ZC706 PCIe Targeted Reference Design (ISE Design Suite 14.4)

User Guide

UG963 (v2.0) January 8, 2013

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/15/12	1.0	Initial Xilinx release.
01/08/13	2.0	Updated for ISE Design Suite v14.4.

XILINX

Table of Contents

Chapter C: Troubleshooting

Appendix D: Additional Resources

Chapter 1

Introduction

This chapter introduces the Zynq™-7000 PCIe® Targeted Reference Design (TRD), summarizes its modes of operation and lists the TRD features.

The overall design is a video processing card that demonstrates these capabilities:

- PCIe connectivity:
  - Use of the Zynq-7000 XC7Z045 AP SoC PCIe Endpoint block in x4 Gen2 configuration
  - PCIe bus-compatible high performance low latency multichannel DMA
  - Performance demonstration using a traffic generator and checker running in Programmable logic (PL) and host software containing a PCIe root port
- Cortex-A9 processing and offload:
  - The use of a XC7Z045 AP SoC to offload and process video data
  - The use of a Sobel filter in PL
  - HDMI-based display controller
  - Cortex-A9 multiprocessing core in the XC7Z045 AP SoC used as a video data coprocessor
  - An example design showing independent memory management in the host system and the Zynq-7000 Processing System (PS)

The Zynq-7000 PCIe TRD demonstrates the operation of:

- PCIe Endpoint block (x4 Gen2)
- High-speed GTX transceivers
- High-speed multichannel DMA interface to a PCIe Endpoint
- Zynq-7000 PS
- Video DMA (VDMA) and Sobel filtering
- HDMI-based display controller

Figure 1-1 depicts the block diagram of the Zynq-7000 PCIe TRD.

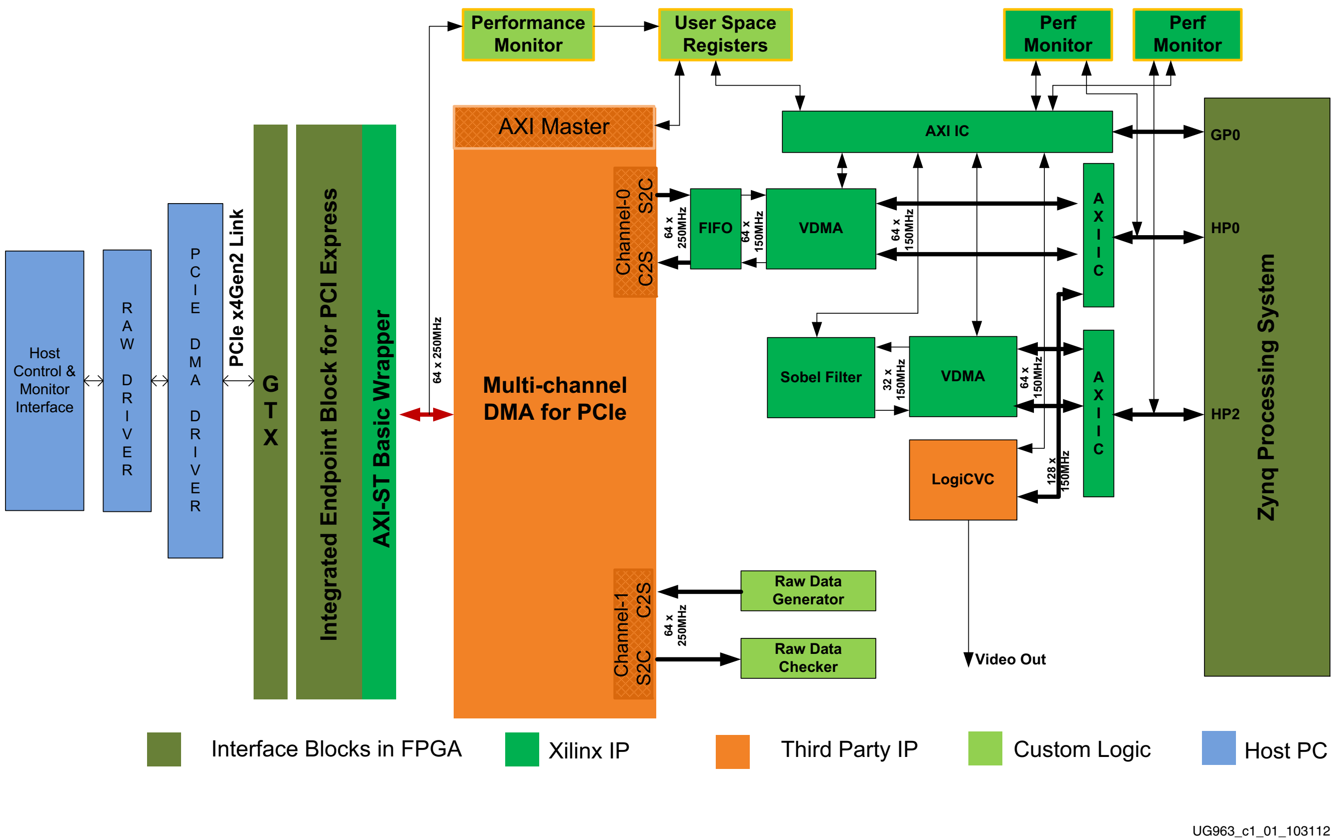


Figure 1-1: **Zynq-7000 PCIe Targeted Reference Design Block Diagram**

The Zynq-7000 PCIe TRD expands the Zynq-7000 Base Targeted Reference Design described in UG925, *Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design* by adding PCI Express® communication with a host system communicating at x4 Gen2 speed. In the Zynq-7000 Base TRD, the video processing pipeline input is provided by a test pattern generator in the PL. In the Zynq-7000 PCIe TRD, the video processing pipeline input is provided by an application running on the host computer at 1080p60 resolution and transmitted to the ZC706 board over the PCIe bus. The data is processed by video pipeline and passed back to the host system over the PCIe bus.

Because the full 1080p60 video stream uses about 4 Gb/s bandwidth, an additional data generator and a checker are implemented and connected to channel 1 of PCIe DMA to demonstrate the maximum x4 Gen2 bandwidth achieved by the hardware.

TRD Components

The Zynq-7000 PCIe TRD features these components:

- PCI Express v2.1 compliant x4 Endpoint block operating at 5Gb/s per lane, both directions:
    - PCIe transaction interface utilization engine
    - MSI and legacy interrupt support
- Bus mastering scatter-gather PCIe DMA to offload host processor:
    - Multi-channel DMA
    - AXI4 streaming interface for data
    - AXI4 interface for register space access
    - DMA performance engine
    - Full duplex operation:
        - Independent transmit and receive channels
- Multichannel video DMA with programmable VSIZE and HSIZE:
    - AXI4 compliant
    - Optional flush on frame sync
    - Optional frame advancement on error
- Multilayer display controller:
    - Alpha blending, transparency and move-around support
    - Continuous switching mode support
- Sobel filter:
    - AXI4 stream interface
    - AXI4 control interface
    - Supports image resolution up to 1080p
- Java-based GUI running on the PCIe host system:
    - Test control panel
    - PCIe performance monitoring
- A Qt-based GUI running in the PS:
    - Monitors power and die temperature
    - PS HP0 and HP2 performance numbers
    - CPU utilization

Table 1-1 lists some terminology used in the document.

Table 1-1: **Terminology**

Design Block Used in the TRD	Name
PCIe DMA	NWL DMA
Video DMA interfacing to the FIFO in the NWL DMA, channel 0	SOURCE VDMA
Video DMA interfacing to the Sobel filter	FILTER VDMA
Host system with PCIe slot	PCIe Host system
Control PC to monitor UART output	Control PC

Data Flow

The Zynq-7000 PCIe TRD demonstrates the use of the XC7Z045 AP SoC to offload video processing tasks from a PCIe host system.

Video Processing and Offload Demonstration on PCIe DMA Channel 0

The user application in the PCIe host system generates 1920 x 1080 pixel video frames containing 8 color bars. The PCIe host system software manages channel 0 of the PCIe DMA to transmit the video stream over a x4 Gen2 PCIe link to the ZC706 board. A PCIe DMA translates the stream of PCIe video data packets into AXI streaming data, which is connected to a Video DMA (VDMA). Software running in the PS on the Cortex-A9 processor manages the AXI VDMA and transfers the raw video frames into the PS DDR3 memory. The hardware-based Sobel filter reads the image using another VDMA and performs edge detection on the raw image and sends the data back to the PS DDR3. The processed data in PS DDR3 can either be transferred back to the host system using channel 0 of the card-to-system (C2S) interface of the PCIe DMA or be displayed on the monitor using the LogiCVC display controller. Due to the limitation of the PS DDR3 bandwidth, the same data cannot be displayed and sent back to the host system simultaneously.

Generator and Checker Demonstration on PCIe DMA Channel 1

A generator and checker on channel 1 of the PCIe DMA allows the RX and TX paths to run independently. The hardware generator in the PL generates data packets with an incremental sequence pattern. The PCIe host system software checker verifies the incremental sequence pattern generated by the hardware generator. Independently, the PCIe host system driver generates a stream of incremental data which is transferred via the PCIe link by the NWL PCIe DMA to the checker implemented in the PL.

Resource Utilization

Table 1-2: **Resource Utilization**

Resource	Total Available	Usage
Slice Registers	437,200	49,425 (11%)
Slice LUT	218,600	38,502 (17%)
RAMB36E1	545	61 (11%)
MMCME2_ADV	8	2 (25%)
BUFG/BUFGCTRL	32	10 (31%)
XADC	1	1 (100%)
IOB	362	26 (7%)
GTXE2_CHANNEL	16	4 (25%)
GTXE2_COMMON	4	1 (25%)

Getting Started

This chapter describes how to set up the ZC706 board, control computer, host computer and software for the Zynq™-7000 PCIe® Targeted Reference Design.

Required Materials

ZC706 Evaluation Kit Contents

Material supplied with the ZC706 Evaluation Kit includes:

- ZC706 board with the Zynq-7000 XC7Z045-2 FFG900C AP SoC
- ISE Design Suite: System Edition (full seat, device-locked to the XC7Z045-2 FFG900C AP SoC)
- USB cable, standard-A plug to mini-B plug
- USB cable, standard-A plug to micro-B plug
- HDMI cable, type-A plug to type-A plug
- Ethernet Cable
- Power Supply: 100 VAC–240 VAC input, 12 VDC 5.0A output
- Power cords to support three mains plug types
- ATX adapter cable, Xilinx part number 2600304.
- SD Card containing the reference design files.
- Fedora 16 LiveCD

Additional Materials

User-supplied materials include:

- Monitor supporting 1080p
- Two PC computers (see Computer Requirements)
- USB mouse (for use with the ZC706 board)

Computer Requirements

Running the Zynq-7000 PCIe TRD requires two PC computers.

Control PC Computer

An Intel-processor-based laptop or desktop PC running Windows 7 operating system. The computer must an SD memory card receptacle, and one USB port to communicate with the ZC706 board.

Required Software

The software listed here must be installed on the control computer:

- ISE Design Suite: System Edition.
- Communications drivers and terminal program. See Appendix A, Setting Up Board Communications for details.

PCIe Host System Computer

An Intel-processor-based desktop PC running Fedora Core 16 Linux operating system is required for the PCIe host system. The computer must have a PCIe v2.0 slot where the ZC706 board is installed in the open chassis of this computer.

Recommended Motherboards

Recommended PCI Express Gen2 PC system motherboards are:

- Sandy Bridge Motherboard
- Ivy Bridge Motherboard

Programming the ZC706 Board

The XC7Z045 AP Soc is configured from a bitstream in a 2 x 128 Mb Quad-SPI flash memory. This bitstream must first be loaded in the Quad-SPI flash memory from the SD Card plugged into J30 on the ZC706 board.

Files for configuring the Zynq-7000 PCIe TRD are compiled in `zc706_pcie_trd.bin` which contains the `zynq_fsbl.elf` bitstream and `u-boot.elf` bitstream along with the Linux kernel, Linux file system image files and Linux device tree binary files.

`zynq_pcie_trd_14_3.zip`

Note: This document refers to the initial release of the Zynq-7000 PCIe TRD, which is for ICE Design Suite v14.3. User should use the appropriate latest version of the TRD which is available at the The ZC706 Evaluation Kit Kit product page: http://www.xilinx.com/products/boards/zc706.

Extracting the Project Files

The Zynq-7000 PCIe Targeted Reference Design files are located in `zynq_pcie_trd_14_3.zip` This file is available for download online at:

http://www.xilinx.com/zc706 (listed under the **Docs & Designs** tab)

To extract the files:

1. Download `zynq_pcie_trd_14_3.zip` to a working directory on the control computer.
2. Unzip the files contained in `zynq_pcie_trd_14_3.zip`.

Programming the SD Card

On the control computer:

1. Plug the SD card into the SD card receptacle.
2. Navigate to the working directory `zynq_pcie_trd_14_3/prog_qspi` and copy `BOOT.bin`, `zc706_pcie_trd.bin`, `devicetree.dtb`, `devicetree_qspi.dtb`, `uramdisk.image.gz`, `uImage`, `init.sh` to the SD card.

   The `BOOT.bin` file enables the PS to boot in the SD boot mode. The `zc706_pcie_trd.bin` file contains the TRD bitstream. The remaining files are required for Linux boot-up.

3. Unmount and remove the SD Card from the computer and insert the it into the SD card receptacle on the ZC706 board (Figure 2-1).

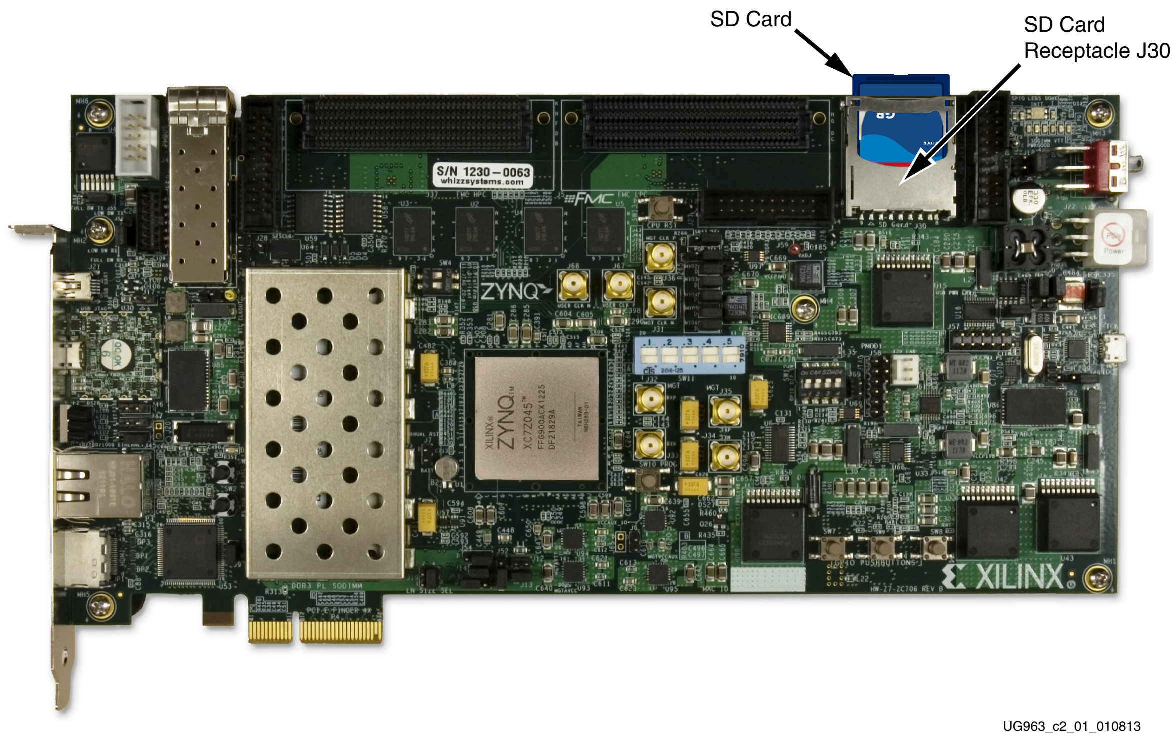


Figure 2-1: **SD Card Location**

Programming the Quad-SPI Flash Memory

This procedure programs the Quad-SPI flash memory with files from the SD card to run the Zynq-7000 PCIe TRD.

