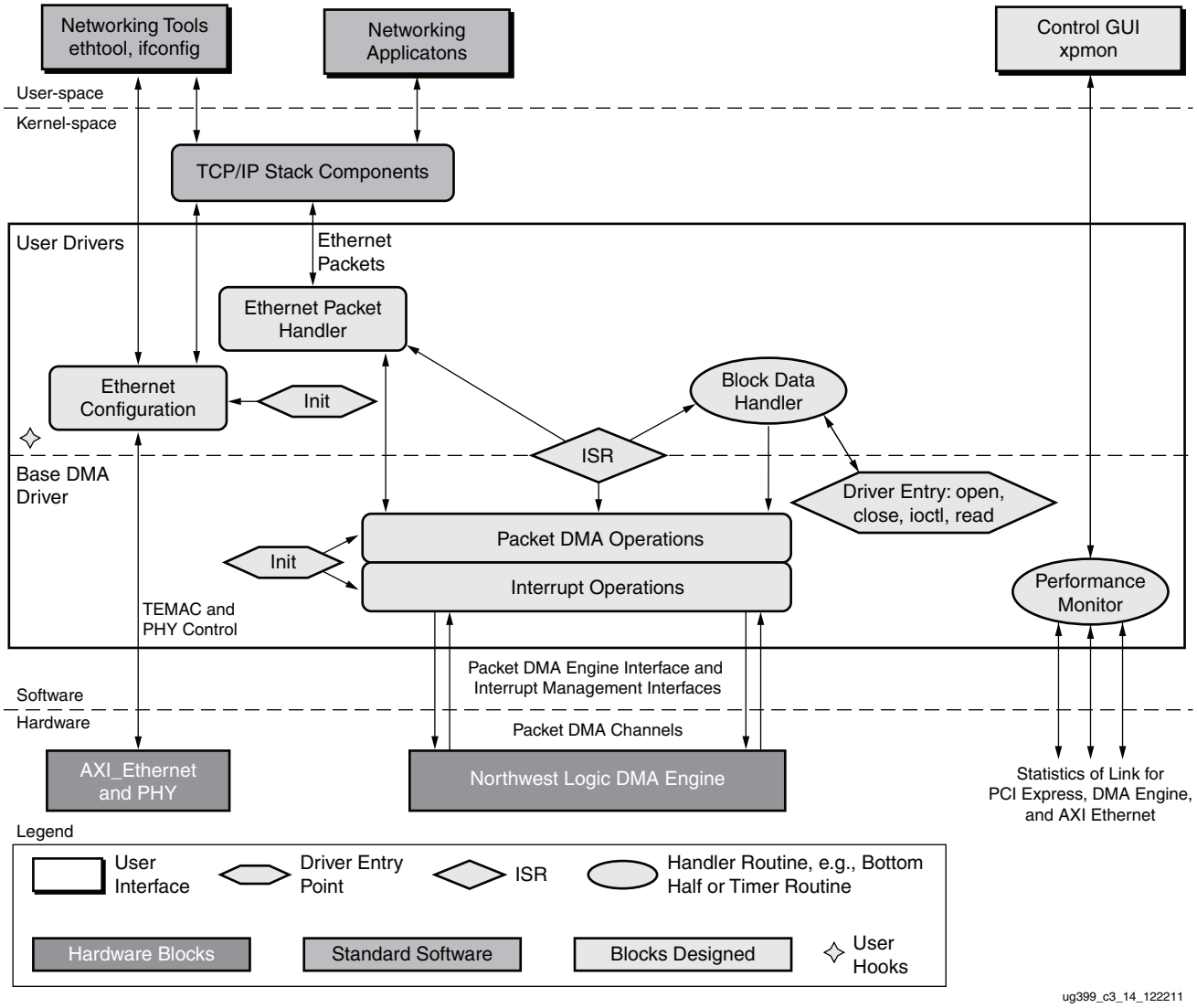


Figure 3-14: **Software Architecture Overview**

Applications

Control and Monitor GUI (xpmon)

A graphical user interface tool (xpmon) monitors device status, runs performance tests, and displays statistics. It conveys the user-configured test parameters to the block data handler in the driver, which then starts an appropriate test. Performance statistics gathered during the test are periodically conveyed to the GUI, where they are displayed in one or more graphs.

Networking Tools

Unlike the block 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.

Networking Applications

Standard networking applications such as web browser, telnet, or Netperf can be used to initiate traffic in the Ethernet flow. The driver fits under the TCP/IP stack software, using the standard hooks provided.

Kernel Components

Driver Entry Points

The driver has several entry points, a few are described in this section. The system invokes the driver entry function when a hardware match is detected after driver insertion (when the PCIe device probed by the driver is found). After reading the device's configuration space, various initialization actions are performed; initialization of the DMA engine(s), AXI_Ethernet, and PHY; setting up of the receive and transmit buffer descriptor rings; and initialization of interrupts.

The other driver entry points are mainly used in the block data flow: when the GUI starts up and shuts down; when a new performance test is started or stopped; and to convey periodic performance statistics results to the GUI.

Some driver entry points are specific to Ethernet configuration, and these are invoked by the system if the user attempts to change any configuration using standard tools.

DMA Operations

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

Ethernet Packet Handler

The driver is informed when a packet is queued for transmission by the upper layer. The driver adds the packet buffer to the transmit descriptor ring and submits it for DMA. Subsequently, when the driver is informed that the packet has been successfully transmitted, it removes the packet buffer from the descriptor ring, frees it, and clears the descriptor for future use.

On the receive side, the full-receive descriptor ring is submitted to the DMA engine, ready to accommodate packets on arrival. Subsequently, when the driver is informed that a packet has arrived, it removes the packet buffer from the descriptor ring, passes it to the upper layer, and clears the descriptor for future use.

Block Data Handler

The data payload for the block data flow is generated and consumed in the block data handler. When a test is started, data buffers of random or fixed sizes are generated based on user selection and then queued for the transmit DMA. The hardware design loops this data back through the DDR3 and the data buffers arrive in the system as receive DMA. The handler discards the data and returns the buffer to a free pool for future use.

Interrupt Service Routine

The interrupt service routine (ISR) manages interrupts from the DMA engine and other errors from hardware (if any).

The driver sets up the DMA engine to interrupt after every N descriptors that it processes. This value of N can be set by a compile-time macro. The ISR invokes the functionality in the block and Ethernet handler routines pertaining to handling received data, and housekeeping of completed transmit and receive descriptors.

Performance Monitor

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

TCP/IP Stack

The TCP/IP stack code also resides in the operating system kernel. It has well-defined hooks for the Ethernet driver to attach to, allowing configuration and communication of all standard networking tools and applications with the driver.

User Hooks

The design and code is developed to allow modification, using compile-time variables and through adaptable APIs to different applications.

DMA Descriptor Management

This section describes the descriptor management portion of the 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. Packets can fit in a single descriptor or can span across multiple descriptors. Also, the receive size of the actual packet can be smaller than the original buffer provided to accommodate the packet.

For proper descriptor management:

- The software and hardware must be able to independently work on a set of buffer descriptors in a supplier-consumer model
- The software must be informed of packets being received and transmitted as data flow happens. On the receive side, the software needs to know the size of the actual received packet

The rest of this section describes the driver design using the features provided by third party packet DMA IP to achieve the design objectives.

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

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.

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

Table 3-7: **Terminology Summary**

Term	Description
HW_Completed	The register containing the address of the last descriptor that the DMA has completed processing. This corresponds to the `Reg_Completed_Desc_Ptr` register in DMA.
HW_Next	The register containing the address of the next descriptor that the DMA processes. This corresponds to the `Reg_Next_Desc_Ptr` register in DMA.
SW_Next	The register containing the address of the next descriptor that software submits for DMA processing. This corresponds to the `Reg_SW_Desc_Ptr` register in the DMA.

Initially, the driver prepares descriptor rings for each DMA channel. In the reference design, the driver prepares four rings.

Transmit Initialization Phase

1. Driver initializes the *HW_Next* and *SW_Next* registers to start of ring
2. Driver resets the *HW_Completed* register
3. Driver initializes and enables the DMA engine

Transmit (S2C) Descriptor Management

In Figure 3-15, the darkened blocks indicate the descriptors under hardware control and the white blocks indicate the descriptors under software control.

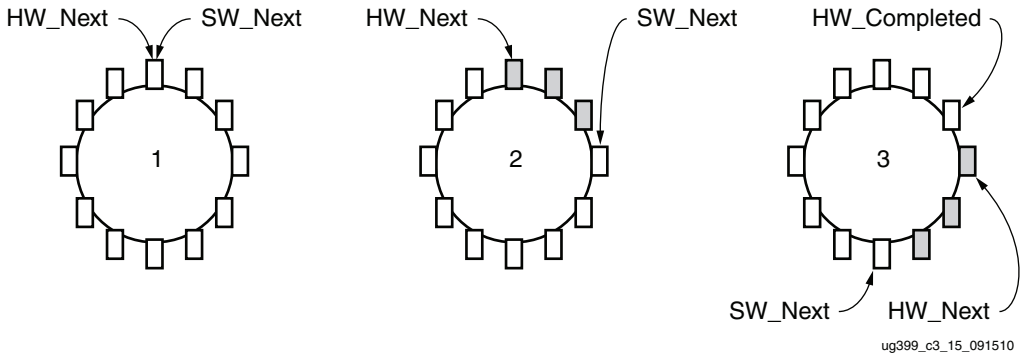


Figure 3-15: **Transmit Descriptor Ring Management**

Packet Transmission

1. The packet arrives in the Ethernet packet handler
2. The packet is attached to one or more descriptors in the ring
3. The driver marks the SOP, EOP, and IRQ_on_completion in the descriptors
4. The driver adds any user-control information (for example, checksum-related) to the descriptors
5. The driver updates the SW_Next register

Post-Processing

1. The driver checks for completion status in the descriptor

2. The driver frees the packet buffer

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

Receive (C2S) Descriptor Management

In Figure 3-16, the darkened blocks indicate the descriptors under hardware control and the white blocks indicate the descriptors under software control.

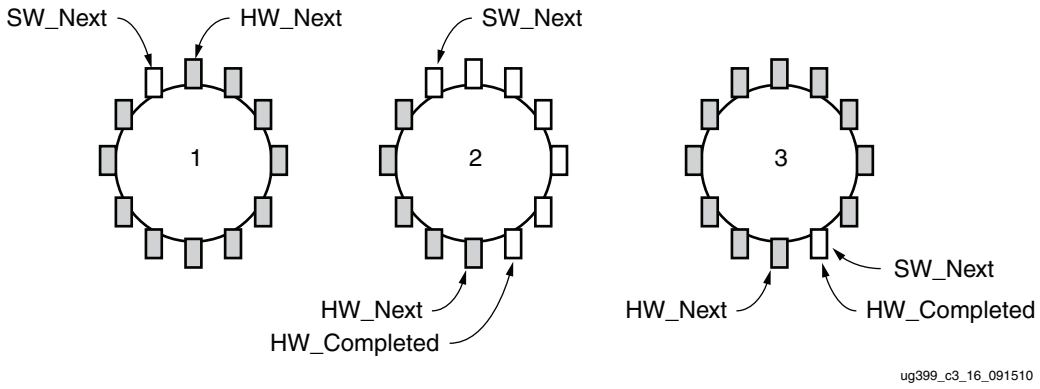


Figure 3-16: **Receive Descriptor Ring Management**

Receive Initialization Phase

1. The driver initializes the receive descriptor with an appropriate Ethernet or block data buffer.
2. The driver initializes the HW_Next register to the start of the ring and the SW_Next register to the end of the ring
3. The driver resets the HW_Completed register
4. The driver initializes and enables the DMA engine

Post-Processing after Packet Reception

1. The driver checks for completion status in the descriptor
2. The driver checks for SOP, EOP, and user status information
3. The driver forwards the completed packet buffer(s) to the upper layer
4. The driver allocates the new packet buffer for the descriptor
5. The driver updates the SW_Next register

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

User Interface—Control and Monitor GUI

The control and monitor GUI is the main application tool for the configuration, status, and statistics display. When installed, the driver appears as a device table entry. The GUI uses the typical file-handling functions (open, close, read, write, ioctl) on this device to

communicate with the driver. These calls result in the invocation of the appropriate driver entry points. The I/O control function *ioctl()* is used as a driver entry point by the application GUI.

Figure 3-17 shows a screen capture of the GUI status. An explanation of the various fields referenced by numbers follows.

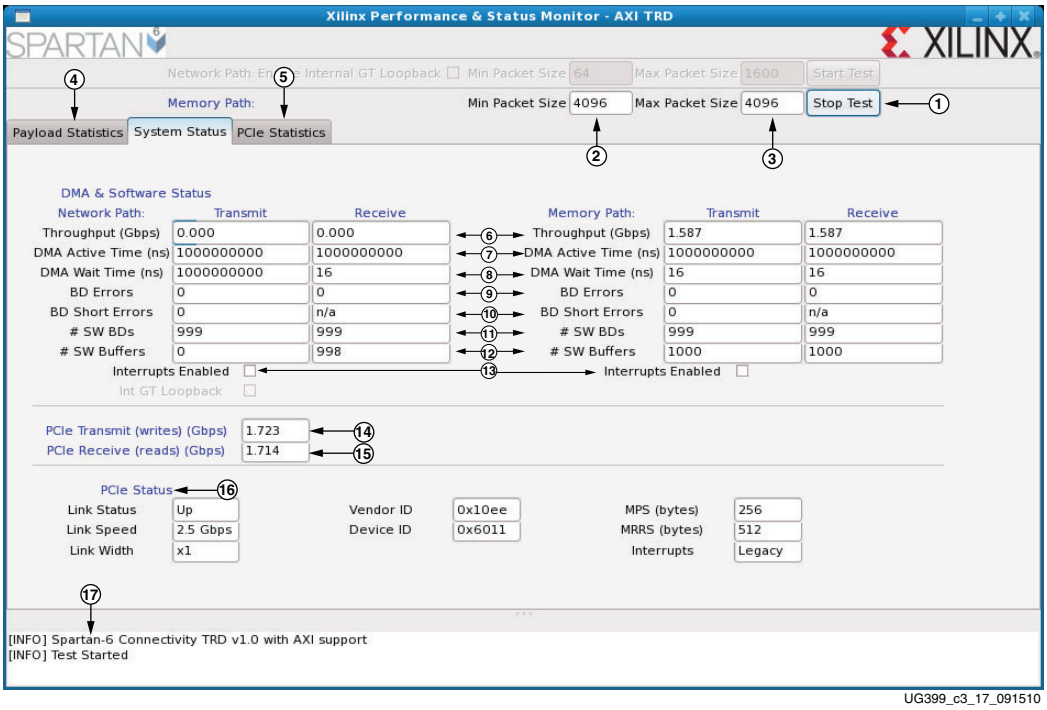


Figure 3-17: **Software Application Screen Capture**

1. Test start/stop control for memory application
2. Min. Packet Size: Minimum packet size selection in bytes
3. Max. Packet Size: Maximum packet size selection in bytes
4. Payload Statistics: Shows the payload statistics graphs based on DMA engine performance monitor
5. PCIe statistics: Plots the PCIe transaction interface utilization
6. Throughput: DMA payload throughput in Gb/s for each engine.
7. DMA Active Time: The time (in nanosecond) the DMA engine has been active in the last one second.
8. DMA Wait Time: The time (in nanosecond) the DMA was waiting for the software to provide more descriptors.
9. BD Errors: Indicates a count of descriptors which caused a DMA error – indicated by the error status field in descriptor update
10. BD Short Errors: Indicates 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 receive direction.
11. # SW BDs: Indicates count of total descriptors set up in the descriptor ring
12. # SW Buffers: Indicates count of total data buffers associated with the ring

13. Interrupts Enabled: Indicates interrupt enable status for that DMA engine
14. PCIe Transmit: Reports the transmit TLP utilization as obtained from the transaction monitor in hardware
15. PCIe Receive: Reports the receive TLP utilization as obtained from the transaction monitor in hardware
16. PCIe Status: Reports the status of various PCIe fields as reported in the endpoint's configuration space
17. Text pane at the bottom shows up informational messages, warnings or errors.

The GUI has individual tabs for the following:

Status

- Link status for PCI Express
- DMA Engine status

The driver always maintains information on the status of the hardware. The GUI invokes *ioctl()* to read this status information and updates it every few seconds.

Statistics

- Link statistics for PCI Express provided by hardware
- Graphic display of all statistics

The driver maintains a set of arrays to hold per-second sampling points of these statistics. These statistics are periodically collected by the performance monitor handler. The arrays are managed in a circular fashion. The GUI periodically invokes an *ioctl()* to read these statistics and then displays them.

A separate test button is provided to run user tests. The test button is applicable only to the memory path in the design as the traffic for Ethernet is generated by upper layers (that is, TCP/IP) and by standard applications.

Test

- Test setup
- Start/Stop of test

When the user starts a test, the GUI informs the driver the parameters of the test:

- Minimum and maximum packet size. If these are different, the driver generates packets of random sizes within these bounds.