1. Complete the communications setup in Appendix A, Setting Up Board Communications.
2. Power off the ZC706 board (SW12).
3. Verify the SD Card is plugged into receptacle J30 as shown in Figure 2-1.
4. Connect the ZC706 board to the control computer and power supply as shown in Figure 2-2.

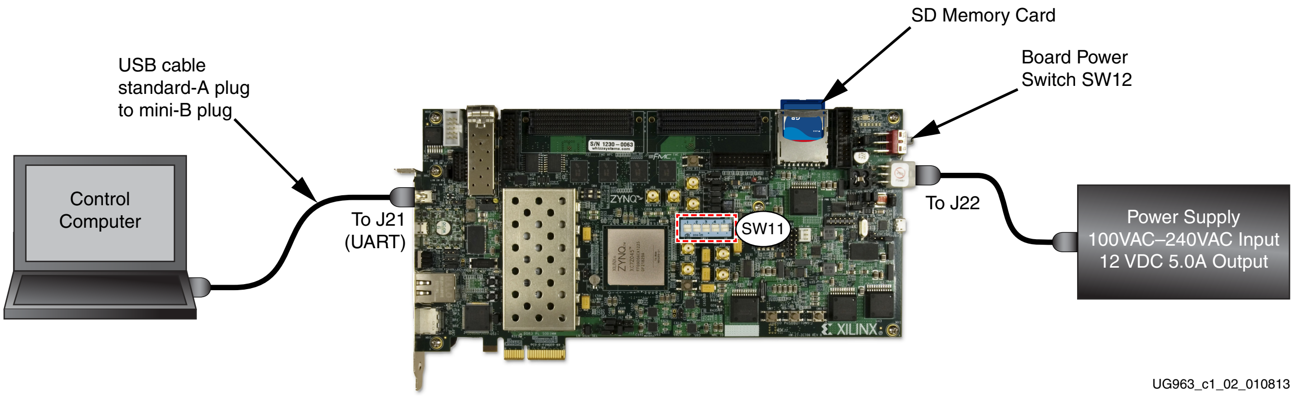


Figure 2-2: **ZC706 Board Programming Setup**

5. Set DIP switch SW11 as shown in Figure 2-3.

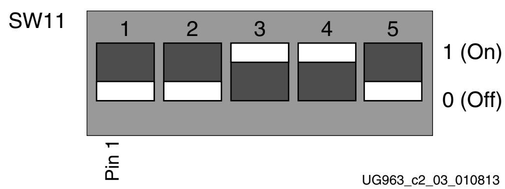


Figure 2-3: **SW11 SD Boot Mode Settings**

6. Power ON the control computer and start TeraTerm Pro using **115200** bits/s, **8** data bits, **None** parity, **1** stop bit, **None** flow control (see Figure A-3, page 68).
7. Power ON the ZC706 board (SW12). The `init.sh` script in the SD card loads the Quad-SPI flash memory with `zc706_pcie_trd.bin` and the Linux kernel images. Initialization progress is shown on the TeraTerm Pro display (Figure 2-4).

   Four commands are executed by `init.sh`:

   `zynq>`**flashcp -v zc706_pcie_trd.bin /dev/mtd0**

   `zynq>`**flashcp -v uImage /dev/mtd1**

   `zynq>`**flashcp -v devicetree_qspi.dtb /dev/mtd2**

   `zynq>`**flashcp -v uramdisk.image.gz /dev/mtd3**

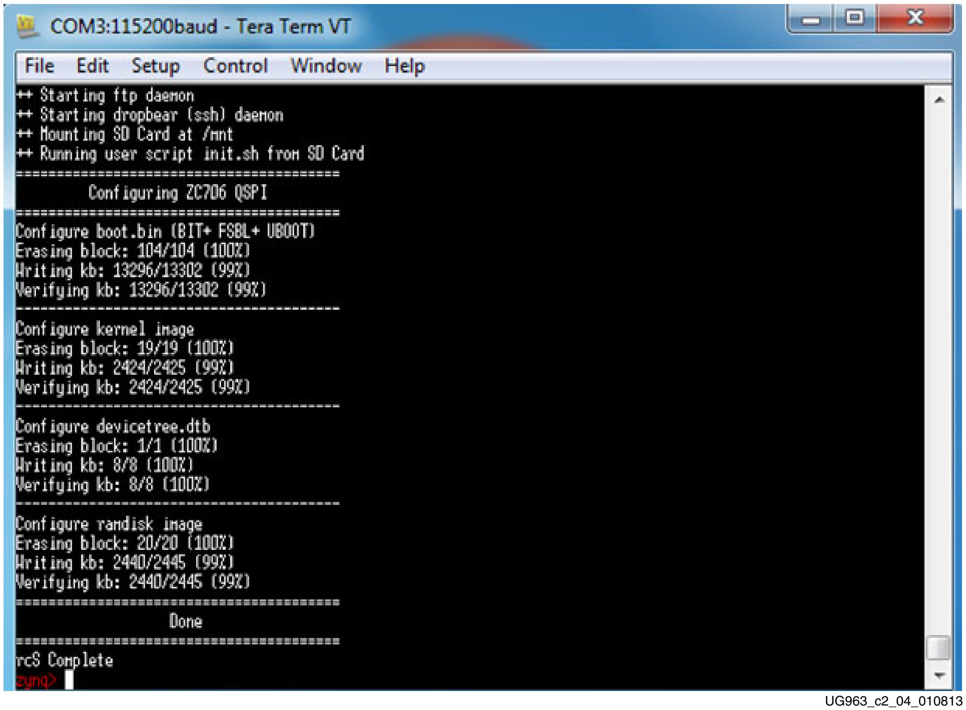


Figure 2-4: **Initialization Progress of the init.sh script**

8. Initialization is complete when the `zynq>` prompt appears on the TeraTerm Pro display.
9. Navigate to the `zynq_pcie_trd_14_3/sd_image` folder and copy `qt_lib.img`, `init.sh`, `zynq_pcie_qt` and `zynq_pcie_qt.sh` files to the SD card used to program the QSPI device. These files are required for loading the Zynq-7000 PCIe TRD. Insert the SD card into the SD card slot.
10. Set DIP switch SW11 as shown in Figure 2-5 for QSPI boot mode.

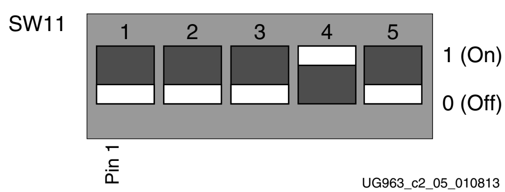


Figure 2-5: **SW11 Quad SPI Flash Memory Settings**

TRD Demonstration Setup

This section describes hardware bring-up, software bring-up, and using the application GUI.

Installing ZC706 Board in the Host Computer Chassis

When the ZC706 board is used inside a computer chassis power is provided from the ATX power supply peripheral connector through the ATX adapter cable shown in Figure 2-6.

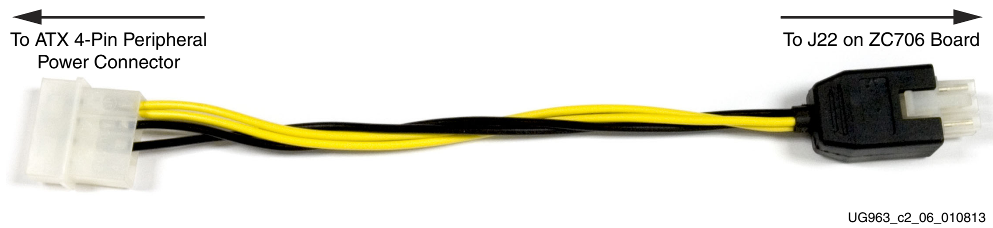


Figure 2-6: **ATX Power Supply Adapter Cable**

CAUTION! *Do* ***NOT*** *plug the PC ATX power supply 6-pin connector into J22 on the ZC706 Evaluation Board. This connector has a different pinout than J22 and is not compatible. Connecting the ATX power supply 6-pin connector into J22 will damage the ZC706 Evaluation Board.*

To install the ZC706 board in a computer chassis:

1. Remove all six rubber feet and standoffs from the ZC706 board.
2. Power down the host computer and remove the computer power cord.
3. Open the chassis, select a vacant PCIe expansion slot and remove the expansion cover at the back of the chassis.
4. Plug the ZC706 board into the PCIe connector at this slot.
5. Install the top mounting bracket screw into the PC expansion cover retainer bracket to secure the ZC706 board in its slot.

IMPORTANT: *The ZC706 board is taller than standard PCIe cards. Ensure that the height of the card is free of obstructions.*

6. Connect the ATX power supply to the ZC706 board using the ATX power supply adapter cable as shown in Figure 2-7:
7. Slide the ZC706 board power switch SW12 to the ON position.

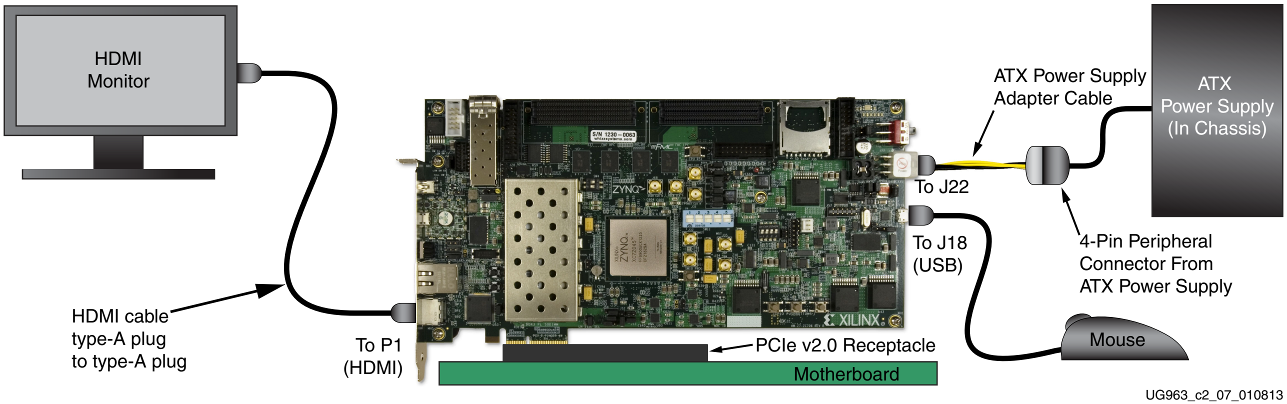


Figure 2-7: **ZC706 Board TRD Setup in Host Computer**

Installing Device Drivers

Host Computer Bootup

The procedures listed in this section requires Linux super user access on the host computer. When using the Fedora 16 LiveCD, super user access is granted by default. if not using the Fedora 16 LiveCD, contact your system administrator for super user access.

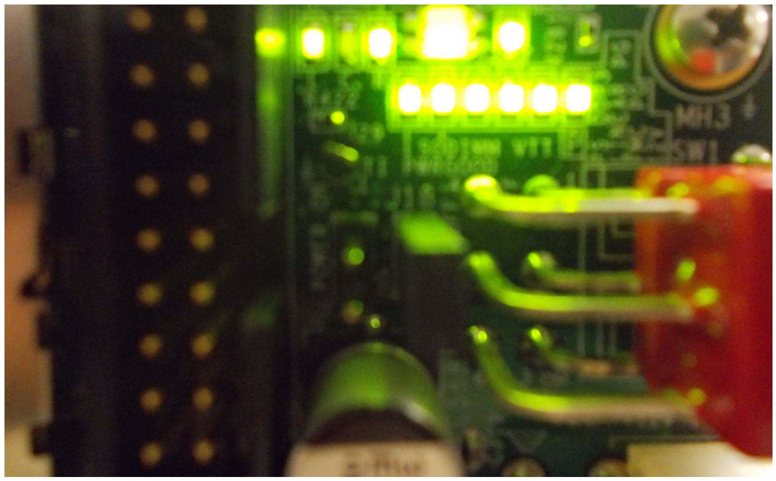
If Fedora 16 is installed on the host computer hard disk, boot as a root-privileged user.

If Linux is not installed, place the Fedora 16 Live DVD in the host computer CD-ROM drive and restart the computer.

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

IMPORTANT: The BIOS boot order must be set so that the CD-ROM drive is the first drive in the boot order. To set the boot order, Power on the computer and press the **DEL** or **F2** key. Set the boot order and save the change.

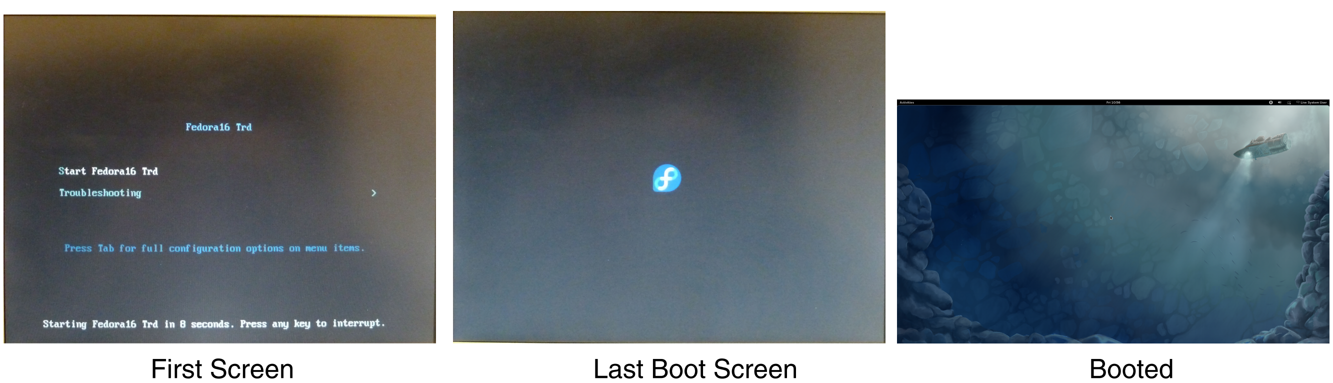
1. Switch SW12 on ZC706 board to the ON position. Power on the PCIe host system. The Zynq-7000 PCIe TRD provides PCIe status on the GPIO LEDs near the ZC706 board power switch. LED R and L should be ON and LED C should be OFF. They are as follows: LED L PCIe link up, LED C user reset from PCIe IP and LED R user clock heartbeat LED.



UG963_c2_08_121712

Figure 2-8: **LED Position on the ZC706 Board**

The images in Figure 2-9 are seen on the monitor during boot up. On the HDMI monitor connected to the ZC706 board a Qt-based application appears that shows device power and temperature.



UG963_c2_09_010813

Figure 2-9: **Fedora 16 LiveCD Boot Sequence**

2. Download `zynq_pcie_trd_14_3.zip` from www.xilinx.com/zc706 and copy it to specific `/tmp` folder of PCIe host PC. Change permission by typing **`chmod 755 -R zynq_pcie_trd_14_3`** on a terminal so that files inside `zynq_pcie_trd_14_3` has execution permission. Double-click on the copied `zynq_pcie_trd_14_3`. The screen capture in Figure 2-10 shows the content of `zynq_pcie_trd_14_3` folder.

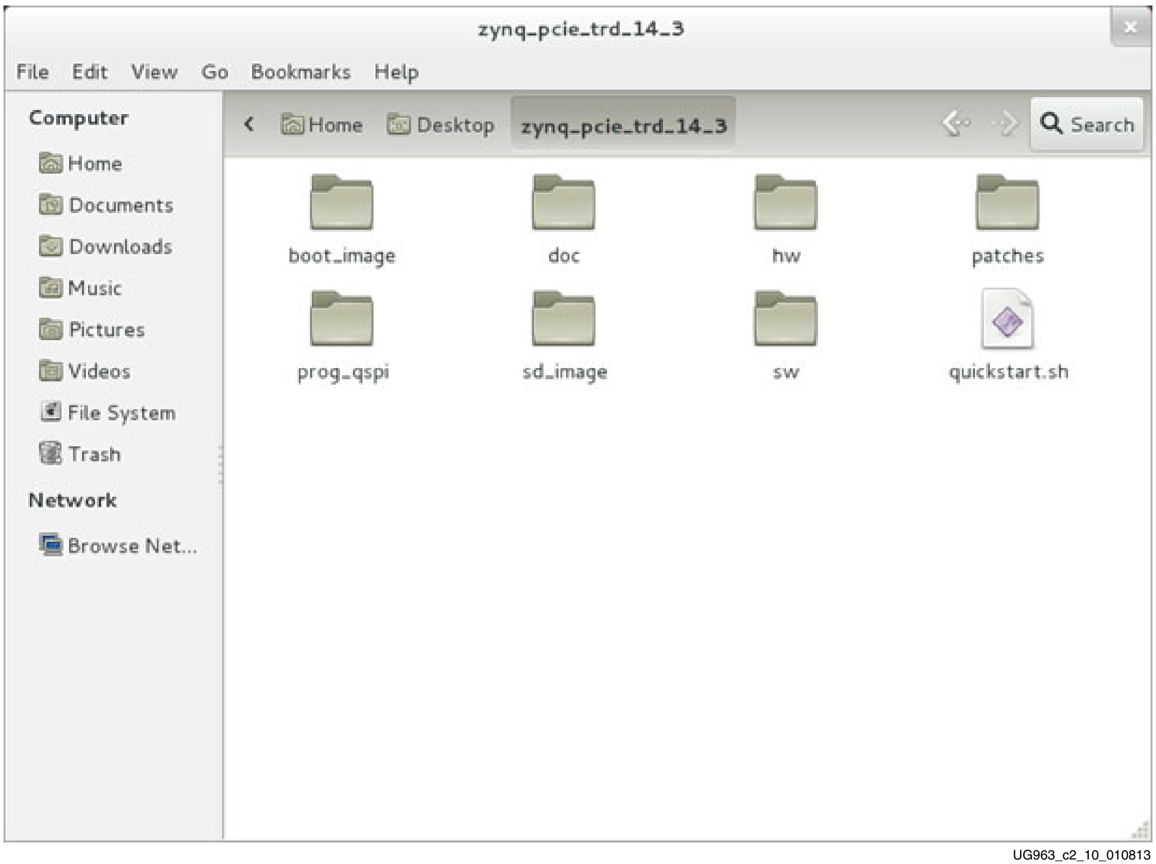


Figure 2-10: **Directory Structure of z7_pcie_trd**

3. Double click on the `quickstart.sh` script. This script sets proper permission and invokes the driver installation GUI. Click **Run in Terminal**.

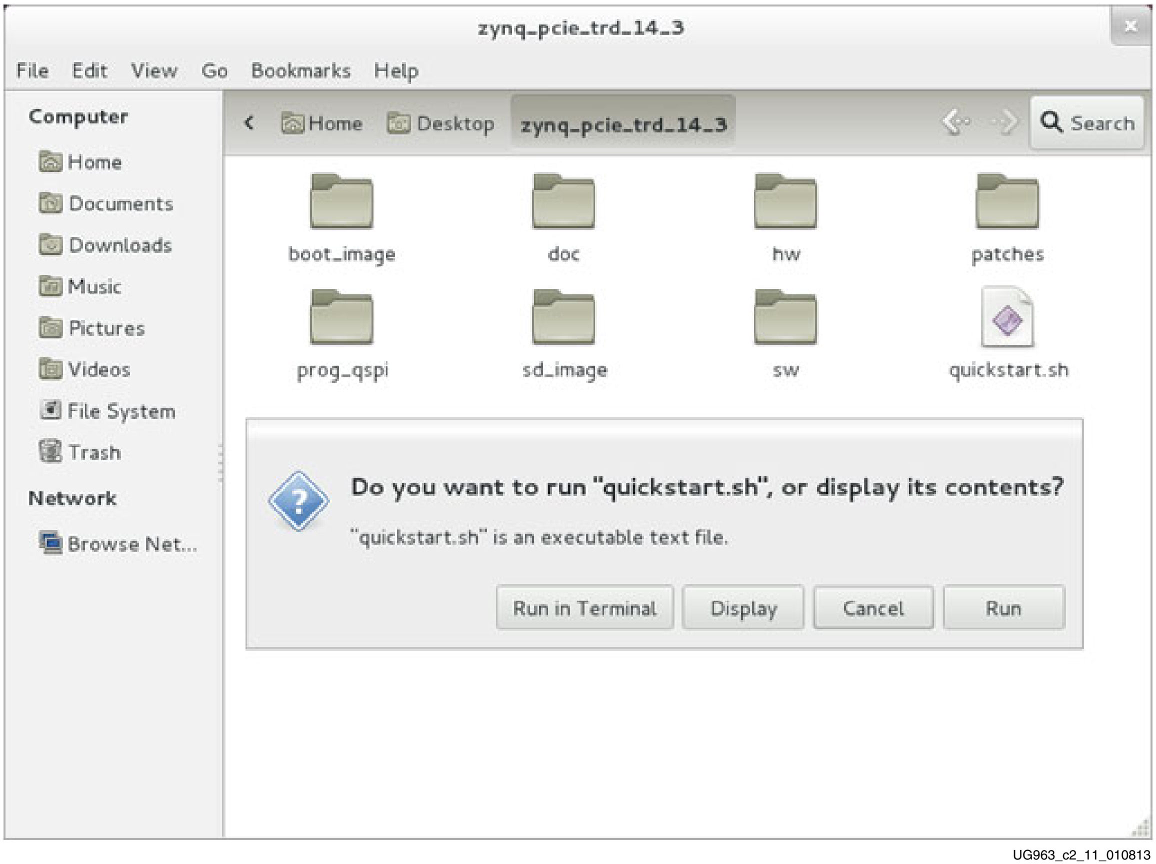


Figure 2-11: **Running Quickstart Script**

5. The GUI with driver installation option pops up as shown in Figure 2-12. The step installs all the software necessary for the host system to control, generate and receive PCIe

traffic to and from the ZC706 board and monitor performance. Select Video/Raw performance mode driver mode and click **Install**.

Figure 2-12: **Landing Page of Zynq-7000 PCIe TRD**

7. After installing the Video/Raw performance mode driver, the control and monitor user interface pops up as shown in Figure 2-13. The control pane shows control parameters such as Sobel Filter and Video Out selection modes.

UG963_c2_13_010813

Figure 2-13: **Video/Raw Performance Mode**

8. Click `Start` button in the Video Path panel to start the PCIe host system generating a 1080p60 video stream and sending it over to ZC706 board via PCIe. The video stream is processed and displayed on HDMI monitor or sent back to the host via PCIe based on the test mode selected in Video Out menu. The Performance Plots tab shows the system-to-card (S2C) and card-to-system (C2S) PCIe performance numbers.

   User can select various test modes from the Sobel Filter drop-down menu:

   - Select option None to display the frames on monitor without sobel
   - Select option Sobel-HW to display the frames on monitor with HW sobel
   - Select option Sobel-SW to display the frames on monitor with SW sobel

   User can select various test modes from the Video Out drop-down menu:

   - Select option HDMI to display Sobel data on HDMI monitor
   - Select option PCIe Host to send data back to PCIe host system

   For option Sobel Filter: None and Video Out: HDMI, video data from PCIe host system is directly sent to the display without being processed by the edge detection Sobel filter.

A color bar pattern appears on the display as shown in Figure 2-14 for this option.

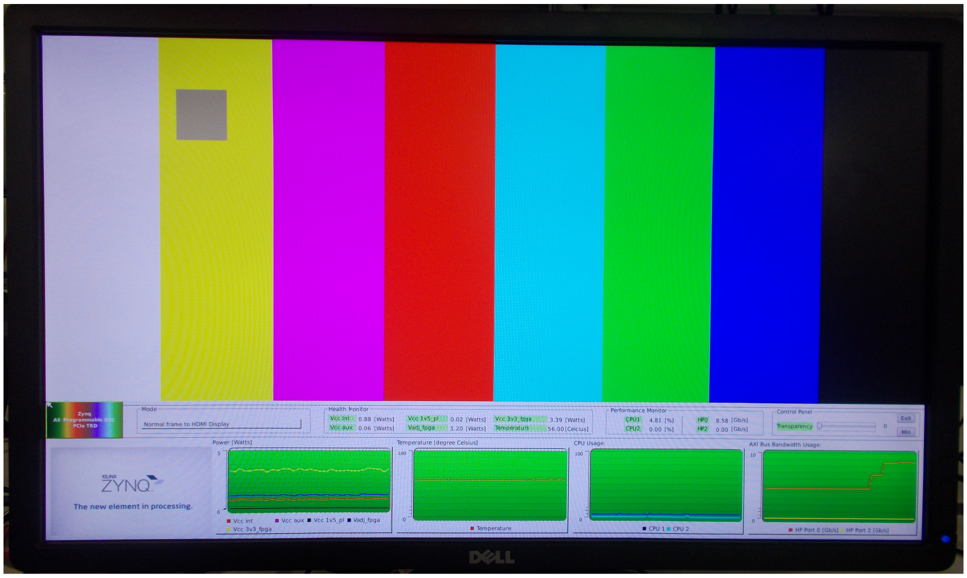


Figure 2-14: **HDMI Display for Color Bar Display**

9. For option Sobel Filter: Sobel-HW and Video Out: HDMI, video data from PCIe host system is directly processed by the edge detection Sobel filter in the PL then sent to the display. Edges of the color bar pattern appear on the display as shown in Figure 2-15 for this option.

Figure 2-15: **HDMI Display for Sobel Output Display**

For option Sobel Filter: Sobel-SW and Video Out: HDMI, video data from PCIe host system is directly processed by the edge detection Sobel filter in the PS then sent to the display. Edges of the color bar pattern appear on the display.

For option Video Out: PCIe Host, video data from PCIe host system is processed by Sobel filter in the PL or PS depending on mode selected in Sobel Filter, then sent back to the PCIe host system via PCIe. The data is not sent to the display.Sobel Filter:none is not a supported option when Video Out is set to PCIe Host.

The Qt GUI monitors the power of the device voltage rails and die temperature. The CPU utilization and PS HP port 0 and HP port 2 performance numbers are also periodically plotted. When the user selects **Sobel Filter: None** HP port 0 performance becomes 8 Gb/s and HP port 2 port performance becomes 0 Gb/s When the user selects Sobel Filter: Sobel-HW both HP port 0 and HP port 2 performance is close to 8 Gb/s. When user selects Sobel Filter: Sobel-SW, the CPU2 performance becomes 100%, HP port 0 performance becomes close to 8 Gb/s and HP port 2 performance becomes 0.

10. As noted in the discussion above, because a single HD stream of video data is insufficient to saturate available PCIe x4 Gen2 bandwidth, data path 1 can be turned on to add additional PCIe traffic. Click on Start button in the Data Path-1 panel to generate additional traffic. On this path the User can vary packet sizes and see performance variation accordingly. Total PCIe BW is updated in the PCIe statistics panel and the performance plot. User can select Loopback, HW Generator and HW Checker option in the GUI for Data Path-1.

Figure 2-16: **Video/Raw Performance Mode Plots**

11. User can click on the Block Diagram option to view the design block diagram as shown in Figure 2-17.

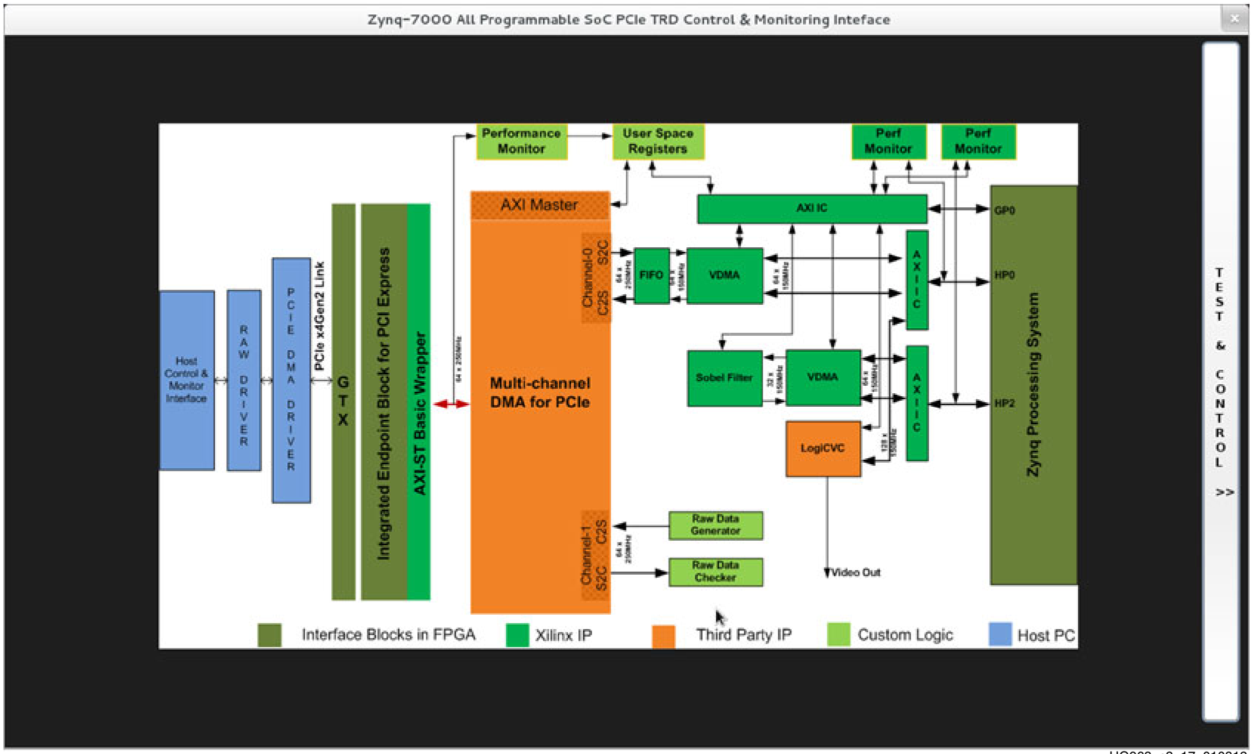


Figure 2-17: **Design Block Diagram**

12. User can exit the Qt GUI by clicking on the Exit button in the GUI as shown in Figure 2-18.

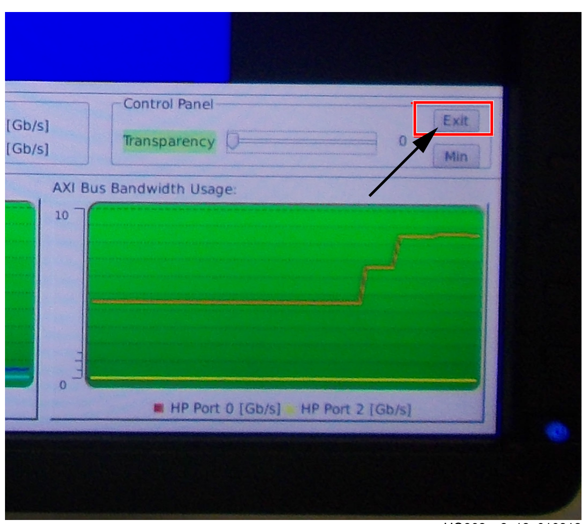


Figure 2-18: **Exiting the Qt GUI**

13. Close the GUI - this process uninstalls the driver on PCIe host system and opens the landing page of the Zynq-7000 PCIe TRD. Driver un-installation requires GUI to be closed first.