The driver entry point sets up the test parameters and informs the block data handler, which then starts setting up the block data buffers for transmission, reception, or both. Similarly, if the user were to abort a test, the GUI informs the driver, which sets up the abort mechanism. The test is aborted by stopping the transmit side flow and then allowing the receive side flow to drain.

The GUI programming environment is GTK+.

XILINX®

Chapter 4

Performance Estimation

This chapter presents a theoretical estimation of performance on the interface for PCI Express, Ethernet interface, and the memory interface. It also presents a method to measure performance.

PCI Express Performance

PCI Express is a serialized, high bandwidth and scalable point-to-point protocol that provides highly reliable data transfer operations. The maximum transfer rate of a protocol specification v1.1 compliant core is 2.5 Gb/s. This rate is the raw bit rate per-lane per-direction and not the actual data-transfer rate. The effective data-transfer rate is lower due to protocol overheads and other system design trade-offs.

The link performance of PCI Express together with packet DMA is estimated under the following assumptions:

- Each buffer descriptor points to a 1 KB data buffer space
- Maximum Payload Size (MPS) = 128B
- 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

Performance is projected by estimating the overheads and then calculating the effective throughput by deducting these overheads. Descriptor fetch/update as required for data movement through DMA is also considered as an overhead.

Independent calculations are made for each direction of a C2S or a S2C DMA engine.

The notes in Table 4-1 describe the abbreviated conventions.

Table 4-1: **PCI Express Performance Estimation with DMA**

Transaction Overhead	ACK Overhead	Comment
MRD: C2S Descriptor = 20/1024 = 2.5/128	8/1024 = 1/128	One descriptor fetch in a C2S engine for 1 KB data; 20B of TLP overhead and 8 bytes of DLLP overhead
CPLD: C2S Descriptor = (20+32)/1024 = 6.5/128	8/1024 = 1/128	Descriptor reception C2S engine
MWR: C2S Descriptor = (20+12)/1024 = 4/128	8/1024 = 1/128	Descriptor update C2S engine
MWR: C2S Buffer = 20/128	8/128	MPS = 128B; Buffer write C2S engine
MRD: S2C Descriptor = 20/1024 = 2.5/128	8/1024 = 1/128	Descriptor fetch in S2C engine
CPLD: S2C Descriptor = (20+32)/1024 = 6.5/128	8/1024 = 1/128	Descriptor reception S2C engine
MWR: S2C Descriptor = (20+4)/1024 = 3/128	8/1024 = 1/128	Descriptor update S2C engine
MRD: S2C Buffer = 20/128	8/128	MRRS = 128B; Buffer fetch S2C engine
CPLD: S2C Buffer = 20/64 = 40/128	8/64 = 16/128	RCB = 64B; Buffer reception S2C engine

Notes:

1. Nomenclature for table: Memory Read transaction (MRD); Memory Write transaction (MWR); Completion with data (CPLD); Card-to-system (C2S) for receive direction DMA; System-to-card (S2C) for transmit direction DMA.

Once all overheads for each DMA engine are estimated, effective throughput is calculated. The throughput calculation for application data (packets available at the DMA streaming interface) for a x1 link is tabulated in Table 4-2.

Table 4-2: **PCI Express Throughput Estimate**

Direction	Overhead	Effective Throughput (Gb/s)
PCIe transmit (C2S only)	$100 \times 27.5/(128 + 27.5) = 17.68\%$	1.64
PCIe receive (C2S only)	$100 \times 16.5/(128 + 16.5) = 11.41\%$	1.77
PCIe transmit (S2C only)	$100 \times 42.5/(128 + 42.5) = 24.9\%$	1.5
PCIe receive (S2C only)	$100 \times 56.5/(128 + 56.5) = 30.6\%$	1.38

The S2C engine (data transmission, that is, reading data from system memory) issues read requests and receives data through completions. This engine exercises data (actual frame) traffic on the PCIe receive link giving a performance of ~1.38 Gb/s. This is the PCIe memory read performance.

The C2S engine (data reception, that is, writing data to system memory) issues write requests. This engine exercises data (actual frame) traffic on the PCIe transmit link giving a performance ~1.64 Gb/s. This is PCIe memory write performance.

PCIe receive on C2S implies a descriptor fetch and ACK-NAK, resulting in a low overhead. PCIe transmit on S2C is made up of descriptor and buffer fetch requests. Read requests do not contribute towards data throughput as they are only headers.

Ethernet Performance

The raw line rate of the Ethernet link is 1.25 Gb/s, which is commonly referred to as gigabit Ethernet after accounting for 8B/10B encoding overheads.

The performance, as seen by various Ethernet applications at different layers, is lesser than the throughput seen at the driver and the Ethernet interface. This is due to the various headers and trailers inserted in each packet by all the layers of the networking stack. Ethernet is used as a medium to carry traffic and various protocols, including TCP/UDP, to implement protocol specific header/trailer formats.

Consider transmission control protocol (TCP) as an example. The protocol header includes the following:

- TCP/IP Overhead: 20 bytes TCP header + 20 bytes IP header
- Ethernet Overhead: 14 bytes Ethernet header + 4 bytes trailer

Based on this overhead, the theoretical TCP throughput is as shown in Equation 4-1, where D is the application message size in bytes.

$$TCP\ throughput = \left(\frac{D}{D + 40 + 18}\right) \times 1000\ Mb/s \qquad \text{Equation 4-1}$$

More precisely, for application message sizes greater than 1460 bytes, the formula is shown in Equation 4-2, where M is the MTU size configured on the system.

$$TCP\ throughput = \left(\frac{D}{D + 40}\right) \times \left(\frac{M}{(M + 14)}\right) \times 1000\ Mb/s \qquad \text{Equation 4-2}$$

The calculation of theoretical throughput is tabulated in Table 4-3. The TCP/IP protocol overhead has a significant impact when the send message sizes are smaller than ~1200 bytes.

Table 4-3: **Theoretical Throughput Estimate for TCP**

Message Size in Bytes	Effectiveness (%)	Theoretical Throughput (Mb/s)
64	52.46	524.59
128	68.82	688.17
256	81.53	815.29
512	89.82	898.25
1024	94.64	946.40
1442	96.15	961.47
2048	96.92	969.21
4096	97.86	978.59

Memory Controller Performance

The Spartan-6 FPGA memory controller block, as used in this design, has a total of 16 I/Os interfacing to external DDR3 memory.

Theoretical Calculation

$$Maximum\ I/O\ Rate\ (double\ data\ rate) = 333.5\ MHz \times 2 = 667\ Mb/s \quad \text{Equation 4-3}$$

$$Maximum\ Bandwidth = (Maximum\ I/O\ rate) \times (Number\ of\ I/Os) \quad \text{Equation 4-4}$$

$$= 667\ Mb/s \times 16 = 10.627\ Gb/s \quad \text{Equation 4-5}$$

Equation 4-4 calculates the theoretical maximum bandwidth of the memory controller. An estimate of memory controller performance is as calculated in Equation 4-9:

With larger burst lengths, high efficiency is achievable. With a 32-bit port using a burst length of 32, a total of 1024 bits are transferred.

Number of bits transferred per cycle is:

$$16(bit\ width) \times 2(double\ data\ rate) = 32\ bits/\text{cycle} \quad \text{Equation 4-6}$$

Total cycles used for 1024 bits:

$$1024/32 = 32\ cycles/transfer \quad \text{Equation 4-7}$$

Assuming 10 cycles read to write overhead:

$$32/42 = 76\%\ efficiency \quad \text{Equation 4-8}$$

Assuming 5% efficiency overhead for refresh:

$$71\%\ efficiency\ at\ 667\ Mb/s\ \text{for 16-bit DDR3} = 7577\ Mb/s = 7.577\ Gb/s \quad \text{Equation 4-9}$$

The final estimated bandwidth available to the Virtual FIFO is 7.577 Gb/s.

In the current design, with x1 PCIe at 2.5 Gb/s line rate, the theoretical maximum rate at which the virtual FIFO can read and write to memory controller is calculated as 32 bits × 62.5 MHz = 2 Gb/s. The average data throughput provided by the PCIe and DMA is ~1.6 Gb/s in each direction.

Measuring Performance

This section describes methods to measure performance and presents an analysis of what to expect when different parameters are varied.

PCI Express performance is dependent on factors like maximum payload size, maximum read request size, and read completion boundary which depend on the systems chosen. With higher MPS values, performance improves as packet size increases.

Table 4-4 lists the registers provided by the hardware to aid in the software 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 Upstream TLP Utilization	This register counts upstream TLP traffic including TLP headers for all transactions.

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

Register	Description
PCIe Downstream TLP Utilization	This register counts downstream TLP traffic including TLP headers for all transactions.
PCIe Upstream TLP Payload	This register counts payload for memory write transactions upstream, which includes buffer write and descriptor updates.
PCIe Downstream TLP Payload	This register counts payload for completion transactions downstream, which includes descriptor or data buffer fetch completions.

These registers are updated once every second by hardware. Software reads them periodically at a one second interval. The value read directly gives the throughput in bytes per second.

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

These registers only estimate hardware performance. A software application is required to measure application performance of both the hardware and software impact on overall throughput.

Ethernet Performance Measurement

Ethernet performance can be measured in a private LAN environment using a standard benchmarking tool like Netperf.

Netperf 2.4 can be used as the testbench for measuring outbound and inbound throughput. This is a data transfer application running on top of the TCP/IP stack. The client can be configured for different message sizes and to open a TCP connection or a UDP connection. The two systems are connected in a private LAN connection, which avoids other external LAN traffic. To measure the CPU utilization as accurately as possible, no other applications (other than the standard ones) should be run during the test.

Throughput Estimate and Analysis

It is possible, based on the theoretical estimates to obtain a throughput of up to 935 Mb/s in both inbound and outbound directions.

For jumbo frames, a higher performance is expected. Frames greater than 1514 bytes are categorized as jumbo frames. The AXI_Ethernet IP used in this TRD supports jumbo frames up to 9K bytes.

Higher performance is expected with an increase in MTU size. A higher value of MTU implies that a packet has more payload and less header content.

CPU Utilization Analysis

Improved CPU utilization for larger packets is expected with checksum offload enabled. By offloading an expensive operation to hardware, such as a checksum calculation, the CPU time can be efficiently utilized elsewhere. With small packets, little improvement can be seen in CPU utilization with checksum offload as checksum computation does not contribute greatly to CPU utilization overhead, as does the protocol overhead of using small packets.

Refer to Appendix D, Setting Up a Private LAN for steps on setting up a private LAN connection.

Memory Path Performance Measurement

The performance on the memory path is provided by the DMA performance registers. Variation in performance can be seen with change in packet size. Change of AXI4 protocol write and read burst size through register programming also impacts performance.

XILINX®

Chapter 5

Designing with the TRD Platform

The Targeted Reference Design (TRD) platform is intended to be a framework for system designers to derive extensions or modify designs.

This chapter outlines various ways for a designers to evaluate, modify and re-run the TRD for the connectivity platform.

The suggested modifications are grouped under these categories:

- Software-only modifications: Only changes required to the software driver are covered. The same bitstream provided with the TRD works. Only driver recompilation is required.
- Design (top-level only) modifications: Changes to parameters in the top-level file of the design (`design/source/s6_pcie_dma_ddr3_gbe_axi.v`). No other files require modifications. The design must be re-implemented. See Implementing the Design, page 52 for instructions on design implementation. Depending on the modification, the software driver might require changes.
- Design Changes: This modification shows the plug-n-play feature of the TRD platform. The design must be re-implemented after making the changes. The process requires licenses for additional IP cores being used (if any). The software driver must be modified accordingly.

Any change to the software driver or macros in Makefiles require re-compilation and re-building of the kernel objects. All paths for various files mentioned in this chapter are under the `s6_pcie_dma_ddr3_gbe_axi` directory.

While describing the modifications, each section also describes the implication of the corresponding modification on the overall functionality or performance.

Software-Only Modifications

This section describes modifications to the platform done directly in the software driver. The same hardware design (bitstream) works. Instructions for Windows driver recompilation are provided in Appendix F.

Macro-Based Modifications

This section describes the modifications that result when 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 setup in the descriptor ring are defined as a compile-time option.

Modify the DMA_BD_CNT in `linux_driver/xdma/xdma_base.c` or `windows_driver/xdma/xdma_private.h` macro to change the size of the buffer descriptor ring used for DMA operations. Smaller rings can adversely affect throughput, which is 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.

Log Verbosity Level

Log verbosity level can be controlled by:

- Adding DEBUG_VERBOSE in the Makefiles for Linux, sources file for Windows under each driver directory, which causes the drivers to generate verbose logs.
- Adding DEBUG_NORMAL in the Makefiles for Linux, sources file for Windows under each driver directory, which causes the drivers to generate informational logs.
- Removing both these macros from the Makefiles for Linux, sources file for Windows under each driver directory, which causes the drivers to only generate error logs.

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

Driver Mode of Operation

The base DMA driver is configured to run in either interrupt mode with MSI or Legacy interrupts, or in polled mode. Only one mode can be selected. The driver is controlled by:

- Adding TH_BH_ISR in the `linux_driver/xdma/Makefile`, or `windows_driver/xdma` which causes the base DMA driver to run in interrupt mode.

By default, polled mode of operation is enabled.

Size of Block Data

The default amount of data being transmitted and received in the block data driver, which changes the throughput observed with this driver, is modified by:

- Modifying NUM_BUFS in `xblockdata/user.c` to change the number of buffers in the free pool available to the driver.

Do not exceed the available system memory when changing these defaults.

Software Driver Code Modifications

This section describes modifications to the software driver code to change design behavior or performance by modifying the block data handler (`xblockdata/user.c`).

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

Design Top-Level Modifications

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

Hardware-Only Modifications

This section outlines the changes that only require parameter changes in the design top-level file (`source/s6_pcie_dma_ddr3_gbe_axi.v`). No change to software is required.

PCIe High-Performance Mode

The Endpoint block for PCI Express provides an optional high-performance mode utilizing extra block RAMs which increases the credits as more packet buffering space is available. This mode can be enabled by defining PCIE_HIGH_PERF during design implementation. Enabling this option shows a change in performance.

Hardware and Software Modifications

This section outlines changes to the top-level design file (`s6_pcie_dma_ddr3_gbe_axi.v`) which also requires software driver modifications.

Jumbo Frames

To enable jumbo frames, transmit and receive Block RAM size in AXI_Ethernet IP is varied by varying the following parameters in the top-level file:

- C_TXMEM in `design/source/s6_pcie_dma_ddr3_gbe_axi.v` modifies the transmit Block RAM depth
- C_RXMEM in `design/source/s6_pcie_dma_ddr3_gbe_axi.v` modifies the receive Block RAM depth

The corresponding change in software requires jumbo frames to be enabled in the Ethernet handler (for Linux):

- Add ENABLE_JUMBO in the `driver/xgbeth/Makefile`

In the Xilinx Ethernet Adapter Properties window, click the **Advanced** tab and select the **Jumbo Frames** property and use the appropriate Jumbo frame MTU size. Figure 5-1 shows Jumbo Frames selection with MTU size of 8KB.

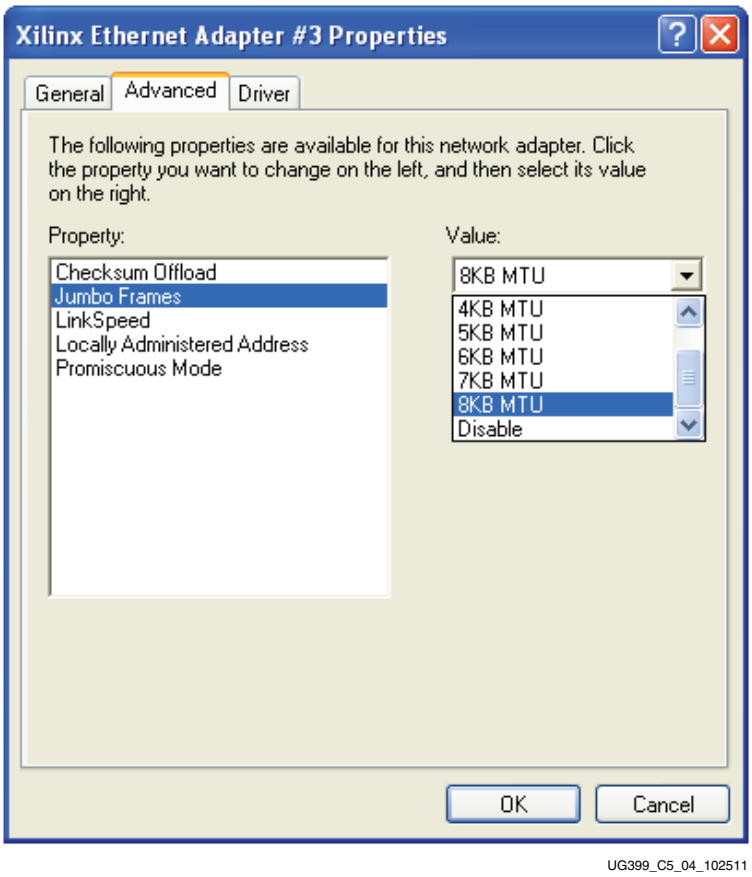


Figure 5-1: **Jumbo Frame Property and Value**

Larger storage for packets implies a smaller number of packets being dropped at the Ethernet receive interface which reduces retransmissions from upper layers in the TCP stack.