Rebuilding the Design

The designs can also be re-implemented using ISE software. Before running any command line scripts, refer to the “Platform Specific Instructions” section in ISE Design Suite: Installation and Licensing Guide (http://www.xilinx.com/support/documentation) to learn how to set the appropriate environment variables for the operating system. All scripts mentioned in this user guide assume the XILINX environment variables have been set.

TIP: The development computer does not have to be the hardware test machine with the PCIe slots used to run the Zynq-7000 PCIe TRD.

For building the HW design, please refer to wiki.xilinx.com/zynq-pcie-trd.

Functional Description

This chapter describes the hardware and software architecture.

Hardware Architecture

The hardware design architecture is described under two sections:

- Processing System (PS)
- Programmable Logic (PL)

The NWL DMA and the video pipe are implemented in the PL. The following sections detail the implementation of the PS and PL blocks.

Processing System

The Zynq-7000 PCIe TRD makes full use of four major components in the PS:

- Application Processor Unit
- Interconnect
- Input/Output Peripherals
- Memory Interfaces

This section describes some of the features of the PS used in this design.

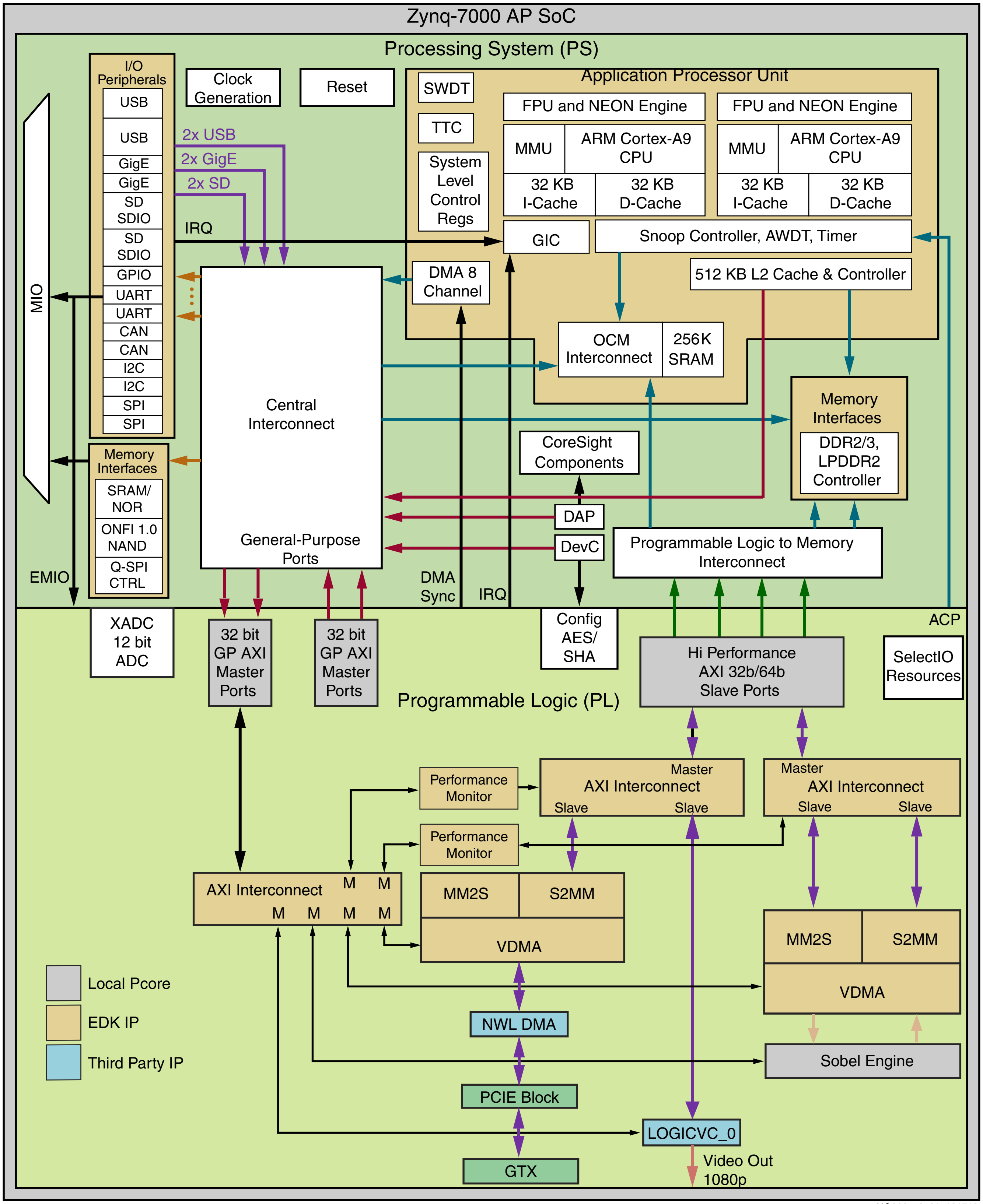


Figure 3-1: **Video Processing and Offloading Demo**

Application Processor Unit

The application processor unit (APU) includes the dual configuration of the ARM Cortex-A9 multiprocessor core, snoop control unit (SCU), level-2 cache controller, dual ported on-chip RAM (OCM), 8-channel DMA, system watchdog timer (SWDT), and triple timer controller (TTC) blocks.

Cortex-A9 Core - The ARM Cortex-A9 multiprocessor core implements the ARMv7 architecture and runs 32-bit ARM instructions, 16-bit and 32-bit Thumb instructions, and 8-bit Java byte code in the Jazelle state. The media processing engine implements ARM NEON coprocessor technology, a single instruction multiple data (SIMD) architecture that adds instructions targeted at audio, video, 3D graphics, image, and speech processing.

General Interrupt Controller (GIC)- The GIC collects interrupts from various sources and distributes these interrupts to each of the ARM cores. The interrupt distributor holds the list of pending interrupts for each ARM Cortex-A9 multiprocessor core and then selects the highest priority interrupt before issuing it to the Cortex-A9 processor interface. Interrupts of equal priority are resolved by selecting the lowest ID. A total of 64 shared peripheral interrupts (PL interrupts and PS I/O peripheral interrupts) are supported, starting from ID 32.

Interconnect

The interconnect unit connects all PS and PL master and slave devices. There are a total of six AXI slave ports dedicated for AXI masters residing in the PL, and four of these ports contain deep FIFOs to improve data throughput. Two AXI master ports provide access to AXI slaves in the PL. In this design, masters in PL are connected through two AXI slave ports with deep FIFOs. One AXI master port is used to access registers in AXI slave IPs in the PL. An advanced peripheral bus (APB) master port is provided for accessing software programmable registers of all PS modules. The top level switch is AXI3-compliant, the soft IPs provided by Xilinx are AXI4-compliant, and the soft AXI interconnect IP provides protocol bridging as needed.

Slave AXI interfaces (S_AXI_HP) - The high performance S_AXI_HP connect the PL to AFI blocks in the PS. The PL has four AXI masters out of which two are connected to the S_AXI_HP0 port and two are connected to the S_AXI_HP2 port. The HP port enables a high-throughput data path between AXI masters in the programmable logic and the processing system DDR3 memory. The main purpose of the AXI FIFO interface (AFI) units is to smooth out variable latency, allowing the ability to stream data continuously from DDR to the PL masters and from the PL masters to DDR. The PL-side interface of AFI runs on the clock coming from the PL. In this design, a 150 MHz clock is connected from the PL side. The DDR-side clock is running at 2/3 of the DDR_CLK (533 MHz). The high performance AXI interface module provides several hooks to assist in bandwidth management of masters connected to different PL ports. Controlling issuance capability available from the PL port is one of the hooks exercised in this design to obtain a fair share of bandwidth between two masters, FILTER VDMA, and the display controller.

AXI master port (M_AXI_GP) - This AXI master port interfaces with AXI slave IPs in the PL through an AXI-Lite interconnect. The CPU manages initializing and controlling the video pipeline through this port.

Input/Output Peripherals

The input/output peripherals (IOP) unit includes communication peripherals. GPIO, Ethernet, USB, I2C, and SD controllers from the PS are used extensively in this design.

GPIO - The 64-bit general purpose input/outputs (GPIOs) are connected to the PL through the extendable multiplexed I/O (EMIO) interface. Sixty-four bits are divided into two banks, each having 32 bits. Because each GPIO bit can be dynamically configured as input or output, GPIO bits are used in this design for a variety of functions.

Memory Interfaces

The memory interfaces unit includes the DDR memory controller and nonvolatile memory controllers. The DDR memory controller includes a 4-port arbiter. One AXI port is dedicated for ARM CPU access and two ports are dedicated for high performance AXI interface master devices in the programmable logic. The remaining port is shared by all other AXI masters. In this design, DDR3 is configured to run at 533 MHz, and the AXI interface is running at 355 MHz.

PL Clocks

The PS provides four (FCLKCLK[3:0]) fully programmable clocks to the PL. These clocks are routed directly to PL clock buffers to serve as a frequency source for the PL. The clock generator module in PL gets a 100 MHz clock from FCLKCLK[0].

PL Reset

The PS provides four (FCLKRESETN[3:0]) fully programmable reset signals to the PL. These signals are asynchronous to PS clocks. The PL logic reset block in this design receives input from FCLKRESETN[0] and generates necessary reset signals for the design implemented in PL.

Programmable Logic

The video IP and custom logic implemented is implemented in PL.

PCIe Endpoint Device

The Zynq-7000 ZC7Z045 AP SoC has a x4 Gen2 capable PCIe block. This block interfaces to the root port in the host system and forms the basic communicating entity with the host. The PCIe Endpoint block provides an AXI Stream compliant interface with 64-bit width operating at 250 MHz.

NWL DMA

The PCIe compatible DMA from Northwest Logic, a Xilinx third-party alliance partner, has AXI stream compliant interface in the user side. Another AXI stream side of the DMA interfaces to the PCIe wrapper. The DMA accepts PCIe transaction layer packets and is responsible for transmitting the completion data back to PCIe block. The NWL DMA also provides an AXI memory-mapped interface that can be used as AXI-Lite compatible register interface.

The NWL DMA performs read and write operations to system memory using buffer descriptors (BD). Each descriptor is mapped to a contiguous buffer of 4 KB in the TRD. A ring of such descriptors is established in a circular fashion and a descriptor is freed up to the free BD pool when the completion bit of the same descriptor is set by the DMA. The BD in the free pool can be reused to transfer another buffer. Each BD carries information of the packet SOP, EOP, byte count, and a pointer to next descriptor. Figure 3-5, page 39 shows the structure of NWL DMA BDs in the system to card (S2C) and card to system (C2S) direction.

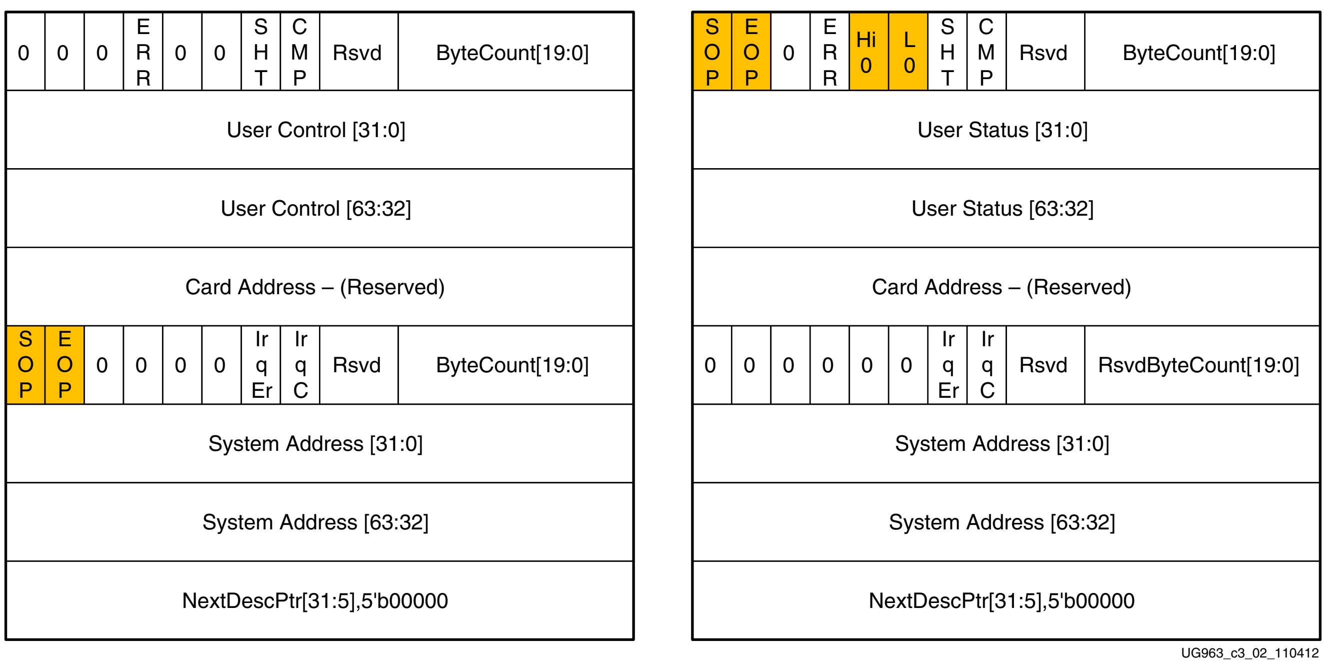


Figure 3-2: **S2C Buffer Descriptor and C2S Buffer Descriptor Format**

In the video processing and offload demonstration, the video frame synchronization signal is generated in host software and the information is passed over to hardware using the user control field of the NWL DMA BD. Figure 3-6, page 39 shows the timing diagram with video frame synchronization signal.

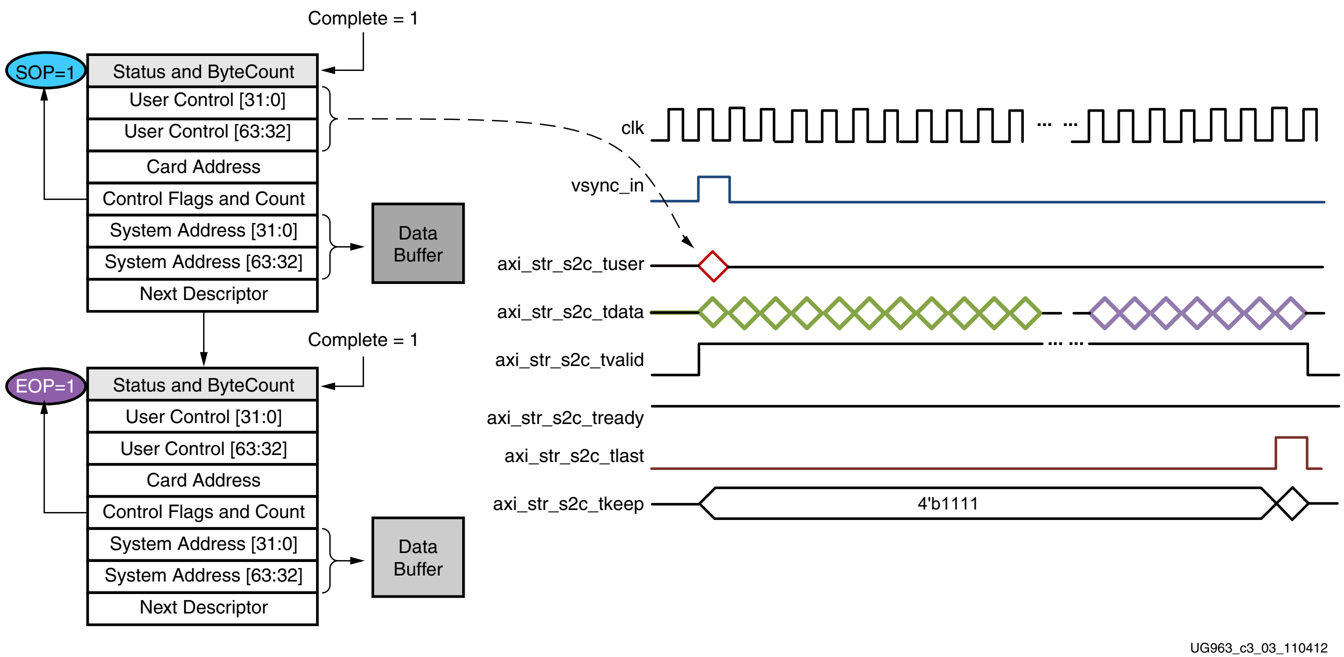


Figure 3-3: **Video Data Transfer in S2C Direction**

In the C2S direction video frames are received continuously. The timing diagram of C2S direction is shown in Figure 3-7, page 40. Note that there video frame synchronization signal is not present in the C2S interface because the video processing logic in host does not require any video synchronization signal.

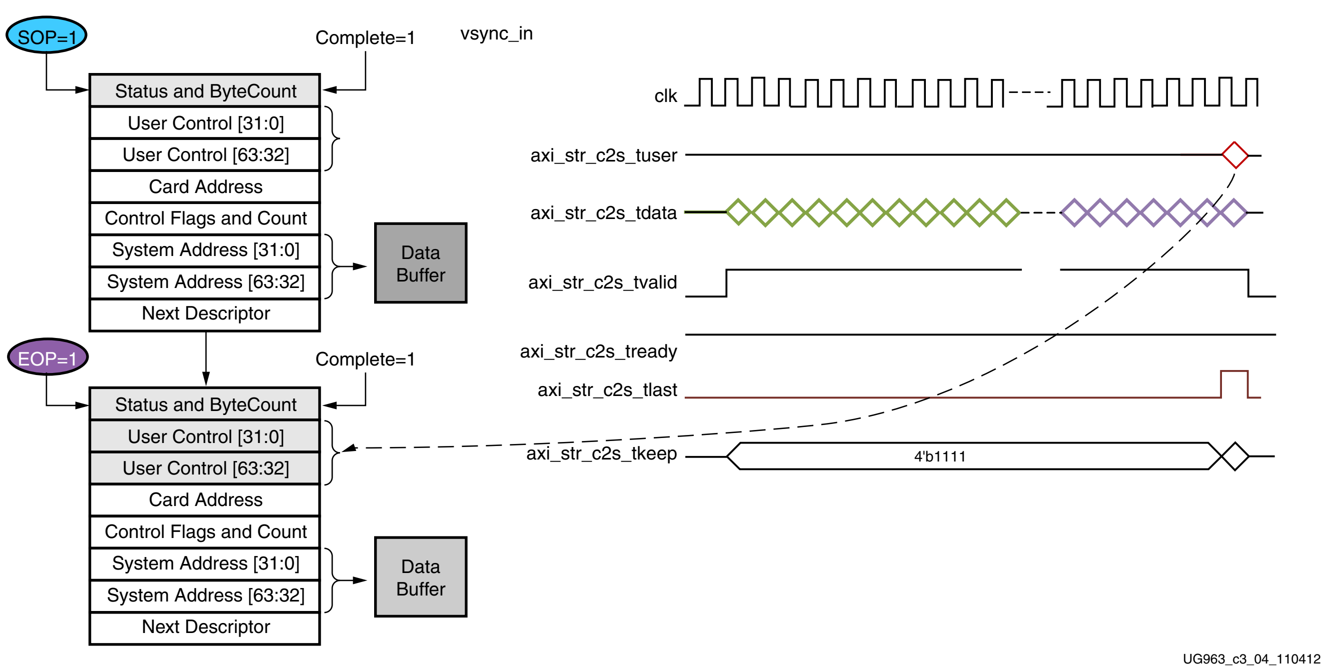


Figure 3-4: **Video Data Transfer in C2S Direction**

AXI Interconnect

The PL design has two interconnects for AXI memory-mapped masters and one interconnect for the AXI register interface. AXI memory-mapped interconnects are connected to masters like AXI_VDMA and logiCVC-ML. Slaves connected to these interconnects include HP0 and HP2 ports of the Zynq-7000 XC7Z045 AP Soc PS. This interconnect operates at 150 MHz and the data width is 64-bits. The read/write acceptance and issuance are set to 8 levels. The acceptance and issuance helps improve system performance. The PS HP port can accept a maximum burst length of 16 levels. This imposes a limitation on getting minimum acceptable bandwidth for every master in a multi-master system. The optimum setting of issuance and acceptance reduces throttle on the bus and compensates for long latencies. The AXI register interface is clocked at 75 MHz. The XC7Z045 AP Soc GP0 port acts as master on this interconnect and connected slaves have register maps. AXI TPG and AXI VTC are examples of slaves connected to this interconnect. The operations of the video pipeline are controlled by registers inside every IP. Depending upon data flow required in the video pipeline, the processor writes these registers through the AXI-Lite interconnect. The AXI-Lite interconnect accepts write or read transfers from the CPU, performs address decoding, selects a particular slave, and establishes a communication channel between the CPU and the slave device.

AXI Video DMA

The AXI VDMA has an AXI streaming interface on one side and an AXI memory-mapped interface on the other side. The VDMA has two channels: MM2S (memory-mapped to streaming) and S2MM (streaming to memory-mapped). The MM2S channel reads the number of data beats programmed through the C_MM2S_MAX_BURST_LENGTH parameter and presents it to the slave device connected through the streaming interface. The data width of the streaming interface can be different than the memory-mapped interface and controlled through C_M_AXIS_MM2S_TDATA_WIDTH. The data width of the S2MM memory-mapped interface is controlled by the C_M_AXI_MM2S_DATA_WIDTH parameter. The S2MM channel receives data from the master device connected through the streaming interface. The C_S_AXIS_S2MM_TDATA_WIDTH parameter decides the width of the streaming interface. Data received on the streaming interface is then written into the system memory through the memory-mapped interface. The C_M_AXI_S2MM_DATA_WIDTH parameter decides the data width of the memory-mapped interface and C_S2MM_MAX_BURST_LENGTH governs the burst length of the write transaction. In this design, the streaming interface data width is set to 32-bits and the memory-mapped interface is configured as 64-bits wide. The AXI VDMA is used in simple register direct mode, which removes the area cost of the scatter-gather feature. Initialization, status, and management registers in the AXI VDMA core are accessed through an AXI4-Lite slave interface. To get the best possible throughput for AXI VDMA instances, the maximum burst length is set to 16. In addition, the master interfaces have a read and write issuance of 8 levels and a read and write FIFO depth of 512 to maximize throughput. The line buffers inside the AXI VDMA for the read and write sides are set to 4K deep and the store and forward feature of the AXI VDMA are enabled on both channels to improve system performance and reduce the risk of system throttling.

logiCVC-ML

The logiCVC-ML is a multi-layer video display controller from Xylon. The logiCVC-ML controller refreshes the display image by reading the video memory and converting the read data into a data stream acceptable for the display interface. It generates control signals for the display, and supports multiple layers with video processing functions such as alpha blending, transparency, and move around.

AXI XADC

The AXI XADC IP provides the device temperature which is periodically monitored by the application running on the PS. The communication between the XADC IP and the SW on the PS happens via the AXI4 Lite Interconnect bus connected to GP0 port of the PS.

AXI Performance Monitor

The AXI Performance Monitor can monitor and analyze system behavior on the AXI interface. This core is used in the TRD to measure read and write throughput on AXI slave ports of the PS (HP0 and HP2), which are used to access DDR memory from the PL. The core consists of the AXI4-Lite interface to configure and control the core. This core is configured to measure the read and write throughput by counting the number of transactions per second. When the configured time interval expires, measured throughput in bytes is loaded into a register and read by the software application. Two AXI Performance Monitors are instantiated to measure read and write throughput of HP0 and HP2 simultaneously.

PL Address Map

Table 3-1 shows the address mapping of various peripherals used in the TRD.

Table 3-1: **PL Address Map**

Instance	Base Address	High Address
SOURCE_VDMA	0x40090000	0x4009FFFF
FILTER_VDMA	0x400b0000	0x400bFFFF
LOGICVC_0	0x40030000	0x4003FFFF
PERF_MONITOR_HP2	0x400E0000	0x400EFFFF
PERF_MONITOR_HP0	0x400F0000	0x400FFFFF
FILTER_ENGINE	0x400D0000	0x400DFFFF
AXI_EXT_SLAVE_CONN_0	0x40020000	0x4002FFFF
AXI_XADC_0	0x40050000	0x4005FFFF

GEN/CHECK Performance Mode Modules

This performance mode module provides the following options-

- AXI-Stream traffic generation (for a programmed frame size) for C2S direction
    - The frame data will be incremental or be a sequence number for ease of data integrity check in software driver.
- AXI-Stream traffic checker (for a programmed frame size) for S2C direction
    - The frame data generated by software will be incremental or be a sequence number for ease of data integrity check in hardware.
- Option to loopback S2C traffic to C2S engine
    - Data is generated by software and looped back in hardware.

The traffic generator and checker interface follows AXI4 stream protocol. The packet length is configurable through control interface.

The traffic generator and checker module can be used in three different modes- a loopback mode, a data checker mode, and a data generator mode. The module enables specific functions depending on the configuration options selected by the user (which are programmed through control interface to user space registers). On the transmit path, the data checker verifies the data transmitted from the host system via the Packet DMA. On the receive path, data can be sourced either by the data generator or transmit data from host system can be looped back to itself. Based on user inputs, the software driver programs user space registers to enable checker, generator, or loopback mode of operation.

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

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

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

Table 3-2: **Packet Format**

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

Table 3-2: **Packet Format *(Cont'd)***

--	--	--	--
TAG	TAG	TAG	TAG

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

Packet Checker

If the Enable Checker bit is set (Registers as defined in Section 7.2.1.2), as soon as data is valid on the DMA transmit channels (namely S2C1) each data byte received is checked against a pre-decided data pattern. If there is a mismatch during a comparison, the data_mismatch signal is asserted. This status is reflected back in register which can be read through control plane.

Packet Generator

If the Enable Generator bit is set (Register as defined in Section 7.2.1.2) and the data produced by the generator is passed to the receive channel of the DMA (namely C2S1). The data from the generator also follows the same pre-decided data pattern as the packet checker.

Register Interface

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

The design has the user space control register interface defining design mode configuration, control and status.

AXI4LITE Interconnect is used to fan out the AXI4 target interface to the appropriate slave address region as defined below.

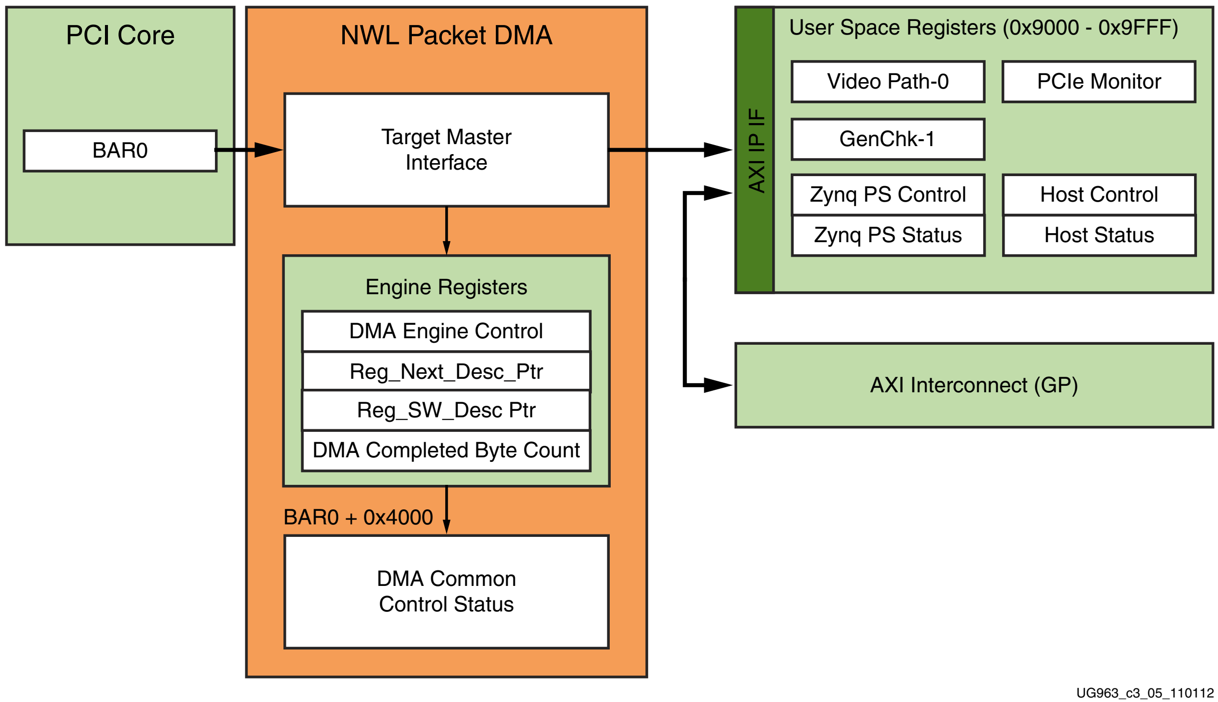


Figure 3-5: **Register Interface**

In order for the user space registers to connect to the AXI-Lite interface; these registers will be wrapped under AXI-Lite slave. IP axi_lite_ipif_v1_01_a will be used to provide the AXI-Lite interface. The registers read/write logic will be connected to the IPIF back end.