PCIe Vendor and Device ID

The vendor and device ID for PCI Express are changed by changing parameters in the top level file:

- CFG_VEN_ID in the file `design/source/s6_pcie_dma_ddr3_gbe_axi.v` changes the vendor ID
- CFG_DEV_ID in the file `design/source/s6_pcie_dma_ddr3_gbe_axi.v` changes the device ID

The corresponding change in software for Linux:

- PCI_VENDOR_ID_DMA: Change this macro in `linux_driver/xdma/xdma_base.c`
- PCI_DEVICE_ID_DMA: Change this macro in `linux_driver/xdma/xdma_base.c`

On Windows, the change of vendor and device ID would be under `windows_driver/xdma/xdma.inx`.

Aurora IP Integration

One architectural modification would be to include LogiCORE™ IP Aurora 8B/10B in the design. This section provides different ways of doing so with specific guidelines.

Aurora IP

The LogiCORE IP Aurora 8B/10B implements the Aurora 8B/10B protocol using the GTP serial transceivers. The core is a scalable, lightweight link-layer protocol for high speed serial communication. It is used to transfer data between two devices using transceivers. It provides an AXI4-Stream interface for data and native flow control.

The core can be generated from CORE Generator™ software with the following options:

- 1- lane, 4 byte duplex framing with immediate native flow control
- 3.125 Gbps line rate using a 125 MHz reference clock and a X1Y0 transceiver in tile-1 (this transceiver is routed to the FMC-LPC on the SP605 board)

The transceiver is programmed to operate in near-end PMA loopback mode by changing the value of the LOOPBACK input. Alternatively, this input can be driven through a register controlled by the software driver.

The Aurora IP core does not provide throttling of the interface in the receive direction (there is no tready signal input). Therefore native flow control is required to prevent loss of packets.

Native Flow Control Logic

Native Flow Control (NFC) logic regulates the data transmission rate at the receiving end of a full-duplex channel. By specifying the number of idle dead beats to be placed into the data stream, NFC logic allows the receiver to control the rate of data sent to it. The data flow can be turned off completely by requesting that the transmitter send only idles.

The IP used in the this configuration operates at a user clock of 78.125 MHz. As indicated by the Aurora protocol specification, the round trip delay through the Aurora interfaces between the NFC request and the first pause arriving at the originating channel partner must not exceed 256 symbol times.

For a 3.125 Gb/s rate, 1 symbol = 10 x 640 ps = 6.4 ns

256 symbol times would be 256 x 6.4 = 1638.4 ns

Using a 78.125 MHz clock (12.8ns) the worst case delay is 128 clock cycles.

This implies that NFC should be asserted when only 128 locations are available in the FIFO.

Design Logic to Buffer the Received Frames in the AXI4-Stream FIFO. Drive the AXI4-Stream NFC interface based on the FIFO occupancy level. This can be accomplished by considering almost full and almost empty levels for the FIFO. When the FIFO is almost full, an XOFF NFC request can be sent which stops further transmission from the channel partner. When the FIFO is almost empty, a XON NFC request can be sent which enables transmission from the channel partner. Pseudocode for NFC Logic is:

```
if (fifo_space_available > fifo_almost_empty_threshold)
     send NFC-XON;
else if (fifo_space_available < fifo_almost_full_threshold)
     send NFC-XOFF;
```

This way, based on FIFO occupancy levels, NFC-XON/XOFF can be controlled thereby preventing loss of frames.

Integration of Aurora IP requires movement of data from PCIe-DMA user clock domain to Aurora user clock domain. Since these are independent clocks, use an asynchronous AXI4-Stream interface FIFO from CoreGEN for crossing clock domains. The asynchronous FIFO should be generated with an option to enable data count ports so that native flow control logic can use the FIFO occupancy levels to drive XON and XOFF.

Replacing Memory Path with Aurora Path

A block diagram for this modification is shown in Figure 5-2. The Aurora IP and the NFC logic designed can be directly plugged in place of the memory path. The same software driver for the block data path would work with no changes required.

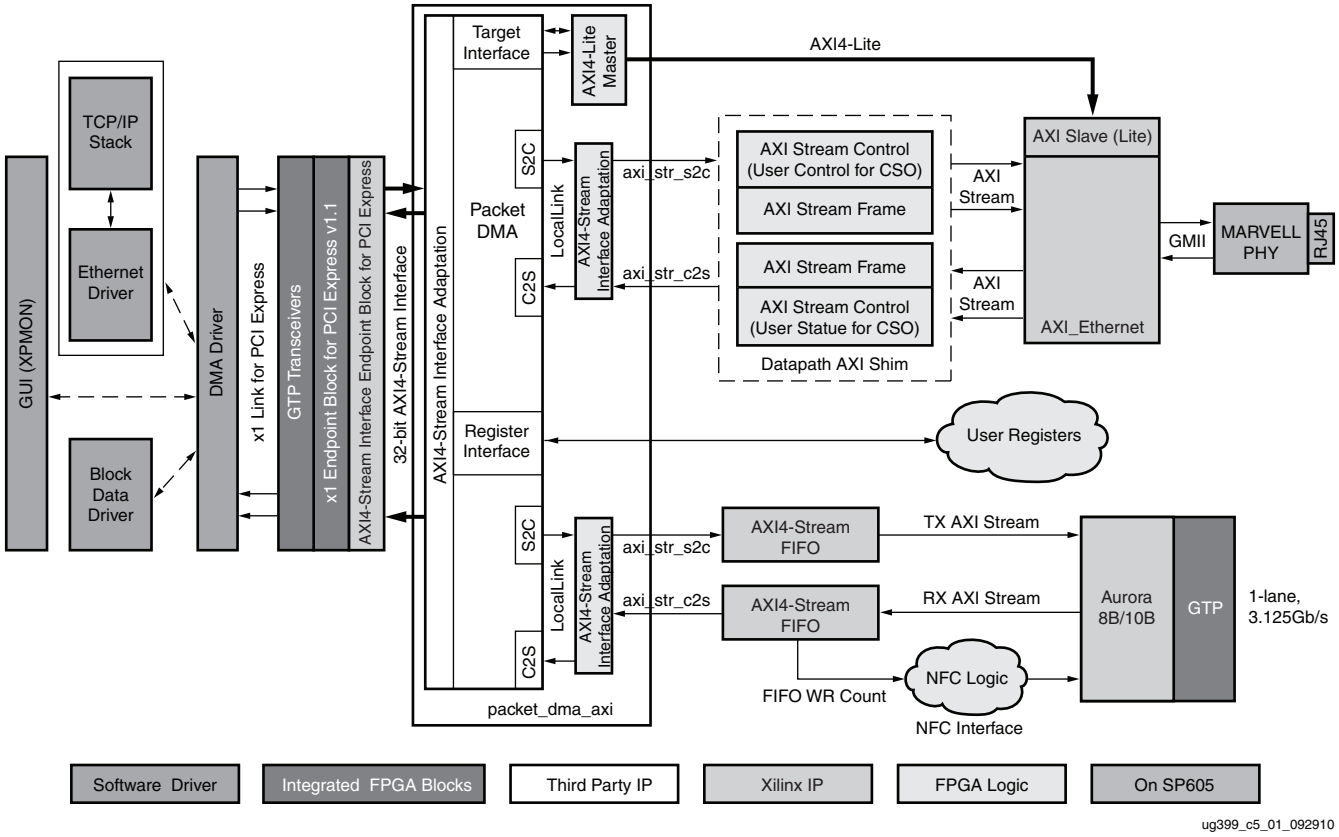


Figure 5-2: **Replacing Memory Path with Aurora Path Modification Block Diagram**

Replacing Network Path with Aurora Path

A block diagram for this modification is shown in Figure 5-3. The Aurora IP and the NFC logic designed can be directly plugged in place of the logic on the network path. Replacing the network path requires modifications to the software driver. The Ethernet driver should not be inserted, and another copy of the block data driver (aurora driver) should be used to drive traffic over Aurora. This requires relevant changes to be made to the aurora driver to map to the appropriate DMA engines.

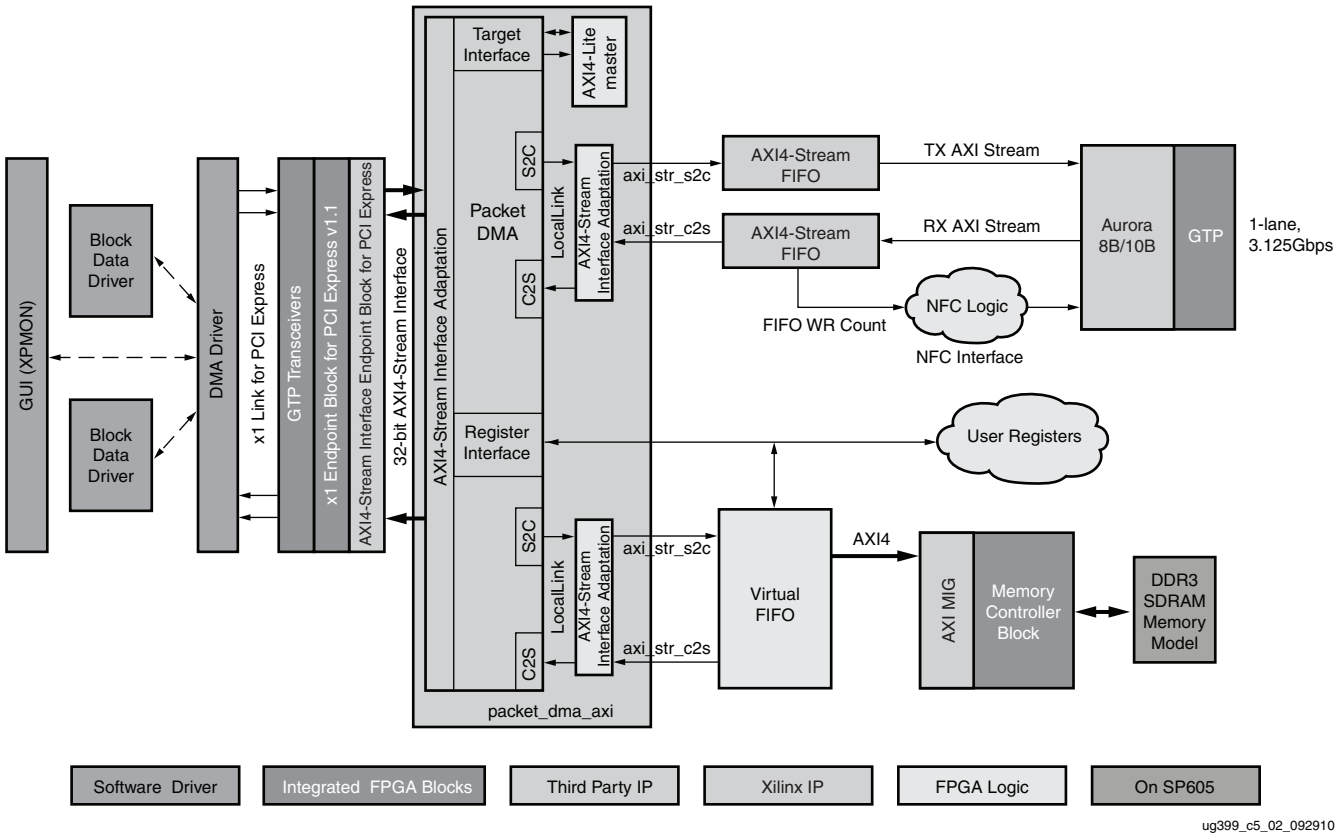


Figure 5-3: **Replacing Network Path with Aurora Path Modification Block Diagram**

Using Multiple Virtual FIFO Instances

The TRD uses one virtual FIFO which utilizes one 32-bit bidirectional port on the Spartan-6 FPGA memory controller block. As additional memory bandwidth is available, the same SDRAM can be partitioned to implement multiple virtual FIFOs. This can be achieved with multiple instances of the virtual FIFO logic. Each virtual FIFO has a dedicated address range within DDR3 SDRAM. These ranges are defined by start and end address ranges for each virtual FIFO instance. Inline processing blocks, if required, can be added as indicated (see Figure 5-4).

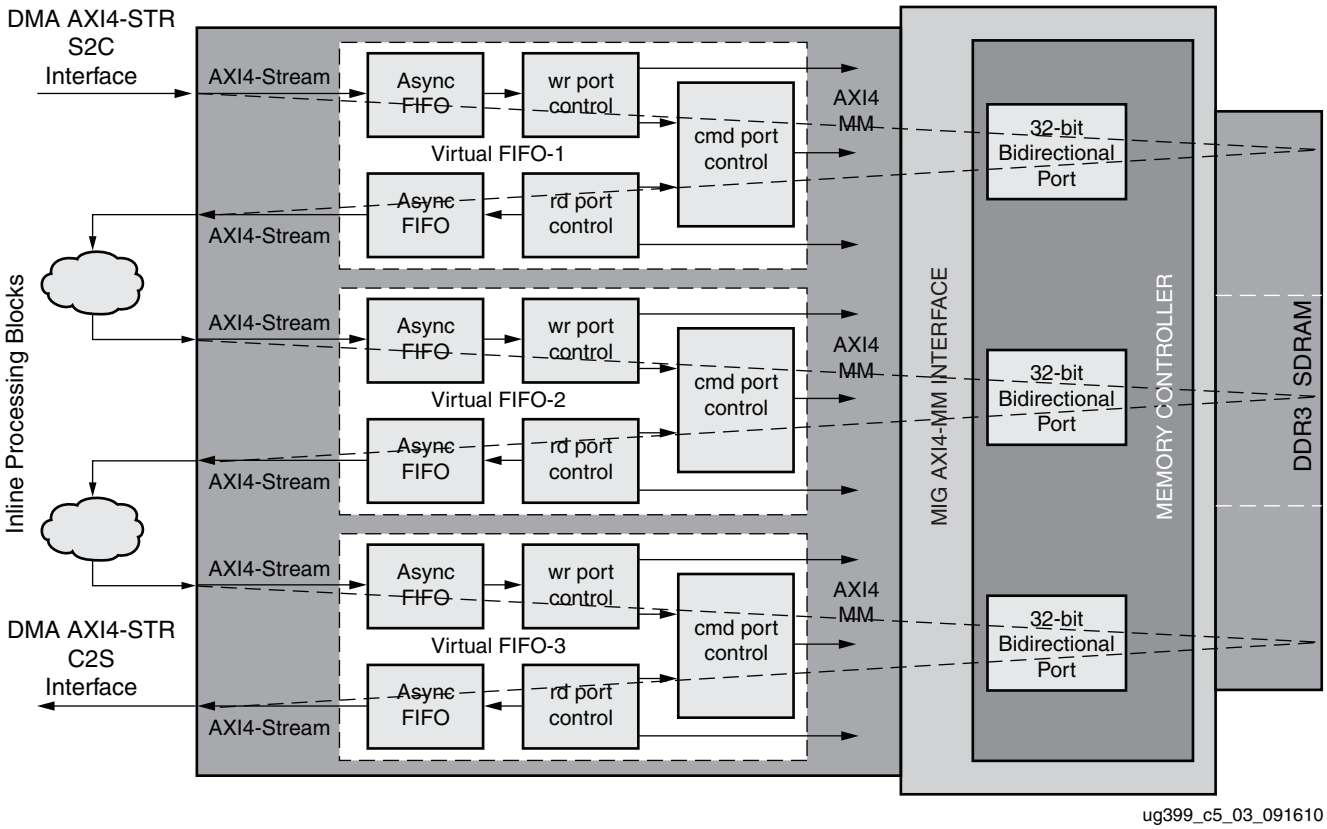


Figure 5-4: **Using Multiple Virtual FIFO Instances**

The design provides hooks to enable multiple virtual FIFO instances:

- Generate MIG IP with multiple 32-bit bidirectional ports. The maximum allowed by the Spartan-6 MCB is four ports. This provides four AXI4-MM interfaces.
- Set the MCB_PORTS = 4 in `design/source/s6_pcie_dma_ddr3_gbe_axi.v`.
- To use all four ports, set NUM_PORTS=4 in `design/source/s6_pcie_dma_ddr3_gbe_axi.v`.
- Modify the multiple virtual FIFO logic connection in `design/source/memory_app/multi_port_vfifo.v`.

The same block data driver can be used to generate traffic. However, performance variation can be observed because of the use of multiple memory controller ports and the resulting arbitration.

If the application developed is for image processing or manipulation, certain image processing operations (for example, image rendering or color inversion,) can be offloaded to hardware as inline processing operations. Other examples of inline processing include operations on chunks of data such as CRC calculation.

XILINX®

Appendix A

Register Description

This appendix is a quick reference to describe the registers programmed by the software driver. For all registers and further details, refer to the specific user guides.

This appendix also describes the hardware registers and mapping of these registers with respect to the base address register (BAR) in PCI Express.

All DMA engine registers are mapped to BAR0. Table A-1 describes the mapping of multiple channel registers.

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 for interrupt handling in the DMA are grouped under a category called common registers. These registers are offset from BAR0 by 0x4000.

Figure A-1 shows the layout of registers.

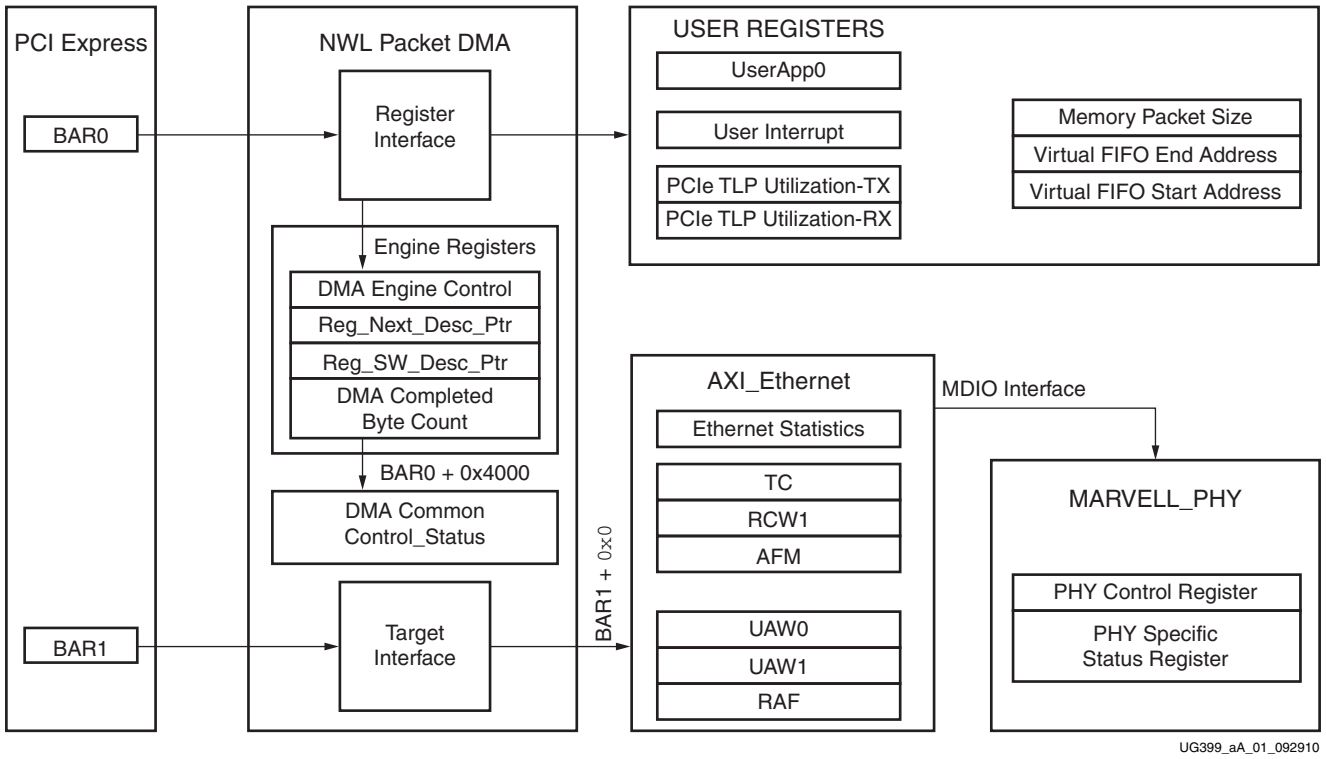


Figure A-1: **Register Map**

The user logic registers are mapped as described in Table A-2. AXI_Ethernet and external PHY registers are mapped to BAR1.

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

User Logic Register Group	Range (Offset from BAR0)
User Application Advertisement Registers	0x8000–0x80FF
User Interrupt Registers	0x8100–0x81FF
TRN Utilization Registers	0x8200–0x82FF
User App0 Registers	0x9000–0x90FF
User App1 Registers	0x9100–0x91FF

DMA Registers

This section describes the prominent DMA registers frequently used by the software driver. For a detailed description of all registers available, please refer to the Northwest Logic Packet DMA Backend Core User Guide, page 10.

Channel Specific Registers

The registers described in Table A-3 through Table A-6 are present in all channels. The address of the register is the channel address offset from BAR0 plus the register offset.

DMA Engine Control (0x0004)

Table A-3: **DMA Engine Control Register (0x0004)**

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 a 1 to clear.
2	Descriptor complete	RW1C	0	Asserted when an interrupt on completion bit is set in the descriptor.
3	Descriptor alignment error	RW1C	0	Asserted when the descriptor address is unaligned and that DMA operation is aborted.
4	Descriptor fetch error	RW1C	0	Asserted when the descriptor fetch errors out. That is, the completion status is not successful.
5	SW_Abort_Error	RW1C	0	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 in operation.
11	DMA_Waiting	RO	0	Indicates DMA waiting on software to provide more descriptors.
14	DMA_Reset_Request	RW	0	Issues a request to user logic connected to DMA to abort outstanding operation and prepare for reset. This is cleared when user acknowledges the reset request
15	DMA_Reset	RW	0	Asserting this bit resets the DMA engine and issues a reset to the 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: 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: The location of the first descriptor in the chain, which 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 second. It has a resolution of four bytes.
[1:0]	Sample Count	RO	0	Sample Count: Incremented every time a sample is taken at a one second interval.

Common Registers

The registers described in this section are common to all engines and 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	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 the global interrupt enable.
2	Interrupt Pending	RO	0	Reflects the state of the DM A interrupt output without considering the state of the global interrupt enable.
3	Interrupt Mode	RO	0	0: MSI mode 1: Legacy interrupt mode
4	User Interrupt Enable	RW	0	Enables generation of user interrupts.
5	User Interrupt Active	RW1C	0	Indicates an active user interrupt.
23:16	S2C Interrupt Status	RO	0	Bit[i] indicates interrupt status of S2C DMA engine[i]. If S2C engine is not present, then this bit is read as zero.
31:24	C2S Interrupt Status	RO	0	Bit[i] indicates interrupt status of C2S DMA engine[i]. If C2S engine is not present, then this bit is read as zero.

Network Path IP Registers

This section defines the commonly used AXI_Ethernet and PHY registers. For a detailed explanation of all registers, please refer to the respective user guides.

AXI_Ethernet Registers

The AXI_Ethernet contains memory and addressable registers for read/write operations. All registers are directly accessible.

The register offsets are determined by parameter C_BASEADDR (set to zero in the design) from BAR1.

Statistics Registers

Only certain statistics registers which indicate errors are read in the design.

Table A-8: **Statistics Register**

Offset	Register	Description
0x298	FCS Errors (lower 32 bits)	A count of received frames that failed the CRC check and were at least 64 bytes in length.
0x2B8	Length/Type out of range (lower 32 bits)	A count of frames received that had length/type field not matching the number of data bytes received.
0x2F0	Underrun Errors (lower 32 bits)	A count of frames that would otherwise be transmitted but could not be completed due to FIFO underrun.

Receive Configuration Word1 Register (0x404)

This register sets the behavior of the receive TEMAC interface.

Table A-9: **Receive Configuration Word1 Register**

Bit	Field	Mode	Default Value	Description
31	RST	RW	0	Reset: When this bit is 1, the receiver is reset. The bit automatically resets to 0. The reset also sets all of the receiver configuration registers to default values. 0: No reset 1: Initiates a receiver reset
30	JUM	RW	1	Jumbo Frame Enable: When this bit is 1, the receiver accepts frames over the maximum length specified in the IEEE 802.3 specification. 0: Receive jumbo frames disabled 1: Receive jumbo frames enabled
29	FCS	RW	1	In-Band FCS Enable: When this bit is 1, the receiver provides the FCS field with the rest of the frame data. When this bit is 0 the FCS field is stripped from the receive frame data. In either case the FCS field is verified. 0: Strip the FCS field from the receive frame data 1: Provide the FCS field with the receive frame data
28	RX	RW	1	Receive Enable: When this bit is 1, the receiver logic is enabled to operate. When this bit is 0, the receiver ignores activity on the receive interface. 0: Receive disabled 1: Receive enabled
25	LT_DIS	RW	0	Length/Type Field Valid Check Disable: When this bit is 1, it disables the Length/Type field check on the receive frame. 0: Perform Length/Type field check 1: Do not perform Length/Type field check

Transmit Configuration Word Register (0x408)

This register sets the behavior of the transmit path of the TEMAC.

Table A-10: **Transmit Configuration Word Register**

Bit	Field	Mode	Default Value	Description
31	RST	RW	0	Reset. When this bit is 1, the transmitter is reset. The bit automatically resets to 0. The reset also sets all of the transmitter configuration registers to their default values. 0: no reset 1: initiates a transmitter reset
30	JUM	RW	1	Jumbo Frame Enable When this bit is 1, the transmitter sends frames over the maximum length specified in IEEE 802.3 specification. 0: send jumbo frames disabled 1: send jumbo frames enabled
29	FCS	RW	0	In-Band FCS Enable. When this bit is 1, the transmitter accepts the FCS field with the rest of the frame data. When this bit is 0 the FCS field is calculated and supplied by the transmitter. 0: transmitter calculates and sends FCS field 1: FCS field is provided with transmit frame data
28	TX	RW	1	Transmit Enable. When this bit is 1, the transmit logic is enabled to operate. 0: transmit disabled 1: transmit enabled

MII Management (MDIO) Configuration Register (0x500)

This register programs the MDIO clock divider and successful programming of this register generates the MDIO clock used to program the PHY.

Table A-11: **MII Management Configuration Register**

Bit	Field	Mode	Default Value	Description
6	MDIO_EN	RW	0	MDIO Enable: When this bit is 1, the MDIO (MII Management) interface is used to access the PHY. 0: MDIO disabled 1: MDIO enabled
5:0	CLK_DIVIDE	RW	0	Clock Divide: This value is used to derive the MDC (MII Management interface clock) signal. The maximum permitted frequency is 2.5 MHz.

User Application Registers

The various user registers are described in this section. All registers are 32 bits wide. Bit fields not defined are considered reserved with a read always returning a value of zero.

Design Version Register (0x8000)

This register tracks the design version so that code maintenance is easily traceable. The software driver uses this register to associate the correct version with the hardware design.

Table A-12: **Design Version Register**

Bit Location	Field	Mode	Default Value	Description
31:28	Device	RO	0000	• 0000: Spartan-6 • 0001: Virtex-6
11:4	Version	RO	0001_0000	Defines TRD version; updated based on release versions.
3:0	Sub-version	RO	0001	AXI4 protocol version of design.

User Application Advertisement Registers

UserApp Advertisement Register (0x8004)

This register advertises which user application is connected to which DMA engine. This enables the software to associate appropriate descriptors with relevant DMA engines in case Ethernet path and memory path are swapped in hardware.

The following bits in the register are engine indicators:

- [31:28]: S2C Engine 0
- [27:24]: S2C Engine 1
- [15:12]: C2S Engine 0
- [11:8]: C2S Engine 1

Table A-13: **UserApp Advertisement Register**

Bit Location	Field	Mode	Default Value	Description
31:28	S2C_0	RO	0001	A value of 0001 indicates network path connected to S2C engine-0.
27:24	S2C_1	RO	0010	A value of 0010 indicates memory path connected to S2C engine-1.
15:12	C2S_0	RO	1001	A value of 1001 indicates network path connected to C2S engine-0.
11:8	C2S_1	RO	1010	A value of 1010 indicates memory path connected to C2S engine-1.

PCI Express Transaction Layer Monitor Registers

This defines the registers implemented for measuring transaction layer utilization.

These registers are updated once every second by hardware. These registers have a resolution of four bytes and provide a 2-bit sample count which increments every second. The sample count provides a mechanism for software to keep track of distinct reads and also to synchronize register values across the same one second interval.

Transmit Utilization Byte Count (0x8200)

This register counts the utilization of the transmit interface of the PCIe core. It increments every clock cycle when both s_axis_tx_tvalid and s_axi_tx_tready are asserted.

Table A-14: **Transmit Utilization Byte Count Register**

Bit Location	Field	Mode	Default Value	Description
31:2	Transmit Utilization Count	RO	0	Gives the count when TLP transmission was active. This register has a resolution of four bytes. Multiply the value obtained by four to get the byte count.
1:0	Sample Count	RO	0	A 2-bit sample count which increments once every second.

Receive Utilization Byte Count (0x8204)

This register counts the utilization of the receive interface of the Endpoint for PCI Express. It increments every clock cycle when both m_axis_rx_tvalid and m_axis_rx_tready are asserted.

Table A-15: **Receive Utilization Byte Count**

Bit Location	Field	Mode	Default Value	Description
31:2	Receive Utilization Count	RO	0	This gives the count when TLP reception was active. This register has a resolution of four bytes. Multiply the value obtained by four to get the byte count.
1:0	Sample Count	RO	0	2-bit sample count which increments once every second.

Upstream Memory Write Byte Count (0x8208)

This register counts the payload of memory write transactions sent upstream on the transmit interface of the PCIe core.

Table A-16: **Upstream Memory Write Byte Count**

Bit Location	Field	Mode	Default Value	Description
31:2	MWR Payload Count	RO	0	This gives the count of MWR payload bytes sent across the transmitted TLP. This register has a resolution of four bytes. Multiply the value obtained by four to get the byte count.
1:0	Sample Count	RO	0	2-bit sample count which increments once every second

Downstream Completion Payload Byte Count (0x820C)

This register counts the payload of completion transactions received at the Endpoint on the receive interface for PCI Express.

Table A-17: **Downstream Completion Payload Byte Count**

Bit Location	Field	Mode	Default Value	Description
31:2	CplD Payload Count	RO	0	Gives the count of the CplD payload bytes in the TLP received. This register has a resolution of four bytes. Multiply the value obtained by four to get the byte count.
1:0	Sample Count	RO	0	2-bit sample count which increments once every second

PCIe Transaction Monitor Control (0x8210)

This is the monitor control registers which defines a software controlled reset. When asserted, this clears the counters.

Table A-18: **TRN Monitor Control Register**

Bit Location	Field	Mode	Default Value	Description
0	Monitor Reset	RW	0	Monitor Soft Reset: When 1, resets the TRN monitor counters.

User App0 Registers

Ethernet Options Register (0x9000)

Provides options with which Ethernet design is generated. It is useful when both partial and full checksum comes into picture as selection is a compile time option in hardware.

This is applicable only to the AXI4 protocol design.

Table A-19: **Ethernet Options Register**

Bit Location	Field	Mode	Default Value	Description
31:30	Type	R0	01	00: Soft TEMAC with 10/100 01: Soft TEMAC with 10/100/1000
29:24	PHY Type	RO	000001	000001: GMII 000101: 1000BASE-X
7:4	Statistics	RO	0001	0001: Ethernet Statistics enabled 0000: Ethernet Statistics disabled
3:2	TX CSUM	RO	01	00: No CSO 01: Partial CSO 10: Full CSO
1:0	RX CSUM	RO	01	00: No CSO 01: Partial CSO 10: Full CSO

User App1 Registers

This group defines the registers specific to user application connected to DMA channel-1, which is the memory application for this design.

Virtual FIFO Options and Status Register (0x9100)

This register indicates the status of DDR3 SDRAM calibration to the software driver and also provides a placeholder for options. It enables software to determine if the hardware is ready for operation.

Table A-20: **Virtual FIFO Status Register**

Bit Location	Field	Mode	Default Value	Description
31	Calibration Status	RO	0	Calibration Done: This bit indicates calibration done status from memory controller.
15:0	Number of Ports	RO	1	Indicates number of ports used.

Virtual FIFO Receive Packet Length Register (0x9110)

This register indicates the size of the packet in bytes to be built in the receive direction. It initializes with a default value of 1 KB.

Table A-21: **Virtual FIFO Receive Packet Length Register**

Bit Location	Field	Mode	Default Value	Description
31:0	Packet Length	RW	32'h0000_0400	DDR3 Receive Packet Length: Indicates the size of the packet (in bytes) to be built in the receive direction.

Virtual FIFO Start Address Register (0x9114)

This register indicates the start address for DDR3 partition. It initializes with the default value of zero on reset. Software programming of this register is optional.

Table A-22: **Virtual FIFO Start Address Register**

Bit Location	Field	Mode	Default Value	Description
31:0	Start Address	RW	0x0	DDR3 Start Address: Indicates the start address in DDR3 from where virtual FIFO starts.

Virtual FIFO End Address Register (0x9118)

This register indicates the end address for DDR3 partition. It initializes with the default value of 32'h07FF_FFFF on reset. Software programming of this register is optional.

Table A-23: **Virtual FIFO End Address Register**

Bit Location	Field	Mode	Default Value	Description
31:0	End Address	RW	32'h07FF_FFFF	DDR3 End Address: Indicates the end address in DDR3 where the virtual FIFO wraps around to the start address.

Virtual FIFO Write Burst Size Register (0x911C)

This register programs the write burst size for AXI-MM transactions. Default value is set to 256.