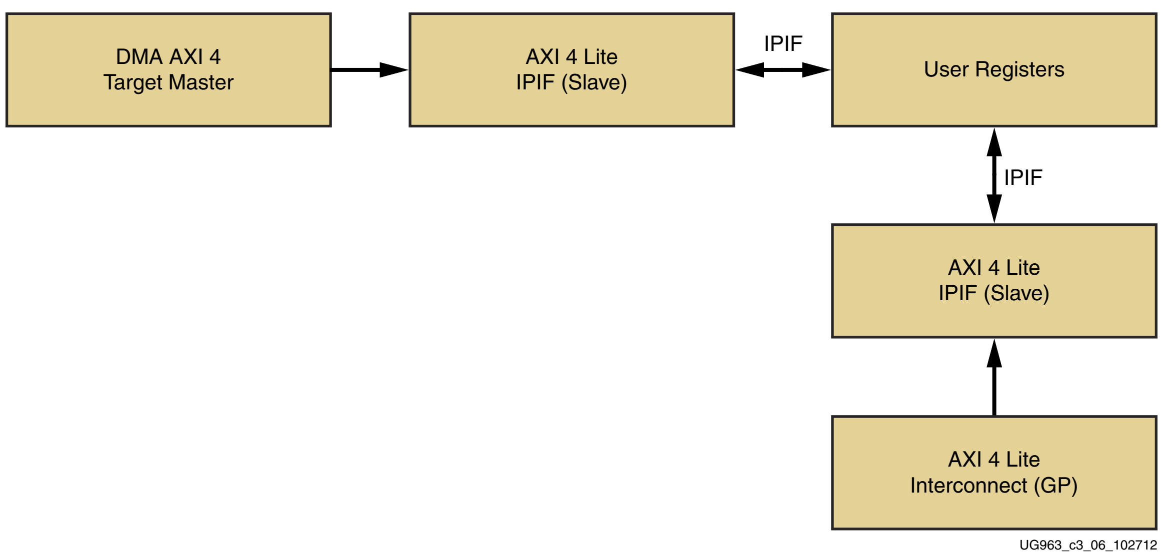


Figure 3-6: **Register Logic Blocks**

Clocking and Reset

This section provides details on clocking and reset connections in the TRD.

Clocking

The PL design has four clock domains:

- PCIe clock domain- 100 MHz differential SSC PCIe clock from host system through PCIe connector
- AXI MM (memory-mapped) interconnect - 150 MHz
- AXI register interface- 75 MHz
- Video clock- 148.5 MHz

The PCIe clocks are derived from the 100 MHz differential clock. The NWL DMA uses the 250 MHz clock generated by the PCIe block. The NWL DMA provides an AXI master interface which will be used as control interface in the TRD for communicating with the host. A domain crossing FIFO will be used to bring the clock frequency down to 75 MHz in the control path.

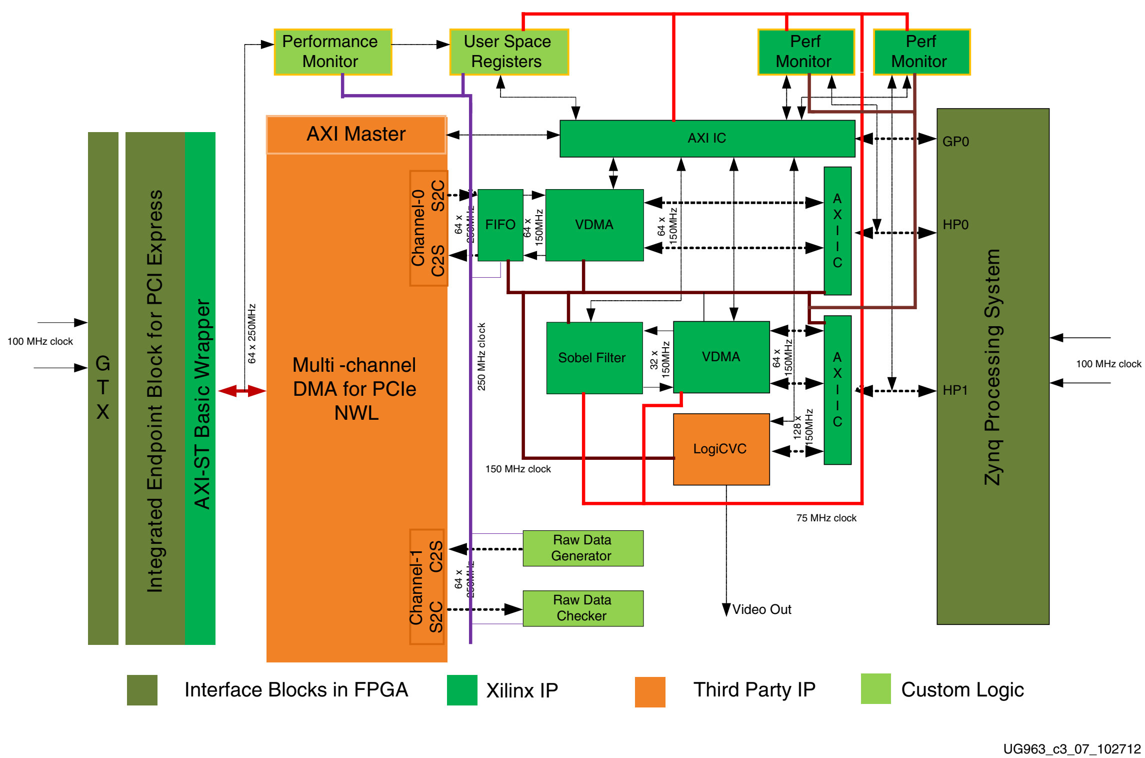


Figure 3-7: **Clocking Scheme**

In the video path, the clock generator module receives a 100 MHz input clock from the PS FCLKCLK[0] and generates 75 MHz and 150 MHz. The AXI-Lite interconnect works on 75

MHz. Apart from the AXI-Lite interconnect, the register interface of AXI VDMA, logiCVC-ML and perf_monitor cores are driven by the 75 MHz clock. sobel_filter is driven by 150 MHz clock. Two instances of the AXI_MM interconnect connected to the HP port of the PS run on 150 MHz. The S2MM (stream to memory map) and MM2S (memory map to stream) channels of VDMAs are running at 150 MHz. The 150 MHz clock also drives the logiCVC-ML memory read interface. The video clock comes from the external clock synthesizer

The clocking scheme of the TRD is shown in Figure 3-10, page 45.

Reset

In PCIe based systems, no external reset pins (esp. switches, push buttons) are used. There is only a PERST# driven by the host processor which is a hard reset and for all intermittent purposes, soft resets (resets driven under software control through register programming) should be used.

Reset to the blocks present in the video processing path is derived from PROC SYS reset module which gets FCLK reset, DCM lock and PCIe user link up as inputs. The Sobel filter gets an additional reset from the PS GPIO pin which is driven by the PS software. The NWL DMA soft reset is applied to SOURCE VDMA block based on the decision from PS software. The raw path traffic generator and checker blocks are reset based on PCIe user link up status and soft reset coming from NWL DMA block.

Figure 3-11, page 47 shows the detailed reset scheme used in the TRD.

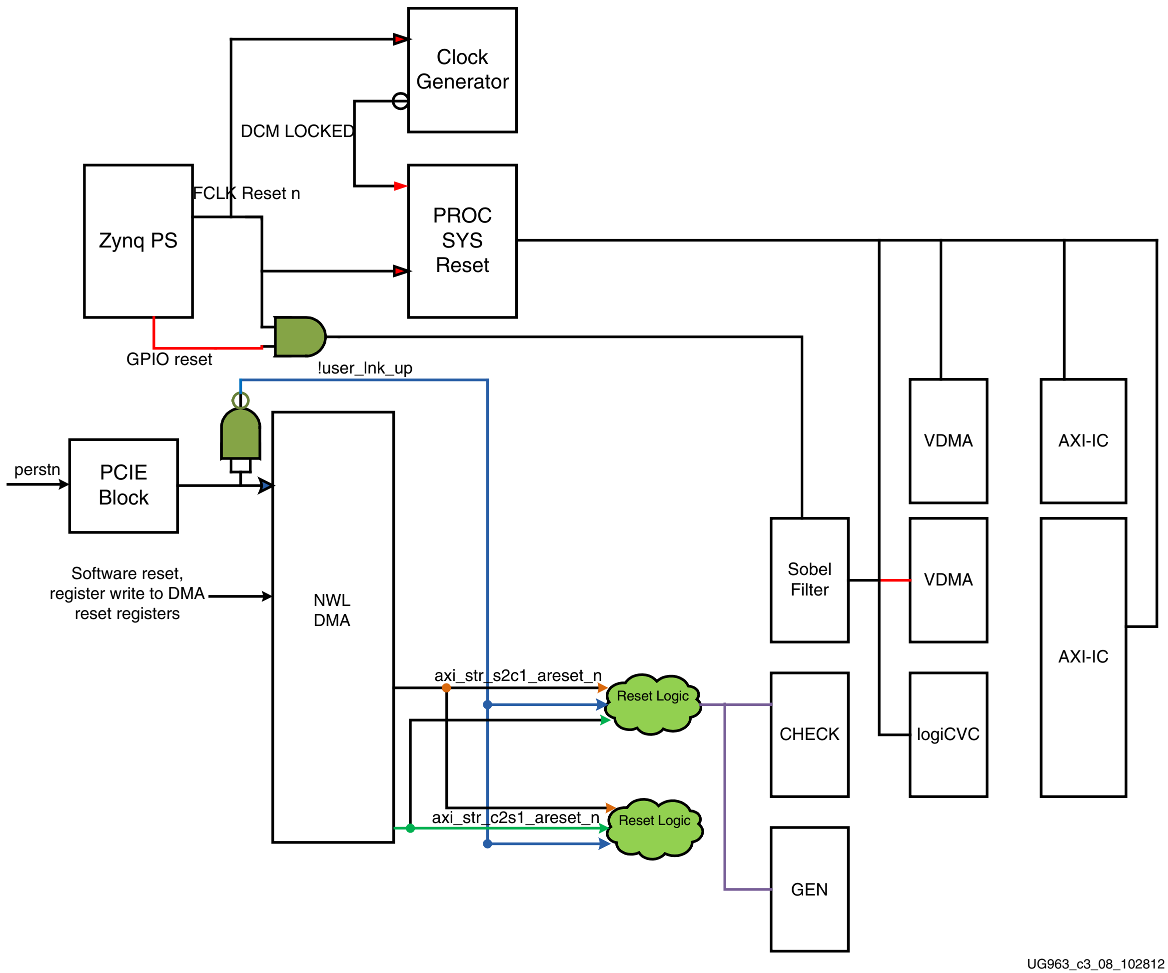


Figure 3-8: **TRD Reset Scheme**

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

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

Software Architecture

The software for Zynq-7000 PCIe TRD comprises of several Linux kernel-space drivers and a user-space application. Traffic is generated from user application. Format of data changes

from 1080p color bar to raw data modes. The following sections explain data and control path flow.

Host System

The software driver architecture for video processing and offloading demo is shown in Figure 3-12, page 48.

The application software in host formats three buffers with three different color bar patterns. Each color bar pattern is transmitted through the NWL DMA to the logiCVC display block present in the video path or transferred back to host after Sobel processing. Each buffer contains one video frame of size 1920x1080p and each data buffer is scheduled to transfer through the NWL DMA in 4KB packet size. Each NWL DMA data buffer spans a size of 1920 pixel.

Each NWL DMA spans = 1920x4 = 7680B

Number of BDs required to transfer one video frame = 1080

A BD ring of size 2999 BDs is initialized to generate traffic on the video path. As the completion BDs come from the DMA,

BDs can be pushed back to free BD pool and the data transfer rate can be maintained.

The host software generates video frame synchronization signal at the start of every video frame. The application software also maintains appropriate inter-frame gap so that video processing can be carried out in PS. Figure 3-12, page 48 shows the video frame processing in the host software.

Following sections describe the software components in the TRD in details. The description is divided into data and control path components.

Data Path Components

Video Traffic generator

This block will generate the video data in the form of color bar. The application opens the interface of application driver through exposed driver entry points. Application transfers the data using read and write entry points provided by application driver interface. Application video traffic generator also performs the data integrity test in receiver side, if enabled.

Driver Entry Point

This block creates a character driver interface and enhances different driver entry points for user application. The Entry point also enables sending of free user buffers for filling DMA

descriptor. Additionally the entry point conveys completed transmit and receive buffers from driver queue to user application.

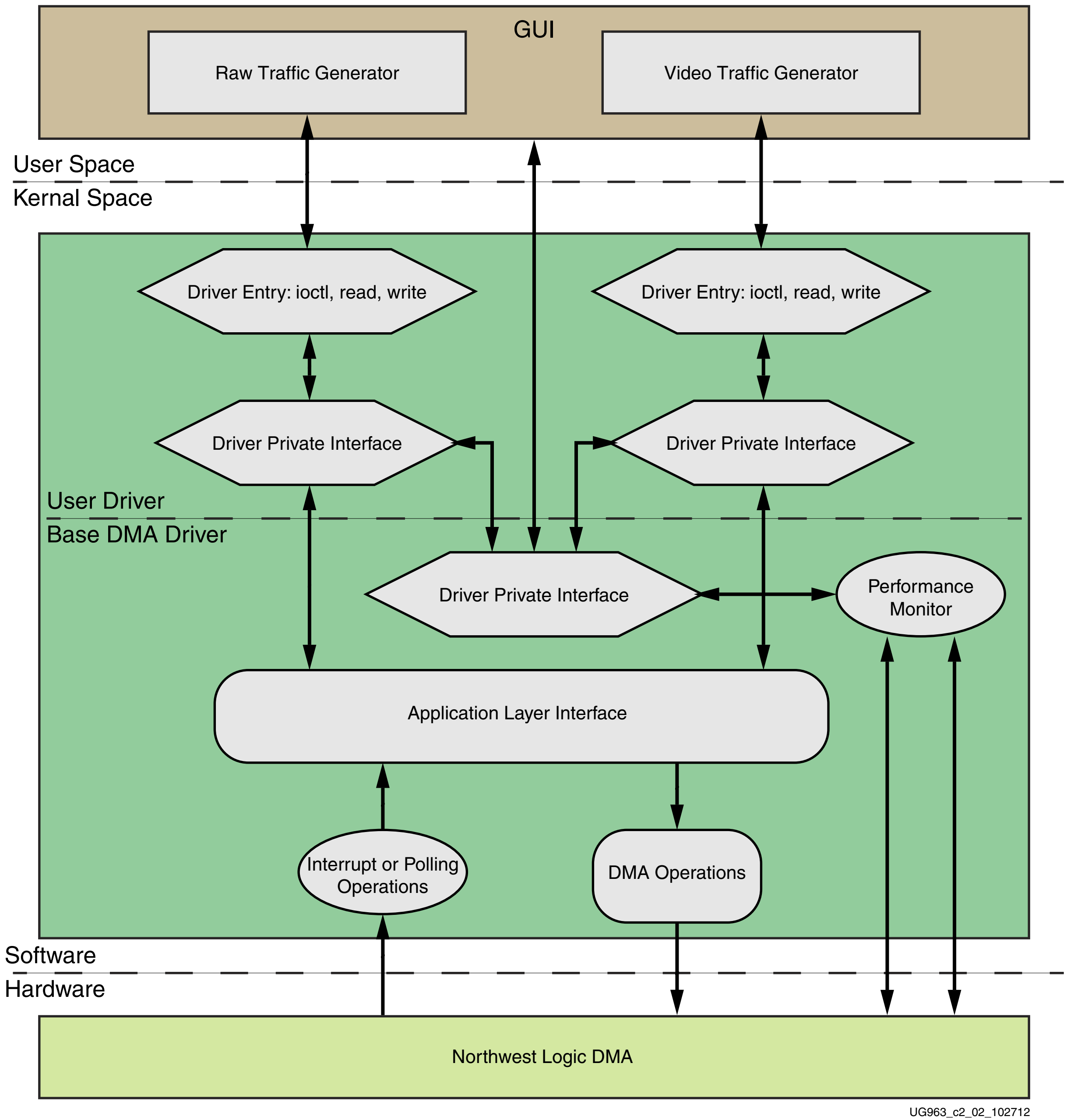


Figure 3-9: **Host Software Architecture**

Driver Private Interface

This block enables interaction with DMA driver through private data structure interface. The data that comes from user application through driver entry points is sent to DMA driver through private driver interface. The private interface handles received data and

housekeeping of completed transmit and receive buffers by putting them in completed queue.