Table A-24: **Virtual FIFO Write Burst Size Register**

Bit Location	Field	Mode	Default Value	Description
8:0	Write Burst Length	RW	9'd256	Write burst size for AXI-MM write transactions.

Virtual FIFO Read Burst Size Register (0x9120)

This register programs the read burst size for AXI-MM transactions. Default value is set to 256.

Table A-25: **Virtual FIFO Read Burst Size Register**

Bit Location	Field	Mode	Default Value	Description
8:0	Read Burst Length	RW	9'd256	Read burst size for AXI-MM Read transactions.

Virtual FIFO Error Statistics Register (0x9124)

This register is the DDR3 error statistics register which records an error count on DDR3. This register accumulates the DDR3 error count and is cleared on reset.

Table A-26: **Virtual FIFO Error Statistics Register**

Bit Location	Field	Mode	Default Value	Description
31:0	Error Stats	RW	0	DDR3 Error Statistics.

XILINX®

Appendix B

Directory Structure

Introduction

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

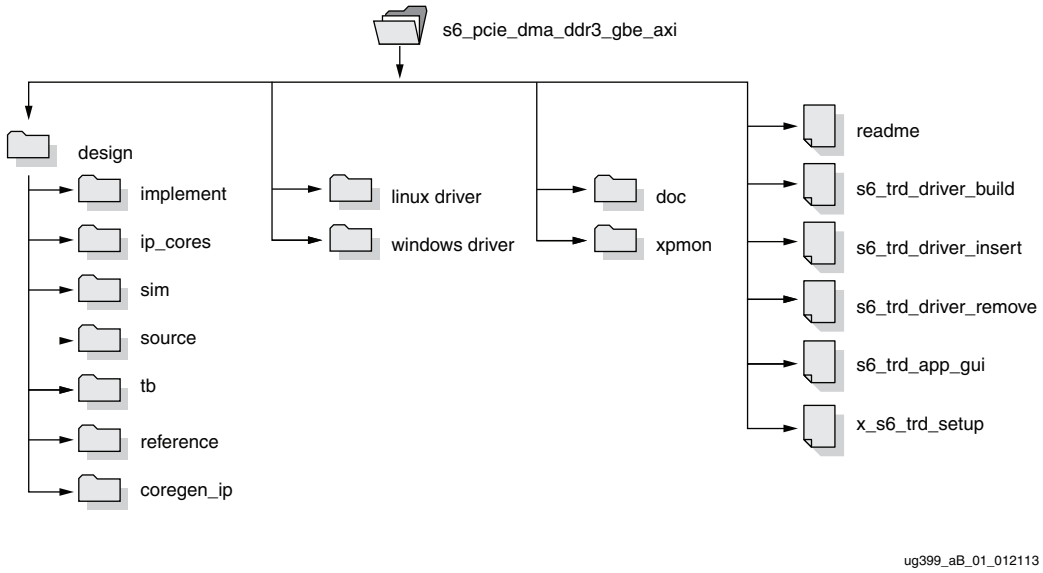


Figure B-1: **Directory Structure**

Design

The design folder contains all the hardware design deliverables.

- implement: Contains the implementation scripts for the design for both windows and Linux operating systems supporting command line flow and PlanAhead™ flow
- ip_cores: Contains the third-party DMA IP related files and Xilinx IP files modified for this TRD
- sim: Contains the simulation scripts for supported simulators for both windows and Linux operating systems
- source: Contains the source code deliverable files
- tb: Contains the testbench related files for simulation
- reference: Bit files and MCS files for golden reference and XCO files generated by Xilinx cores

- coregen_ip: Includes the CORE Generator IP and netlists used in the design

linux_driver

Source code of required drivers for Linux OS

windows_driver

Source code of required drivers and application GUI for Windows OS

xpmon

Contains source code for the application GUI on Linux.

doc

Contains the TRD user guide.

readme

Details the use of various simulation and implementation scripts.

s6_trd_driver_build

Contains the script to build the driver and GUI modules.

s6_trd_driver_insert

Contains the script to insert the driver modules.

s6_trd_driver_remove

Contains the script to remove the driver modules.

s6_trd_app_gui

Contains the script to invoke the XPMON application GUI

x_s6_trd_setup

This is the executable for Windows driver setup.

XILINX®

Appendix C

AXI4 Interface Migration Considerations for TRD

This appendix describes the migration considerations for users starting on AXI4 protocol. The following sections discuss the dominant changes in the TRD for AXI4 interface migration.

- The interface between the Endpoint Block for PCI Express® and Packet DMA changes from TRN to AXI4-Stream interface

  The changes from the TRN interface to the AXI4-Stream interface are described in Appendix C of UG672, *Spartan-6 FPGA Integrated Block for PCI Express User Guide* (*Migration Considerations*).

- The Packet DMA LocalLink-like user interface changes to the AXI4-Stream interface

  The corresponding mapping between the LocalLink-like DMA user interface and the AXI4-Stream interface is explained in Data Plane Interface, page 67. As the AXI4-Stream interface has no concept of "start-of-packet" in the S2C direction, the adaptation layer ignores the s2c_sop signal from the DMA interface (refer to `design/source/common/s2c_axi_str.v`).

  In the C2S direction, c2s_sop is provided to the DMA user interface. This is done by building SOP by tracking end of packet (`design/source/common/axi_str_c2s.v`).

```
//- Logic to build SOP across C2S
  assign valid_beat = axi_str_c2s_tvalid & axi_str_c2s_tready;
  always @(posedge axis_aclk)
    if (~axis_areset_n | (axi_str_c2s_tlast & valid_beat))
      expect_sop_flag <= 1'b1;
    else if (expect_sop_flag & valid_beat)
      expect_sop_flag <= 1'b0;
  assign c2s_sop              = expect_sop_flag & valid_beat;
```

  The user control and user status signal is mapped to tuser in the AXI4-Stream interface. Care is to be exercised when passing this through an upsizer or downsizer block to maintain the correct signal timing and value.

- Packet DMA target interface changes to the AXI4-Lite interface

  The module `design/source/common/target_axi_control.v` converts the DMA provided target interface to AXI4-Lite master interface so that it can directly plug into an AXI4 slave slot. Although target interface supports burst transactions, for the TRD only single DWORD transactions are used, therefore a Lite wrapper is designed to save on resources.

The target interface has independent write and read interfaces, therefore mapping this to AXI4-Lite interface's address write, write data, write response, address read and read data channels becomes simple.

On the target write interface, on write transaction availability, the write command is acknowledged so that write address and data is made available. The write address AXI4-Lite channel is driven with the address received, and write data with the data received. The write response channel, in case of an erroneous response, is mapped to a status register.

On availability of a target read command, the read address AXI4-Lite interface channel is driven and data is obtained on the read data channel. When data is received from the AXI4-Lite interface, the target read command phase is granted and data read is provided.

Because the transactions on the target interface arrive only after address decoding, meaning it maps to valid user space, all completions over the target interface have a successful status. In the case of user space register holes, a completion value with a pattern of `32'hDEAD_DEAD` is returned.

AXI4 Interconnect Use Model

Because the TRD has only one master and one slave, the AXI4-Lite interconnect is not required in between, however, the following use models can be considered for future use with inclusion of AXI4-Lite interconnect:

- **1:N Interconnect Scheme**: This is the model (see Figure C-1) that can be used to connect to multiple slaves when all user space registers are mapped to the same non-zero base address register (BAR 1-5).

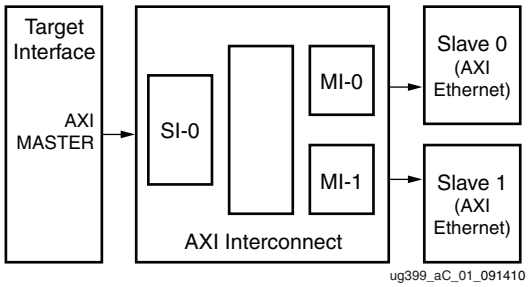


Figure C-1: **1:N Interconnect Use Model**

- **M:N Interconnect Scheme**: This is the model (see Figure C-2) that can be used to connect multiple masters and multiple slaves when different BARs are used so that masters can be separated out for various BARs.

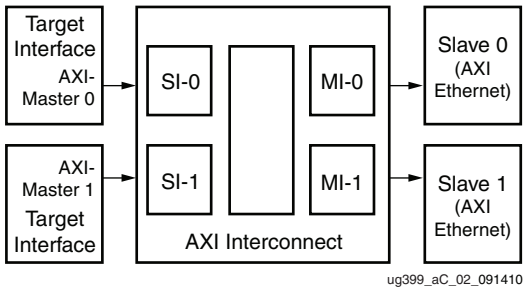


Figure C-2: **M:N Interconnect Use Model**

In the TRD, this migration of the Packet DMA to AXI4 interface is already done (see Figure C-3). All of the relevant modules implementing the adaptation layers are provided as source code. Note that in diagrams depicting AXI4 interfaces, arrow directions are always shown from master to slave.

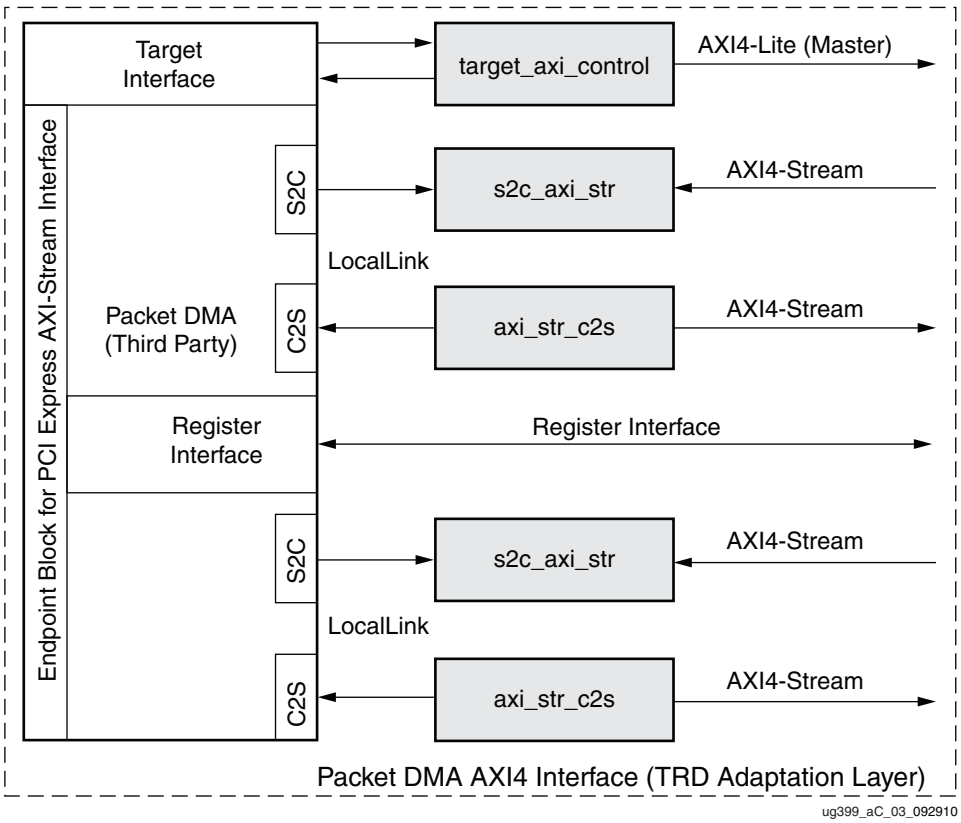


Figure C-3: **AXI4 Interface Migration**

XILINX®

Appendix D

Setting Up a Private LAN

Introduction

This section describes how to set up a private LAN connection between two machines for Ethernet performance measurement.

Figure D-1 shows a private LAN connection between two machines.

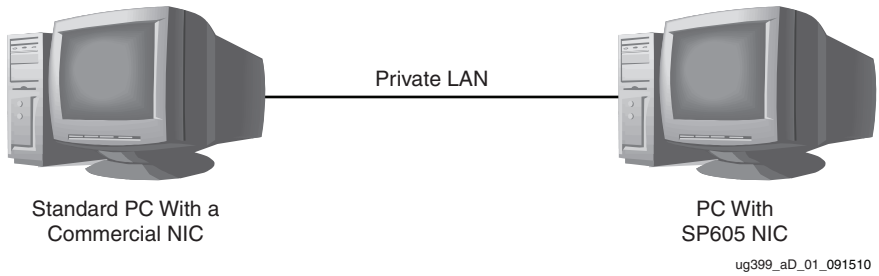


Figure D-1: **Private LAN Setup**

To set up a private LAN connection, connect an Ethernet cable between the two machines; connect as a private LAN setup. One of them is a standard machine which has a commercial network interface card (NIC) and the other has SP605 NIC. The machine with SP605 NIC is referred to as the unit-under-test (UUT) and the other machine, with a commercial NIC, is referred to as the standard machine.

Linux Flow and Netperf

1. Assign a static IP address on both machines. Make sure that they have the same subnet mask. Using a terminal in command line mode, execute:

   ```
   $ ifconfig ethX up 172.16.64.7
   ```

   For this example, the standard machine is assigned an IP address of `172.16.64.9` and the UUT is assigned an IP address of `172.16.64.7`.
2. After the interface is activated and the static IP addresses are assigned, try a ping between the machines.
3. Install Netperf v2.4 on both machines. Netperf works with a client server model. In this setup, UUT is programmed as the client and the other standard machine as the server. On a terminal on the standard machine, invoke netserver:

   ```
   $ netserver
   ```
4. Open a terminal on the UUT and run Netperf

   ```
   $ netperf -H <IP-address-standard machine>
   ```

This command runs a ten second TCP_STREAM test by default and reports outbound performance. Refer to the Netperf manual for the various test options available.

Windows Flow and Iperf

To configure the IP address of the NIC from Windows, do the following:

1. Double-click **Internet Protocol (TCP/IP)**. The Internet Protocol (TCP/IP) properties dialog box opens (Figure D-2).
2. Select **Use the following IP address**.
3. Enter an IP address in the **IP Address** field. Click **OK**.

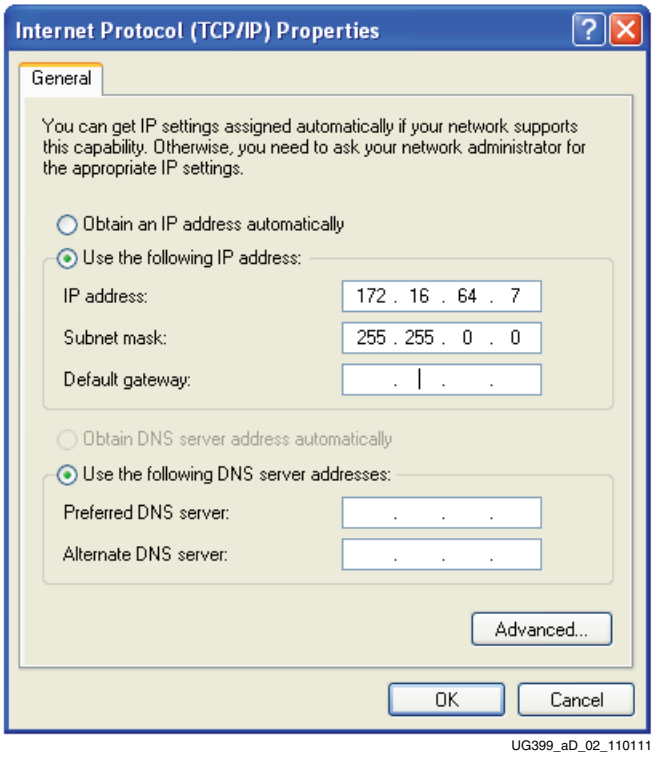


Figure D-2: **IP Address Configuration**

Use the iperf or netperf utility to run the network performance test. In the Windows command prompt, browse through where iperf or netperf executable is installed. In one peer machine, run iperf or netperf server and on another machine run iperf or netperf client.

1. To run iperf server, execute the command **`<PATH> iperf -s`**. Figure D-3 shows running iperf server on command prompt.