Application Driver Interface

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

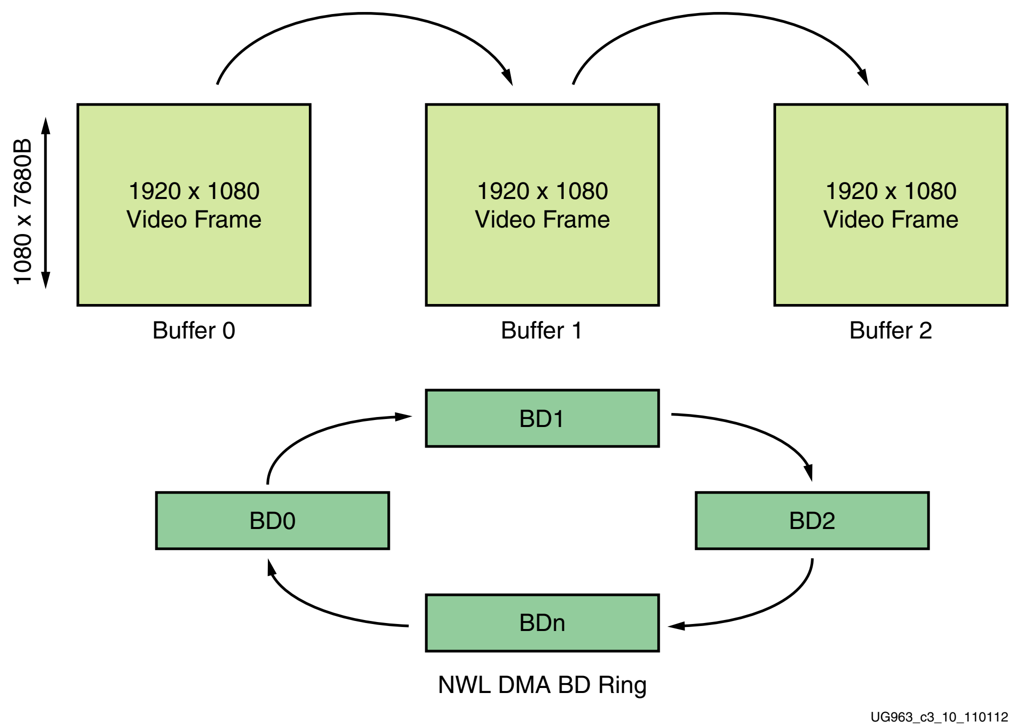


Figure 3-10: **Buffer Format**

NWL DMA Operation

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

Interrupt or Polling Operation

If interrupts are enabled, the interrupt service routine (ISR) handles interrupts from the DMA engine. The driver sets up the DMA engine to interrupt after every N descriptors that it processes. This value of N is set by a compile-time macro. The ISR schedules bottom half (BH) which invokes the functionality in the driver private interface pertaining to handling received data and housekeeping of completed transmit and receive buffers.

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

a. Housekeeping of completed transmit and receive buffer

b. Handling of received data.

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, is bursty, and packets are not of fixed sizes. For example, connect/disconnect establishment and ACK/NAK packets are small. Therefore, the software is not able to determine in advance the number of packets to be transferred, and accordingly set up a descriptor chain for it. Packets can fit in a single descriptor, or may be required to span across multiple descriptors. Also, on the receive side the actual packet may be smaller than the original buffer provided to accommodate it.

It is therefore required that:

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

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

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

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

Table 3-3: **Terminology Summary**

Term	Description
HW_Completed	Register with the address of the last descriptor that DMA engine has completed processing
HW_Next	Register with the address of the next descriptor that DMA engine will process
SW_Next	Register with the address of the next descriptor that software will submit for DMA
ioctl()	Input Output Control function is a driver entry point invoked by the application tool

Dynamic DMA Updates

This section describes how the descriptor ring is managed in the Transmit or System-to-Card (S2C) and Receive or Card-to-System (C2S) directions. It does not give details on the driver's interactions with upper software layers.

Initialization Phase

Driver prepares descriptor rings, each containing 8192 descriptors, for each DMA channel. In the current design, driver will thus prepare 4 rings.

Transmit (S2C) Descriptor Management

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

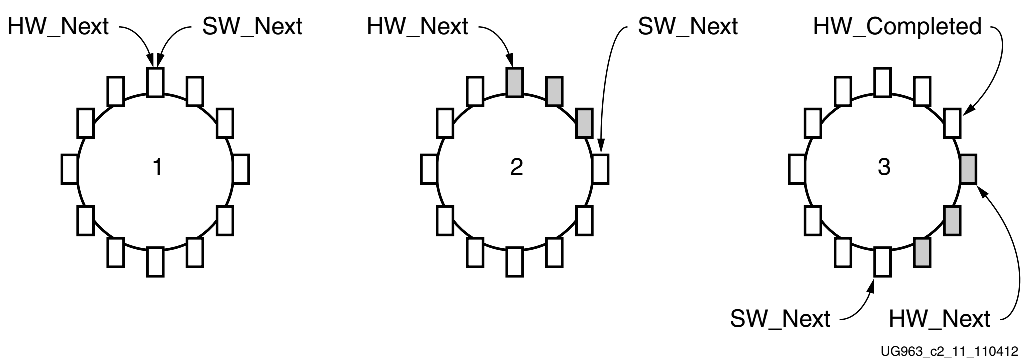


Figure 3-11: **Transmit Descriptor Ring Management**

- Initialization Phase (continued)
    - Driver initializes HW_Next and SW_Next registers to start of ring
    - Driver resets HW_Completed register
    - Driver initializes and enables DMA engine
- Packet Transmission
    - Packet arrives in packet handler
    - Packet is attached to one or more descriptors in ring
    - Driver marks SOP, EOP and IRQ_on_completion in descriptors
    - Driver adds any User Control information (e.g. checksum-related) to descriptors
    - Driver updates SW_Next register
- Post-Processing
    - Driver checks for completion status in descriptor
    - Driver frees packet buffer

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

Receive (C2S) Descriptor Management

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

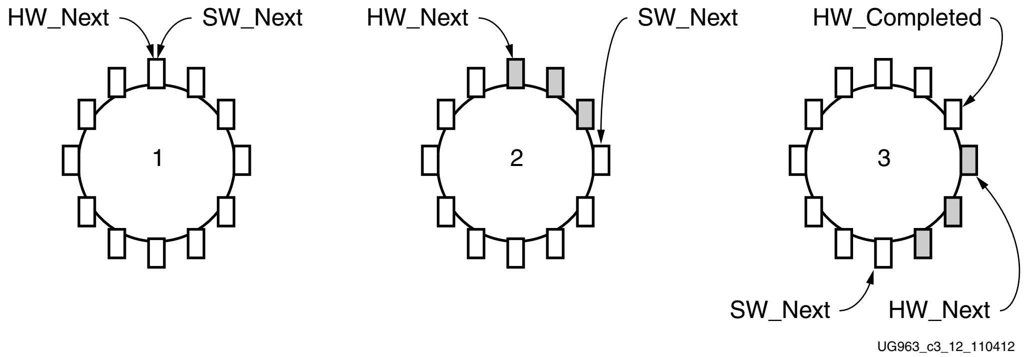


Figure 3-12: **Receive Descriptor Ring Management**

- Initialization Phase (continued)
  - Driver initializes each receive descriptor with an appropriate Raw Data buffer
  - Driver initializes HW_Next register to start of ring and SW_Next register to end of ring
  - Driver resets HW_Completed register
  - Driver initializes and enables DMA engine
- Post-Processing after Packet Reception
  - Driver checks for completion status in descriptor
  - Driver checks for SOP, EOP and User Status information
  - Driver forwards completed packet buffer(s) to upper layer
  - Driver allocates new packet buffer for descriptor
  - Driver updates SW_Next register

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

User Interface - Control & Monitor GUI

While invoking GUI a launching page is displayed which detects the PCIe device for this design (Vendor ID = 0x10EE and Device ID = 0x7042). Only on detection of the appropriate device, it allows driver installation to proceed through.

GUI Control Function

Several parameters are controlled through the GUI:

- Video processing selection on the video path
- Packet size variation on the data path.
- Test type **loopback**, **Hw checker**, or **Hw generator** for performance on the data path
- GUI monitor function

The driver always maintains information about the hardware status. The GUI periodically invokes an I/O Control interrupt, ioctl() to read this status information which is comprised of:

- PCIe link status and device status
- DMA engine status

The driver maintains a set of arrays to hold per-second sampling points of different kinds of statistics, which are periodically collected by the performance monitor handler. The arrays are handled in a circular fashion. The GUI periodically invokes an ioctl() interrupt to read these statistics, and then displays them:

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

The various GUI fields as per the numbering in Figure 3-16, page 57 are explained here:

1. Sobel Filter- available options are: None, HW-Sobel, SW-Sobel. None does not enable any processing on the generated Video frame. HW-Sobel enables Sobel processing in the PL and SW-Sobel enables processing of the video frame in the PS.
2. Video Out- available options are: HDMI, PCIe Host. In the HDMI mode, the generated video frames are displayed on the 1080p60 monitor. In the PCIe Host mode, the Sobel processed (HW or SW) frames return back to PCIe Host.
3. Test start/stop control for performance mode
4. DMA statistics and software BD gives following information:

- Throughput (Gb/s). DMA payload throughput in gigabits per second for each engine.
- DMA Active Time (ns). The time in nanoseconds that the DMA engine has been active in the last second.
- DMA Wait Time (ns). The time in nanosecond that the DMA was waiting for the software to provide more descriptors.
- BD Errors. Indicates a count of descriptors that caused a DMA error. Indicated by the error status field in the descriptor update.
- BD Short Errors. Indicates a short error in descriptors in the transmit direction when the entire buffer specified by length in the descriptor could not be fetched. This field is not applicable for the receive direction.
- SW BDs. Indicates the count of total descriptors set up in the descriptor ring.

Figure 3-13: **Software GUI**

Table Table 3-4 describes the user interface functions.

Table 3-4:

Callout	Function	Description
1	Sobel Filter	Sobel Filter ON/OFF selection option. Sobel-HW enables hardware Sobel filter operation. Sobel-SW enables software Sobel filter operation.
2	Video Out	Output video option. HDMI option enables display on HDMI monitor. PCIe Host option sends the processed video data (HW-Sobel or SW-Sobel) back to PCIe host
3	Stop Button	Test stop button.

Table 3-4:

Callout	Function	Description
4	DMA Channel Parameters	NWL DMA Statistics.
5	PCIe Transmit Writes	Reports the transmitted (Endpoint card to host) utilization as obtained from the PCIe performance monitor in hardware (Gb/s).
6	PCIe Receive Reads	Reports the received (host to Endpoint card) utilization as obtained from the PCIe performance monitor in hardware (Gb/s).
7	Operating Mode	In performance GEN/CHEK mode user has options of selecting Loopback or Hw Gen/Hw checker option.
8	Message Log	Shows messages, warnings, or errors.
9	PCIe Endpoint Status	Reports the status of PCIe fields in the Endpoint configuration space.
10	Host System Initial Credits	Initial Flow control credits advertised by the host system after link training with the Endpoint. A value of zero implies infinite flow control credits.
11	Transmit Performance Plots	Plots the PCIe transactions on the AXI4-Stream interface and shows the payload statistics graphs based on DMA engine performance monitor.
12	Receive Performance Plots	Plots the PCIe transactions on the AXI4-stream interface and shows the payload statistics graphs based on DMA engine performance monitor.
13	Close Button	Closes GUI.
14	Block Diagram Button	This button displays a block diagram of the currently running mode.

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

Processing System

Figure 3-14 shows the block diagram of the Software running on the Cortex-A9 processor in the Zynq-7000 device. A multi-threaded Linux application calls the appropriate device driver to configure the hardware and enable a particular data path. Three major components are involved-

- Boot Loader
- Xilinx Linux Kernel
- Application

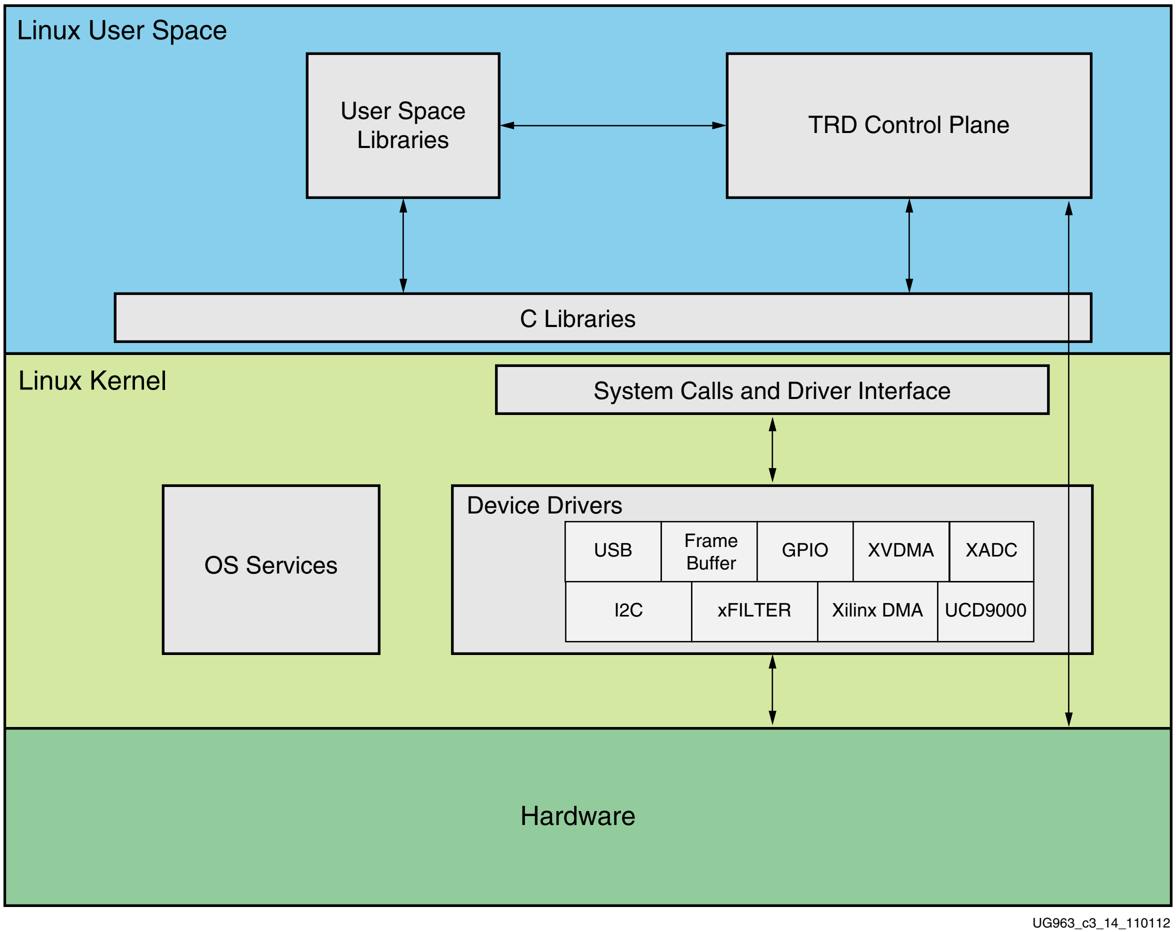


Figure 3-14: **Cortex-A9 Software Block diagram**

A two-stage boot loader will be used for the XC7Z045 AP SoC Linux boot-up. The Xilinx Linux kernel is based on the mainline open source kernel git tree, adding support for a variety of Xilinx IP core drivers and reference boards.