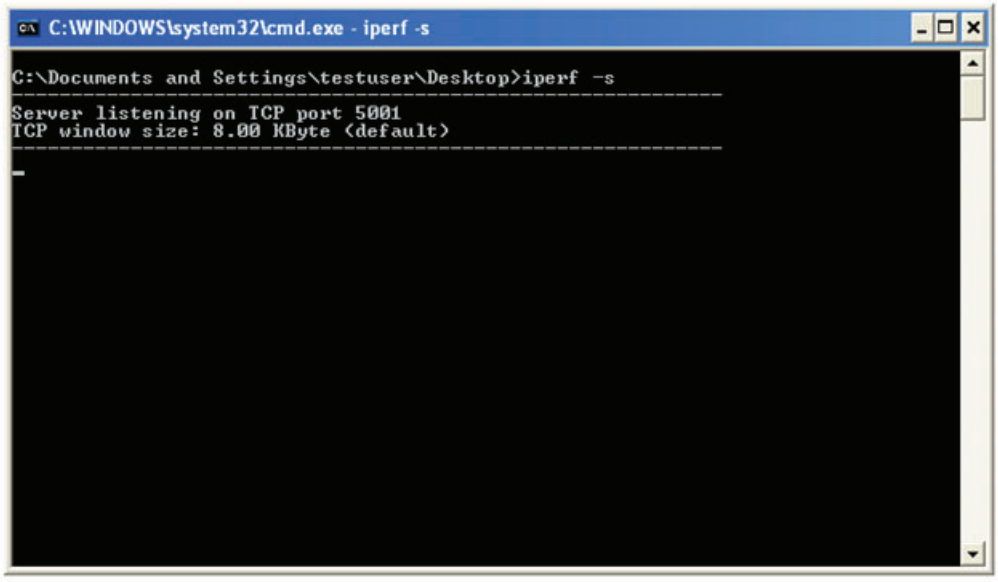


Figure D-3: **Iperf Server**

1. To run iperf client, execute the command **<PATH>iperf -c <IP address> -i <interval> -t <time>**. Where:

   <interval> = seconds between periodic bandwidth report
   <time> = transmit time in seconds

   Figure D-4 shows running iperf server on command prompt.

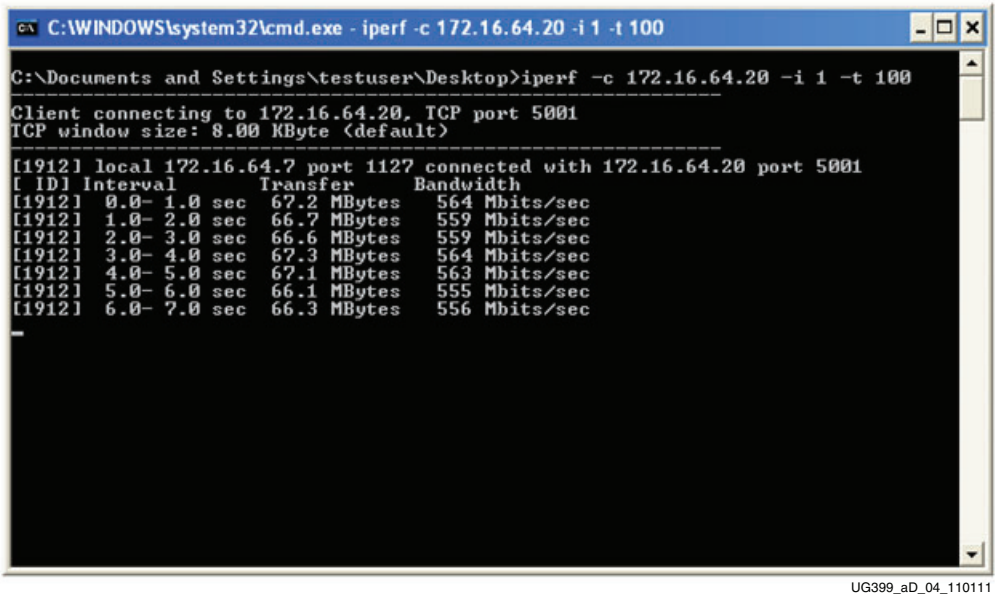


Figure D-4: **Iperf Client**

XILINX®

Appendix E

Troubleshooting

Introduction

This section includes some troubleshooting tips (Table E-1). It is not meant as an exhaustive troubleshooting guide. It is based on the following assumptions:

- User has followed instructions as explained in Chapter 2, Getting Started.
- User has made sure that the PCI Express link is up and the Endpoint device is discovered by the host and can be seen with `lspci`.
- Visual indicators (LEDs) as listed on page 19 are functioning and have been checked.

Table E-1: **Troubleshooting Tips**

Issue	Possible Resolution
Activation of Ethernet interface fails with network configuration GUI	1. Check the MAC address to make sure the MAC address is programmed as provided with the SP605 Connectivity Kit. 2. If assigning an IP address statically, make sure that it does not clash with any other IP address on the network. Contact the network administrator regarding specific IP address allocation. 3. In the network configuration GUI, under the Devices tab, for the device, uncheck the **bind to MAC address** option in device properties. Device properties are invoked by double-clicking on the device.
Network is connected but webpage does not load in the browser	1. Check the browser's network proxy settings suitable for your network with your network administrator 2. Make sure that the browser is not in work-offline mode
1000BASE-X Design not working.	1. Make sure that the additional required hardware is connected as explained in Testing 1000BASE-X Mode, page 56. Make sure the Ethernet connection is 1 Gb/s. 2. Make sure that the correct design is downloaded to hardware (`sp605_use_1000basex`) 3. Make sure that `driver/xgbeth/Makefile` has **DUSE_1000BASEX** defined under **EXTRA_CFLAGS**
ModelSim simulation does not work	1. Make sure that the libraries required for simulation with ModelSim are compiled and that the correct initialization file is used.

XILINX®

Appendix F4

Compiling Windows Drivers

If the driver source code is modified, the device drivers must be recompiled. This appendix describes how to compile Windows drivers using the Windows Driver Kit (WDK), available for download from Microsoft. The instructions and screen captures that follow are based on WDKv7.1.0, but are applicable for all versions of WDK.

The Windows driver source code is located in directory: `s6_pcie_dma_ddr3_gbe_axi/windows_driver`.

WDK Installation

Go to http://www.microsoft.com/whdc/devtools/wdk/wdkpkg.mspx and follow the instructions for downloading and installing the WDK.

Using WDK

The WDK provides two build environments:

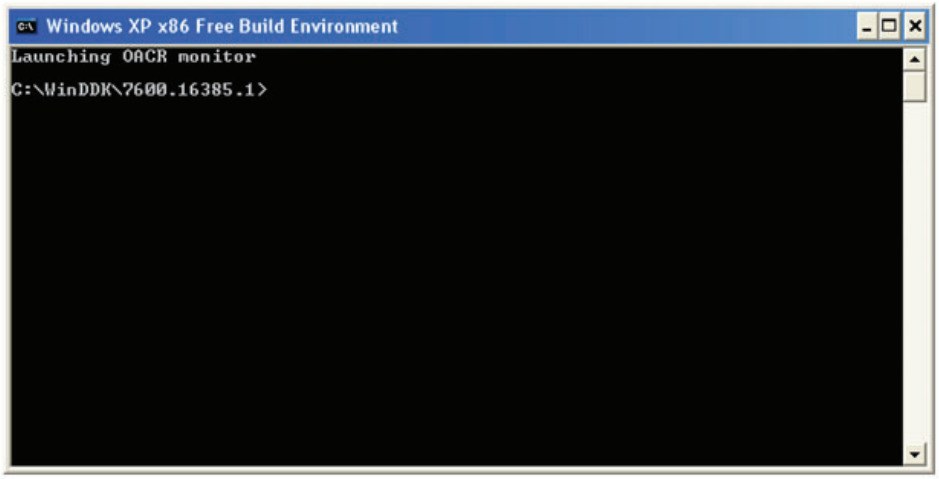
- Free build – This environment creates drivers in which code is optimized and debugging is disabled. Use the free build environment to compile production drivers and drivers used for performance testing.
- Checked build – This environment creates drivers in which code is not optimized and includes conditional code to enable debugging. Use the checked build environment to compile drivers during development.

Note: Follow the MS-DOS 8.3 naming convention when using WDK. For example, placing code under a Windows path name like `C:\Documents and settings` will not work in the build command shell. Either change the path or use `C:\DOCUME~1` to navigate to the relevant folder.

Compiling the Drivers

To compile the XDMA, XINC, and XBLOCK drivers, follow these steps:

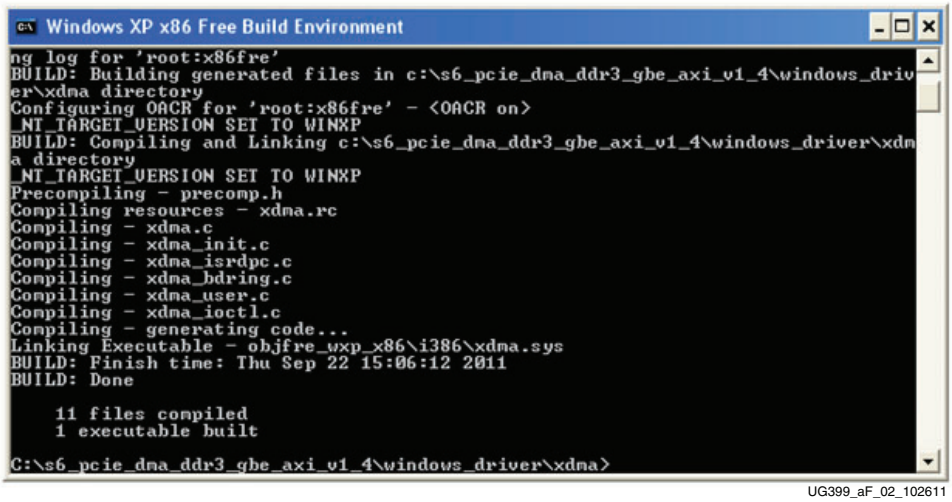
1. Start the WDK free build environment. Choose **Start** → **All Programs** → **Windows Driver Kit** → **WDK<version>** → **Build Environments** → **[Windows XP]**. The free build environment window opens (Figure F-1).



UG399_aF_01_102611

Figure F-1: **Free Build Environment**

2. Compile the XDMA code by navigating to the `windows_driver/xdma` directory. From the command line, execute the command **`build /w -cezZ`**. This command invokes the Microsoft make routines and builds the driver components. Results are shown in Figure F-2.



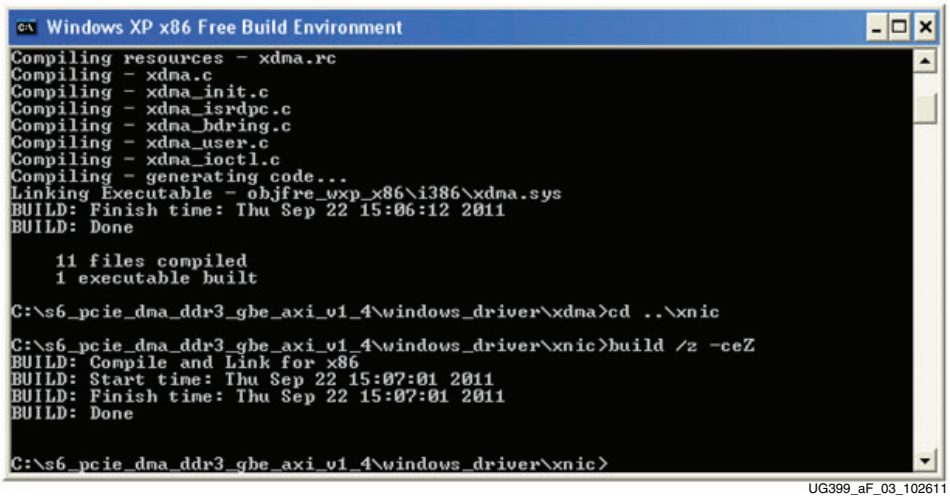
UG399_aF_02_102611

Figure F-2: **XDMA Driver Build Results**

On successful completion of the build, the driver file `xdma.sys` and the setup information file `xdma.inf` are created in the `objfre_wxp_x86\i386` directory.

Note: If the WDK checked build environment is used, the driver file and the setup information file are created in the `objchk_wxp_x86\i386` directory.

3. Compile the XNIC code by navigating to the `windows_driver/xnic` directory. From the command line, execute the command **`build /w -ceZ`**. This command invokes the Microsoft make routines and builds the driver components. Results are shown in Figure F-3.



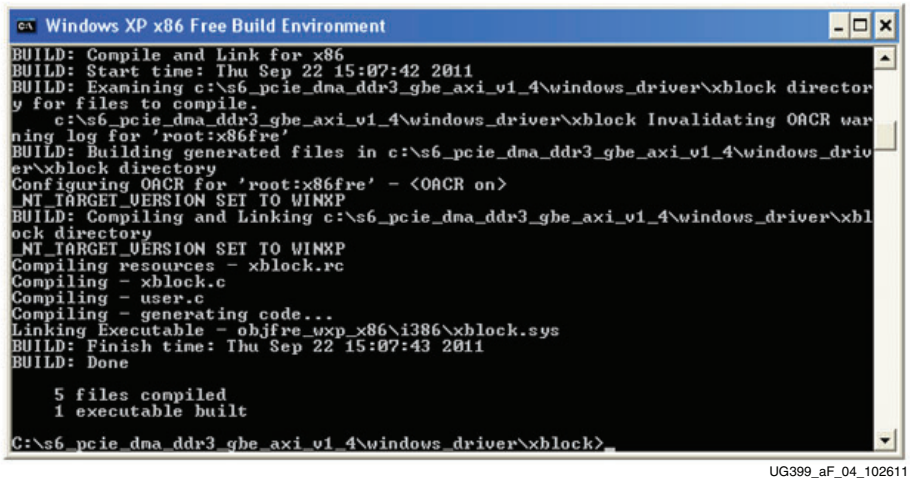
UG399_aF_03_102611

Figure F-3: **XNIC Driver Build Results**

On successful completion of the build, the driver file `xnic.sys` and the setup information file `xnic.inf` are created in the `objfre_wxp_x86\i386` directory.

Note: If the WDK checked build environment is used, the driver file and the setup information file are created in the `objchk_wxp_x86\i386` directory.

4. Compile the block datapath code by navigating to the `windows_driver/xblock` directory. From the command line, execute the command **`build /w -ceZ`**. This command invokes the Microsoft make routines and builds the driver components. Results are shown in Figure F-4.



UG399_aF_04_102611

Figure F-4: **XBLOCK driver Build Results**

On successful completion of the build, the driver file `xblock.sys` and the setup information file `xblock.inf` are created in the `objfre_wxp_x86\i386` directory.

Note: If the WDK checked build environment is used, the driver file and the setup information file are created in the `objchk_wxp_x86\i386` directory.

Create a Recompiled Driver Package

To create the recompiled driver package, follow these steps:

1. Create a directory named `compiled_drivers`.
2. In the `compiled_drivers` directory, create a directory named `xdma`.
3. Copy `xdma.sys` and `xdma.inf` from the `xdma\objfre_wxp_x86\i386` from the XDMA driver compilation directory to the `compiled_drivers\xdma` directory.
4. In the `compiled_drivers` directory, create a directory named `xnic`.
5. Copy `xnic.sys` and `xnic.inf` from the `xnic\objfre_wxp_x86\i386` from the XNIC driver compilation directory to the `compiled_drivers\xnic` directory.
6. In the `compiled_drivers` directory, create a directory named `xblock`.
7. Copy `xblock.sys` and `xblock.inf` from the `xblock\objfre_wxp_x86\i386` from the XBLOCK driver compilation directory to the `compiled_drivers\xblock` directory.
8. Copy the GUI executable `xpmon.exe` from the `s6_pcie_dma_ddr3_gbe_axi/windows_driver/xpmon` directory to the `compiled_drivers\` directory.
9. Copy `WdfCoInstaller01009.dll` from the `s6_pcie_dma_ddr3_gbe_axi/windows_driver/utils` directory and place one copy each under to the `xdma`, `xnic`, and `xblock` directories.

The resulting directory structure is shown in Figure F-5.

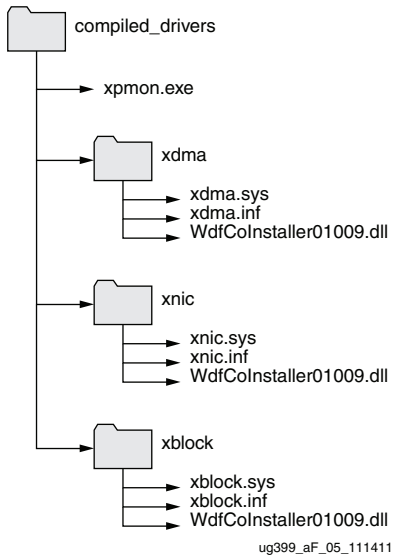


Figure F-5: **Directory Structure**

Uninstall Previous Drivers

If previous drivers are installed in the system, they must be uninstalled. Follow these steps:

1. Run **`x_s6_trd_setup.exe`**. The Confirm Uninstall dialog box opens (Figure F-6). Click **OK**.

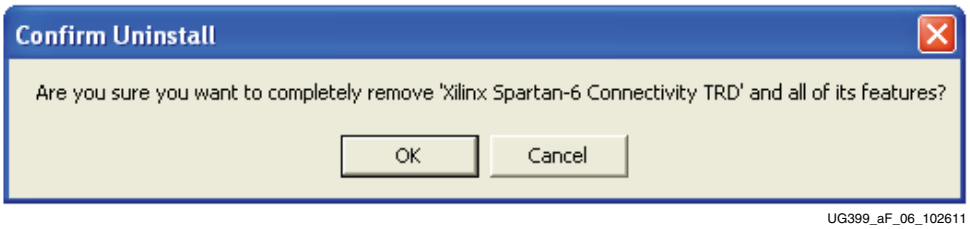


Figure F-6: **Confirm Uninstall**

2. When the uninstallation is completed, Figure F-7 opens. Click **Finish**.

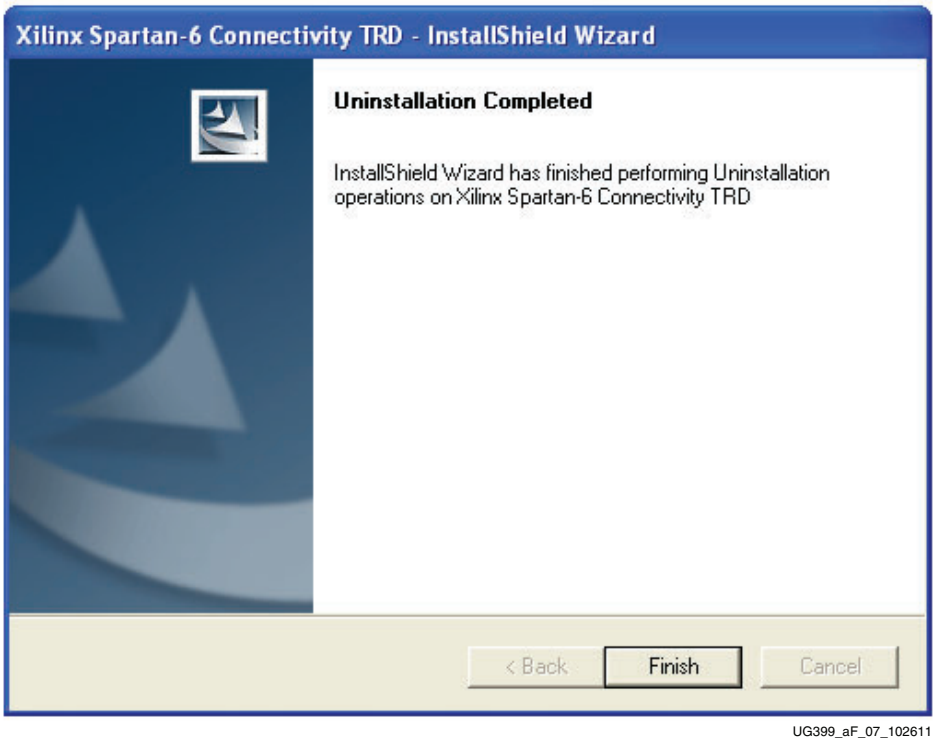


Figure F-7: **Uninstallation Completed**

Install Recompiled Drivers

After previous drivers are uninstalled, follow these steps to install the recompiled drivers:

1. Run `x_s6_trd_setup.exe`. When the InstallShield Wizard dialog box opens (Figure F-8), click **Next**.

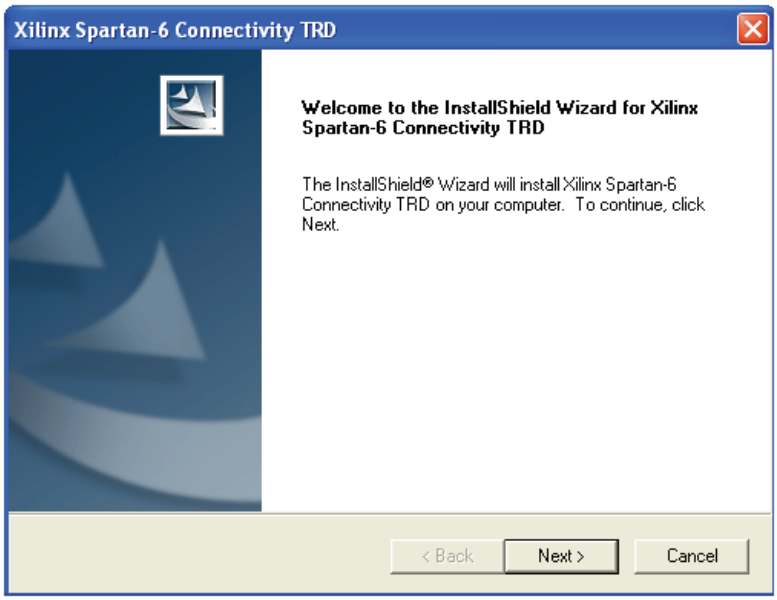


Figure F-8: **InstallShield Wizard**

2. When the Setup Type dialog box opens, select **Typical** (Figure F-9). This option installs the GUI and driver files and directories to the `C:\Program Files` directory. Click **Next**.

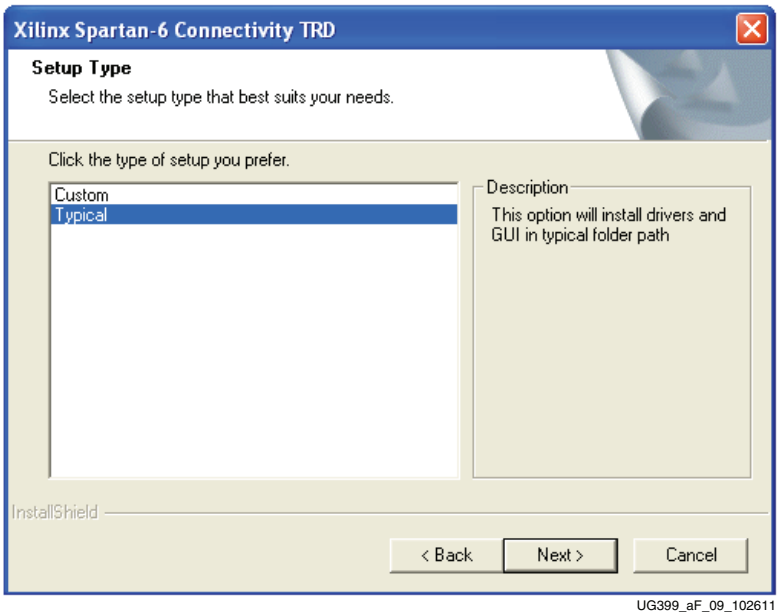


Figure F-9: **Typical Setup Selected**

3. When the XNIC driver selection dialog box opens (Figure F-10), select the driver type that matches the MAC to PHY interface used by the SP605 board. For example, if the bitfile programs the SP605 board to use GMII, select **GMII.** Click **Next.**

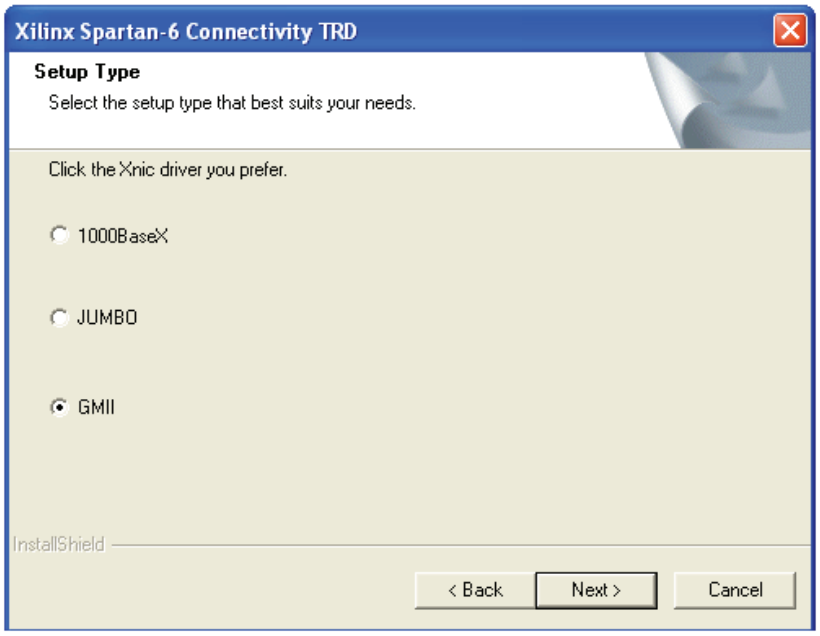


Figure F-10: **XNIC Driver Selection**

4. The current settings show the destination directory where the program files will be located (Figure F-11).

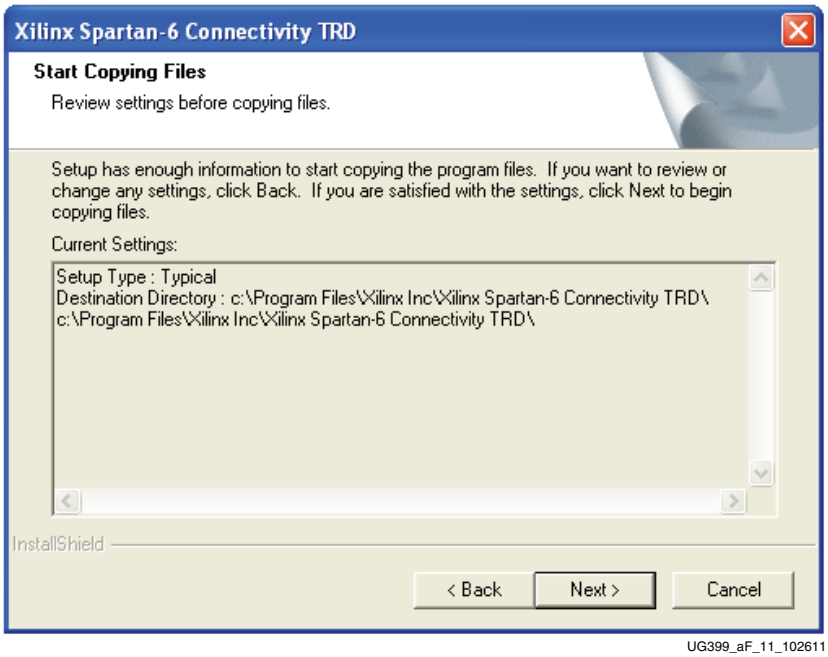


Figure F-11: **Current Settings for Setup**

5. If the dialog box shown in Figure F-12 opens, click **Continue Anyway**.

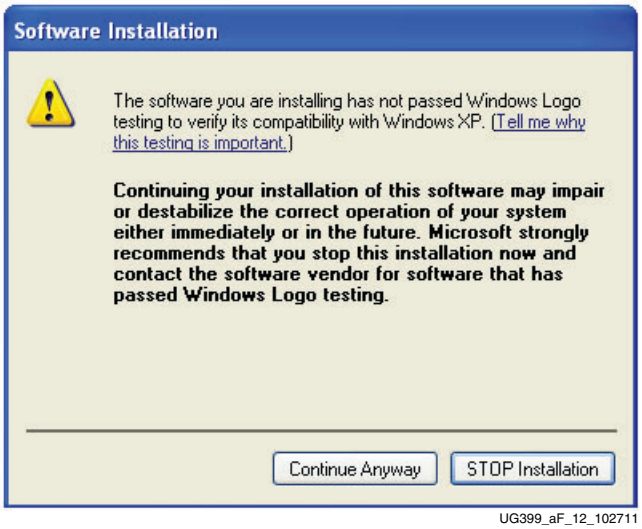


Figure F-12: **Software Installation Dialog Box**

6. When the InstallShield Wizard Complete dialog box opens (Figure F-13), click **Finish**.

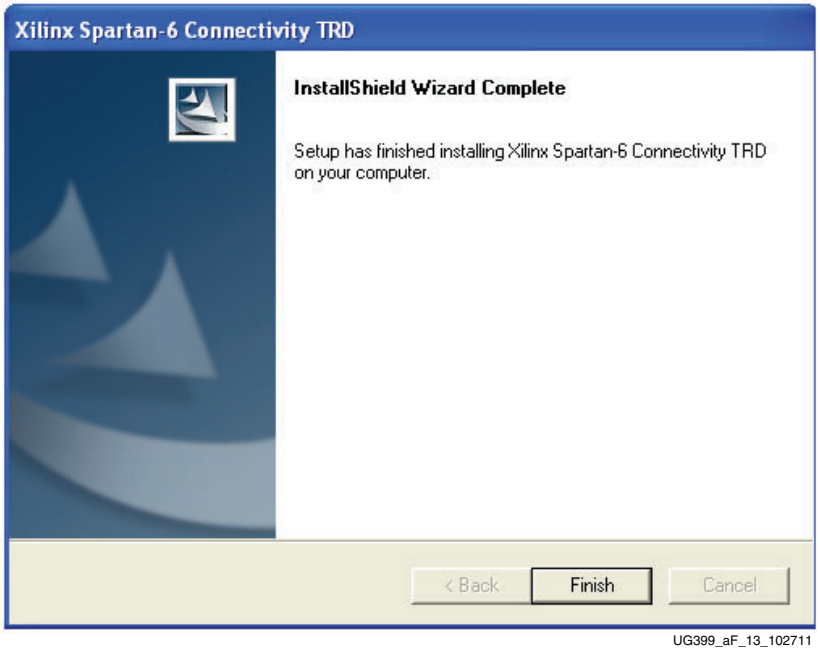


Figure F-13: **Installation Complete**

Install XDMA Driver

1. After the drivers are copied to the system, the Add Hardware Wizard opens (Figure F-14) for installation. Click **Next**.

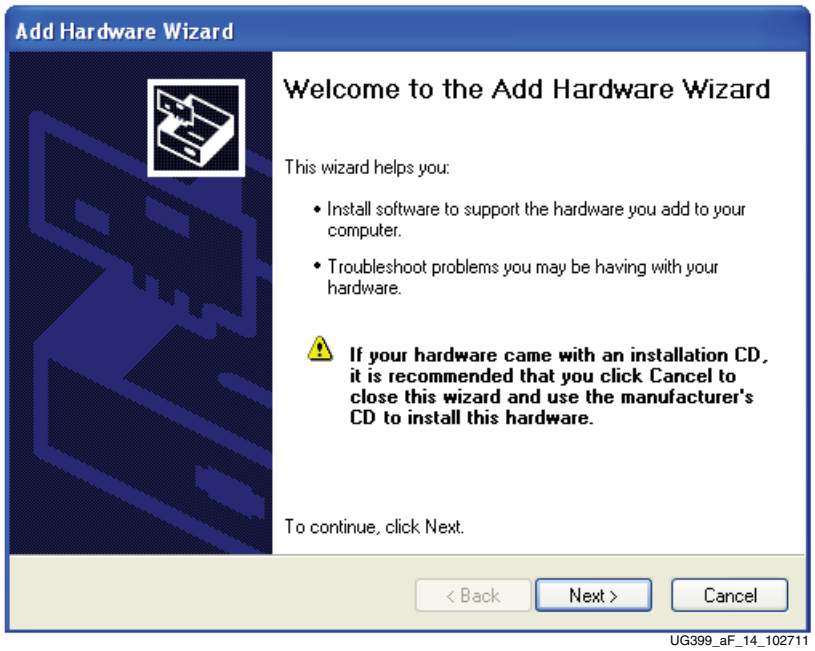


Figure F-14: **Add new Hardware Wizard**

2. To load the XDMA, drivers, select **Install from a list or specific location** and click **Next** (Figure F-15). This will enable installation of user-compiled drivers instead of the default drivers provided by the setup.

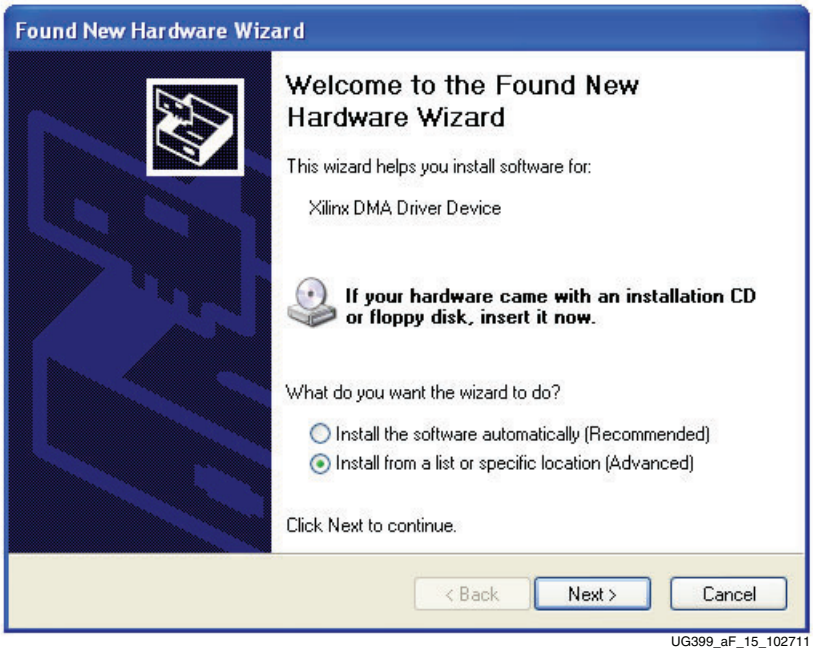


Figure F-15: **Specify Driver Location**

3. Browse to `<Path>\compiled_drivers\xdma`, and click **Next** (Figure F-16).

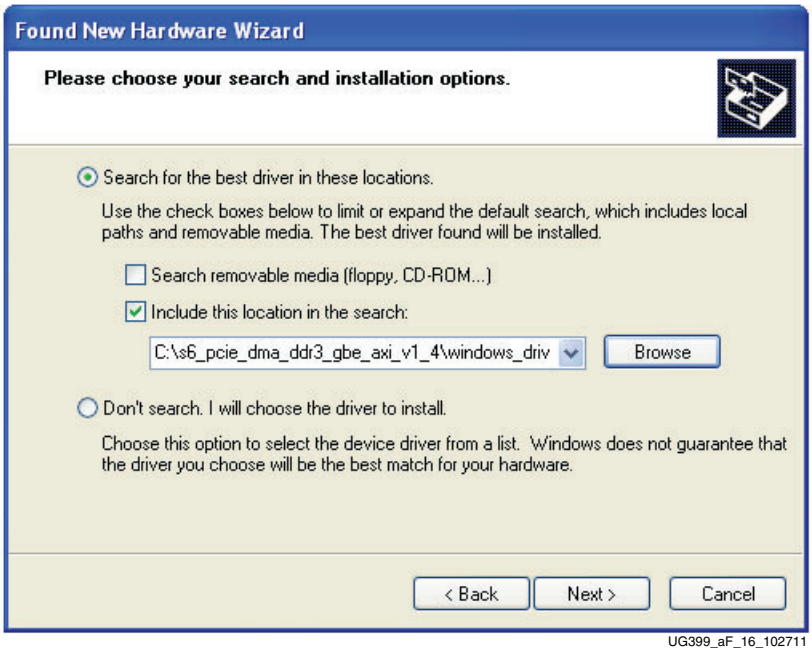


Figure F-16: **Search Path for XDMA Drivers**

4. When the XDMA driver is installed Figure F-17 opens. Click **Finish**.

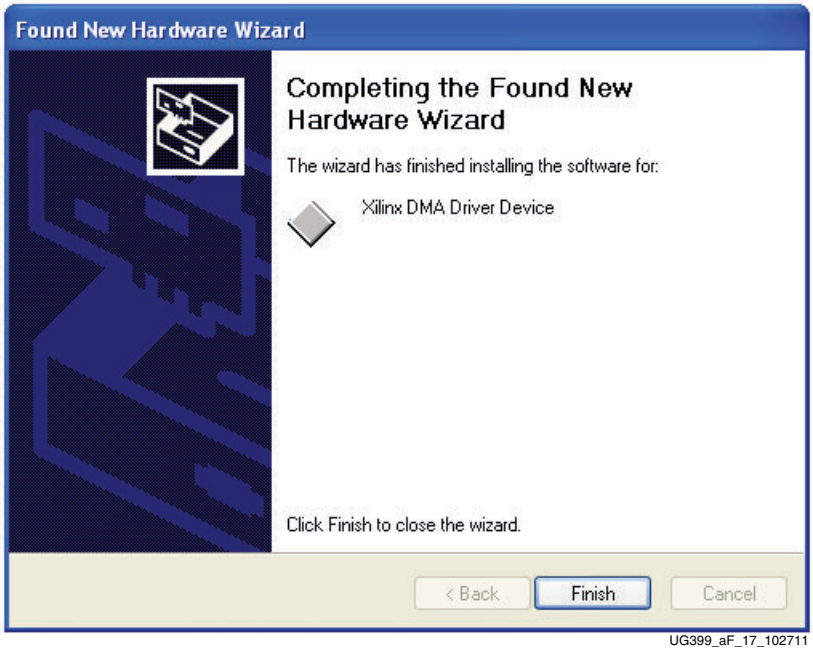


Figure F-17: **XDMA Driver Installation Complete**

Install XNIC Driver

1. Figure F-18 indicates the Add Hardware Wizard detects the ethernet adapter for which the XNIC driver must be installed. Select **Install from a list or specific location** and click **Next**.

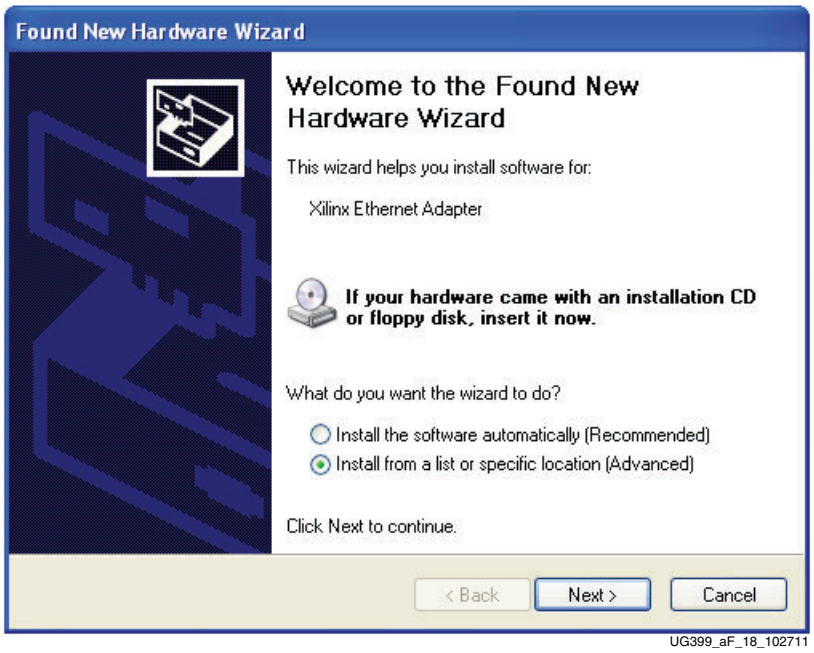


Figure F-18: **Add New Hardware Wizard for XNIC Driver**

2. Browse to `<path>\compiled_drivers\xnic`, and click **Next** (Figure F-16).

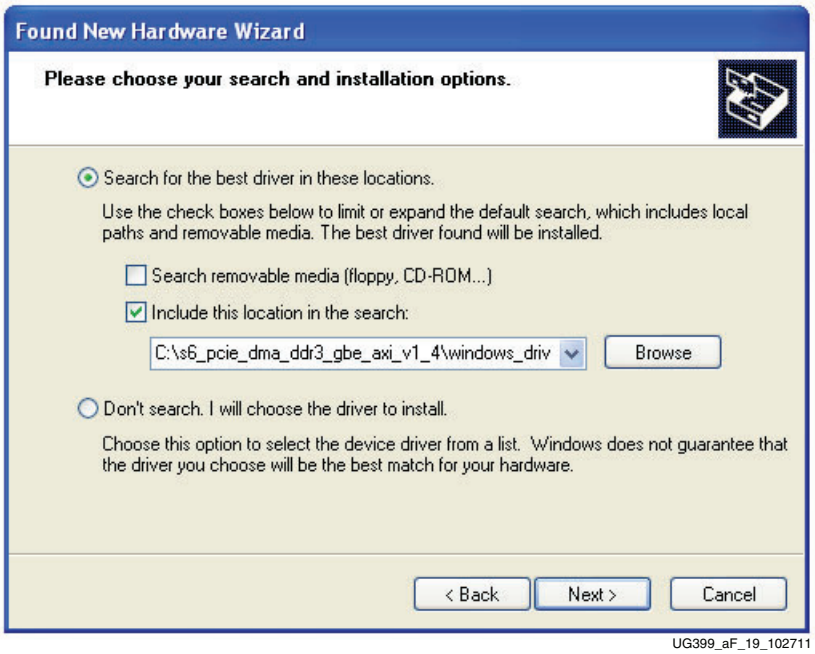


Figure F-19: **Search Path for XNIC Drivers**

3. If the dialog box shown in Figure F-20 opens, click **Continue Anyway**.

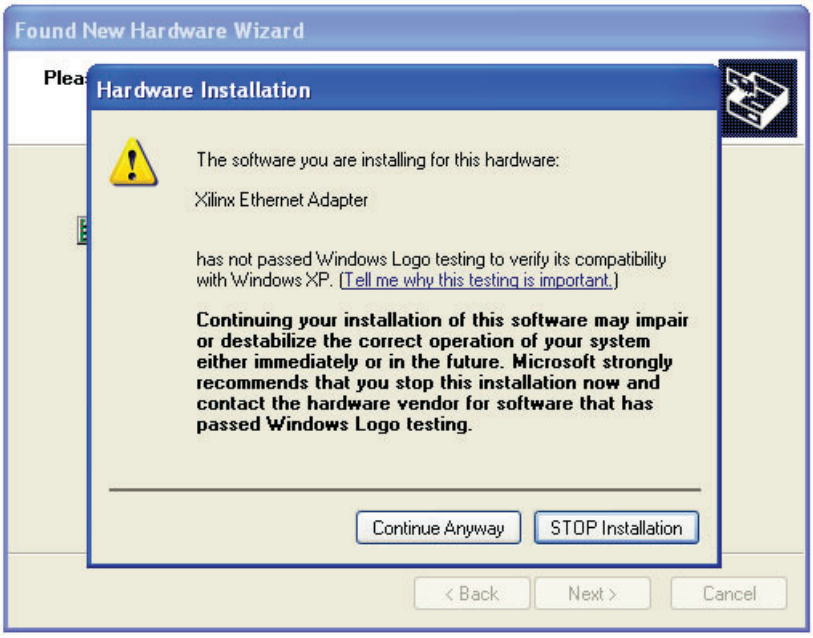


Figure F-20: **Hardware Installation Dialog Box**

4. When the XNIC driver is installed Figure F-21 opens. Click **Finish**.

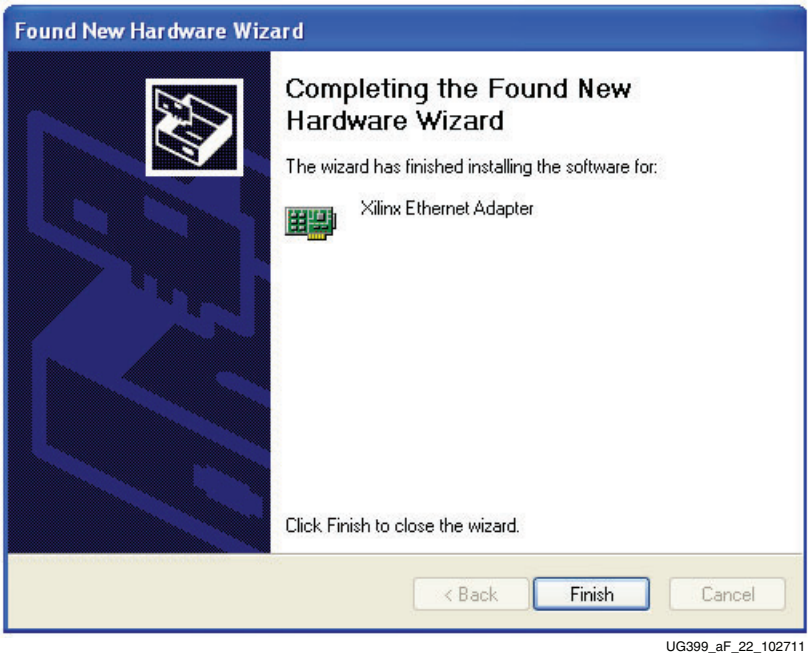


Figure F-21: **XNIC Driver Installation Complete**

Install XBLOCK Driver

1. Figure F-22 indicates the Add Hardware Wizard detects the block device for which XBLOCK driver must be installed. Select **Install from a list or specific location** and click **Next**.

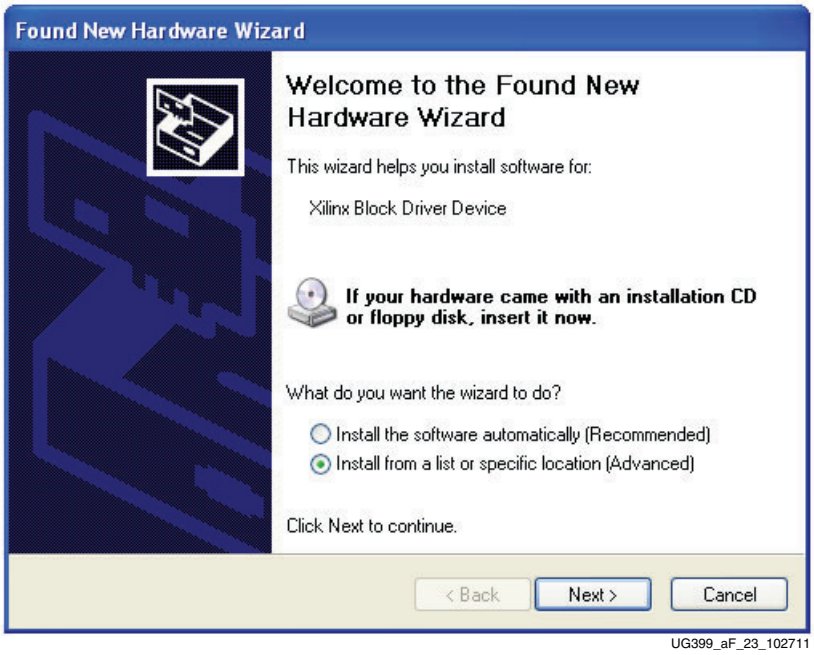


Figure F-22: **Add New Hardware Wizard for XBLOCK Driver**

2. Browse to `<path>\compiled_drivers\xblock`, and click **Next** (Figure F-23).

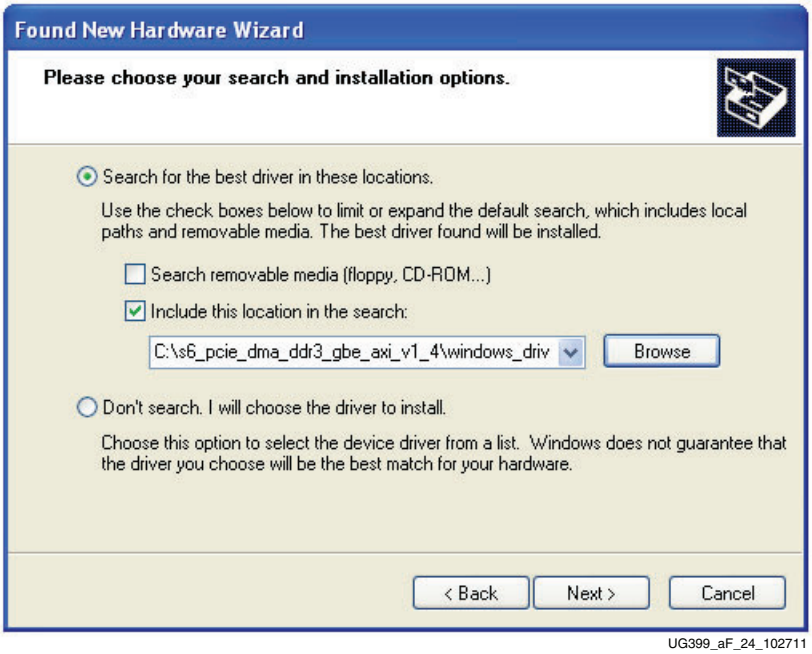


Figure F-23: **Search Path for XBLOCK Drivers**

3. When the XBLOCK driver is installed Figure F-24 opens. Click **Finish**.

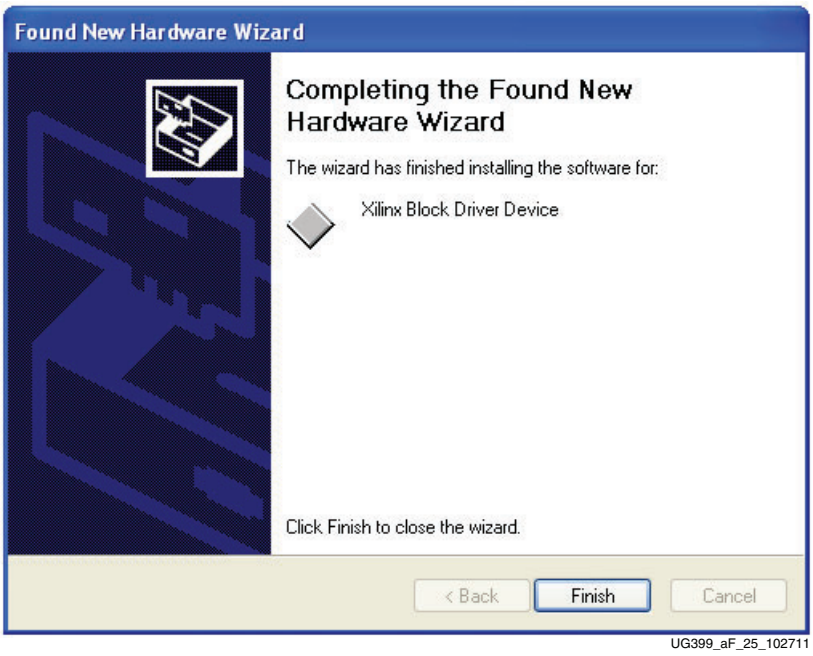


Figure F-24: **Xilinx Block Data Driver Installation Completion Window**

Launch Performance Monitor

1. To launch the Performance Monitor application, double-click the xpmon executable `xpmon.exe` in the `compiled_drivers` directory. See Using Application GUI, page 29 to run the application GUI.