The application software performs the following-

- Control and Decision making
- User space device control

Boot Loader

A two-stage boot loader is used for the XC7Z045 AP SoC Linux boot-up. The FSBL is responsible for initializing required hardware and loads the second-stage boot loader, U-Boot, which is responsible for loading kernel images in the DDR memory. The FSBL source code is generated through the Xilinx SDK tool, depending on the hardware design specification.

Xilinx Linux Kernel

The Xilinx Linux kernel is based on the mainline open source kernel git tree, adding support for a variety of Xilinx IP core drivers and reference boards. The source code is available on the Xilinx Open Source ARM git Repository. The Xilinx Linux kernel is extended (patched) to support IPs specific to this TRD.

Frame Buffer Driver

Linux provides a standard frame buffer, which is hardware-independent, and the application can use this buffer without knowing the underlying display controller. The Xylon frame buffer driver for CVC IP is registered with the standard frame buffer driver to provide support for the logiCVC-ML display controller. The Xylon frame buffer driver is compiled with the kernel and probes for the hardware and resolution specification by scanning the dtb file at boot. If there is no entry for logiCVC-ML IP in the dtb file, then the driver does not load itself.

PS-GPIO Driver

The GPIO SYSFS interface is used for GPIO configuration. The GPIO SYSFS interface allows the user to control I/O pins using files under the `/sys` directory. When the system boots, all GPIO pins are owned by the kernel. The pins do not show up in the SYSFS system until they are exported. To export them, write the pin number (for example 54) to the file `sys/class/gpio/export`.

This results in the pseudo files for pin 54 showing up under `/sys/class/gpio/gpio54` as:

```
/sys/class/gpio/gpio54/direction
```

```
/sys/class/gpio/gpio54/value
```

The user can set the direction by writing in or out to the direction file. For the out direction, writing 0 or 1 on the corresponding value file resets or sets the pin, respectively. Similarly, the user can read the value file for the in direction.

XVDMA Driver

XVDMA is a character driver used for configuring and controlling VDMA transactions for both TPG and Sobel hardware. XVDMA internally calls the Xilinx DMA driver to complete the task and interrupt handling. The device node that uses the XVDMA driver is `/dev/xvdma`. The following configuration modes of VDMA will be used in the TRD:

Table 3-5: **VDMA Operation**

VDMA	S2MM		MM2S	
	Hardware Sobel	**Software Sobel**	**Hardware Sobel**	**Software Sobel**
SOURCE VDMA	Circular	Park	Circular	Park
FILTER VDMA	Circular	Park	Circular	Park

xFilter

xFilter is a character driver used for configuring and controlling a filter IP, the Sobel filter in this case. This driver takes input from the dts file for register map and interrupt ID. xFilter works in two modes-Continuous mode and On demand mode. In Continuous mode, once start ioctl is issued, xFilter runs until stop ioctl is issued. In On demand mode, each start ioctl represents the filtering of a single frame.

The start ioctl call is always asynchronous. The user can confirm completion of the filter operation on the current frame by calling the wait_for_completion ioctl, which is a blocking call.

XADC Driver

XADC Driver is a sysfs-based driver that comes with a stock Xilinx Linux kernel. It provides a sys filesystem interface for reading temperature and voltage.

To get the device temperature read the following file:

```
/sys/bus/platform/drivers/xadcps/f8007100.xadc/temp
```

The output gives the temperature in milli-Celsius.

UCD9000 Driver

UCD9000 is a driver for the UCD90120 voltage controller chip that comes with the Xilinx Linux kernel. It provides a Linux sysfs virtual file system interface for reading current and voltage.

To get a specific voltage rail current in mA, read this file:

```
/sys/bus/i2c/devices/8-0065/curr<rail_id>_input
```

To get a specific voltage rail voltage in mV, read this file:

```
/sys/bus/i2c/devices/8-0065/in<rail_id>_input
```

Application

The application is divided into the following functional blocks:

- Graphical display
- User space device control
- Software Sobel filter processing
- Performance & Power Monitoring

There are separate threads each for SW sobel filter processing and monitoring purpose.

The application is based on Qt framework. It is a multi-threaded and event based. The main application is Qt based GUI. The functionality (Video paths, hardware controls etc) are plugged into this main application as separate threads. These threads interact with each other, and by means of thread-synchronization mechanisms, various functions are achieved.

There are mainly four threads spawned at the beginning.

CMD_EXECUTE: this thread continuously parses the commands from PCIe host. If it finds a valid command, it takes necessary action. For example, if this thread finds the command to start Hardware Sobel, it configures the VDMAs accordingly and starts the Video path. This thread continues to execute till it finds the Exit event.

HOST_COMM: this thread continuously monitors the PCIe Host command/status registers. It monitors every one second and updates the command/status words. These command/status are further processed and executed by another thread.

SW_SYNC: this thread is mainly to handle the software Sobel video path. In all video paths, the VDMAs are operated in circular mode, that means the buffer switching and processing done in circular fashion and hardware tracks the same. However in software Sobel, the VDMA is operated in park mode that means software processes each buffer and switches only after completing the process. Hence a dedicated thread is created to handle this video buffer switching.

PERF_MONITOR: this thread continuously reads the performance numbers on HP0 and HP2 ports, also it reads various XADC numbers such as temperature, voltages, currents, compute powers and provide all these numbers to graphs plot thread.

The thread interaction is shown in Figure 3-15.

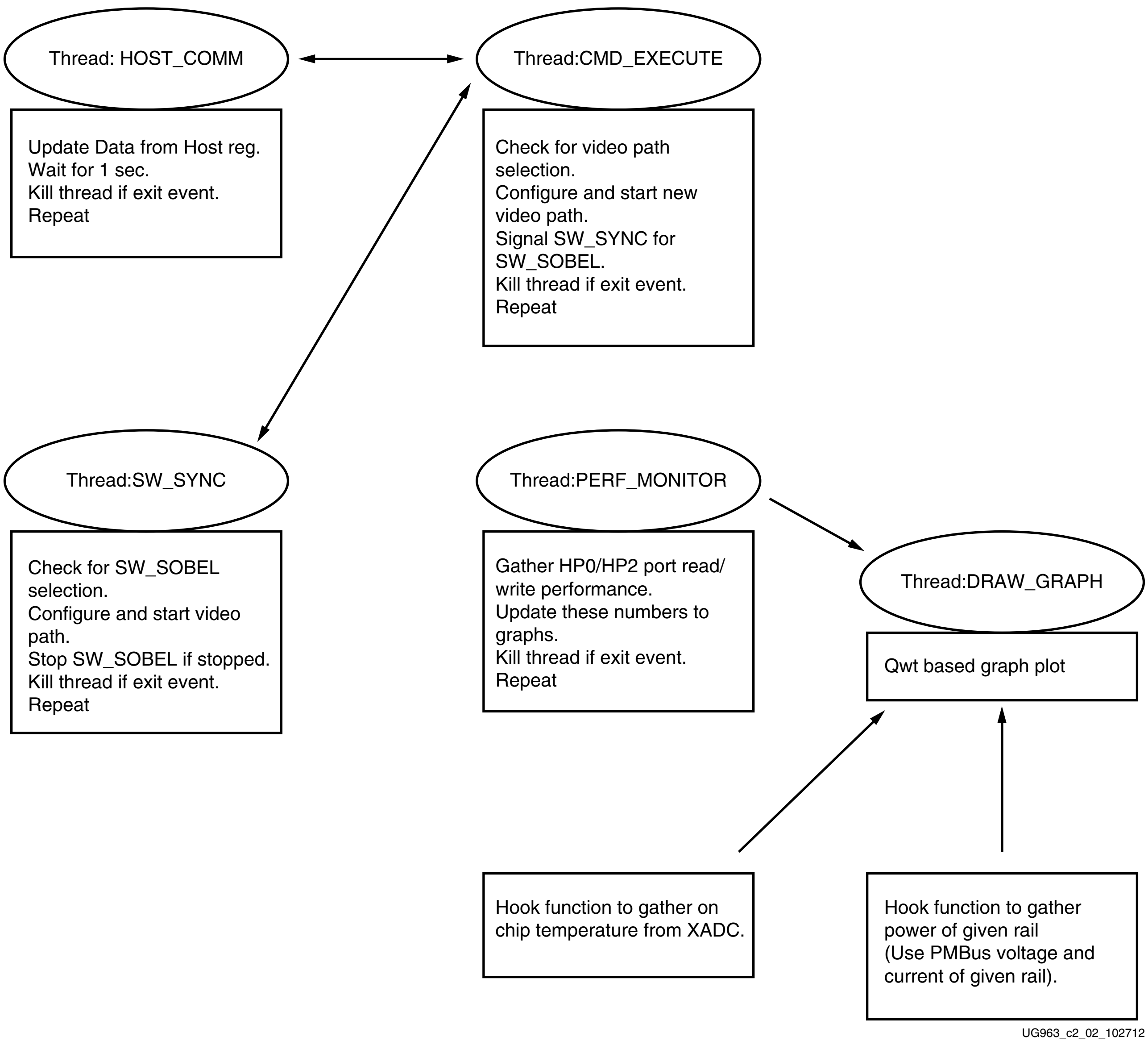


Figure 3-15: **Thread Interaction in the PS**

The main tasks the graphical display performs include:

- Plotting graphs
- Displaying the video area
- Power and temperature monitoring

This section explains the PS GUI in detail.

1. Power plots for VCCINT, VCCAUX, VCC1.5, VCCADJ and VCC3.3V.

2. Device temperature plot.
3. CPU1 and CPU2 utilization plot. This plot specifies the CPU utilization for each of processors present in the PS.
4. HP0 and HP2 port performance plot. This plot signifies memory read and write happening in each of HP0 and HP2 ports.
5. GUI Min button: This button minimizes the Qt GUI screen by removing the graph plots and keeping only the text views. Once the screen is minimized, the Min button is replaced with another button Max. User can click on that button to maximize the GUI.
6. The Health Monitor section is read only and displays various statistics.
7. Performance monitor displays the HP port performance numbers
8. Transparency: this is a slider to control the transparency of the Qt GUI screen, so that the demo view can be maximized. Using mouse, this slider can be moved to left or right
9. Exit button: this button is to exit the application

Figure 3-16 shows the GUI that runs on the PS.

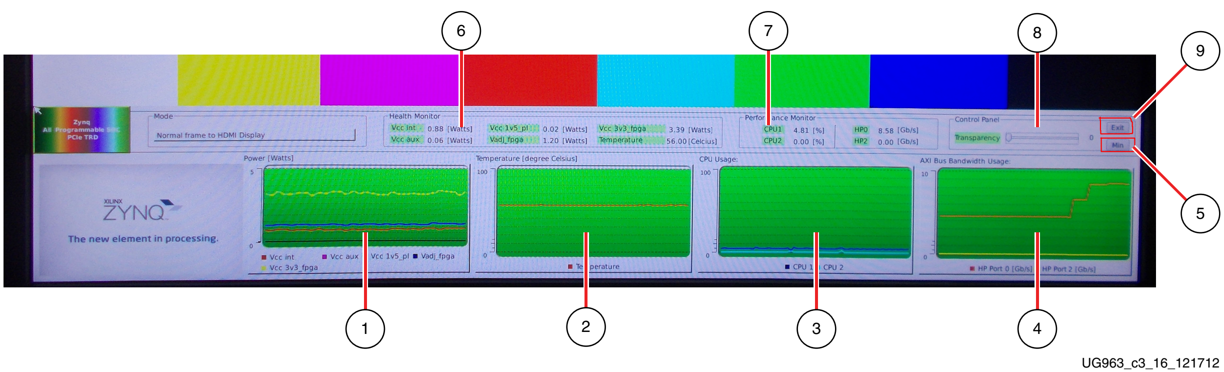


Figure 3-16: **GUI running in the PS**

Chapter 4

Performance Estimation

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

Theoretical Estimate

This section provides a theoretical estimate of performance.

PCIe – DMA

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

PCIe performance depends on parameters like MPS, MRRS, RCB, credits etc. which are very much host system dependent. The calculation below gives a ball-park estimate on the performance of the PCIe-DMA block considering various memory read and write transactions.

Assumptions:

- Each descriptor points to a 4KB buffer space which is page size in systems
- MPS = 256B, MRRS = 512B, RCB = 64B
- 3DW header is considered for calculation without ECRC – overhead for TLP = 20B
- One ACK assumed per TLP – Overhead for ACK DLLP = 8B
- Update FC DLLPs are not accounted for in the calculation below

Calculations are done independently for each direction in case of a C2S or a S2C DMA engine.

Table 4-1 shows the address mapping of various peripherals used in the TRD.

Table 4-1: **PCIe Performance Calculation**

Transaction	PCIe-TX Overhead	PCIe-RX Overhead	Comment
C2S Only			
Buffer Write (MWr)	20/256 = 320/4096	8/256 = 64/4096	MPS = 256B
BD Fetch (MRd)	20/4096	8/4096	One descriptor defines 4KB
BD Completion (CplD)	8/4096	20/4096	
BD Update (MWr)	20/4096	8/4096	
S2C Only			
BD Fetch (MRd)	20/4096	8/4096	One descriptor defines 4KB
BD Completion (CplD)	8/4096	20/4096	
Buffer Fetch (MRd)	20/512 = 160/4096	8/512 = 64/4096	MRRS = 512B
Buffer Completion (CplD)	8/64 = 512/4096	20/64 = 1280/4096	RCB = 64B
BD Update (MWr)	20/4096	8/4096	

The dominant directions for C2S and S2C DMA are PCIe-TX and PCIe-RX respectively. PCIe receive on C2S implies descriptor fetch and ACK-NAK, hence the overhead is low. PCIe transmit on S2C comprises of descriptor and buffer fetch requests.

Table 4-2 shows aggregating the overhead specific to those directions for S2C and C2S.

Table 4-2: **PCIe Performance Estimate**

Direction	Overhead	Usable Bandwidth	Effective Gen2 Throughput (4 Gb/s per lane per dir)
PCIe-TX (C2S only)	368 bytes for every 4KB	91.76%	3.67 Gb/s per lane
PCIe-RX (S2C only)	1380 bytes for every 4KB	74.8%	2.99 Gb/s per lane
PCIe-TX (S2C + C2S)	948 bytes for every 4KB	81.2%	3.24 Gb/s per lane
PCIe-RX (S2C + C2S)	1544 bytes for every 4KB	72.62%	2.9 Gb/s per lane

For Gen 2 x4, the effective throughput comes out to be 14.68 Gb/s in C2S and 11.962 Gb/s in S2C directions independently.

However, it is to be noted that the estimation above does not consider flow control overheads related to credits.

The S2C engine (which deals with data transmission i.e. reading system memory) issues read requests and receives data through completions. This engine exercises data (actual frame) traffic on PCIe receive link which gives a performance of ~2.99 Gb/s per lane. This resembles the PCIe memory read performance.

The C2S engine (which deals with data reception i.e. writing to system memory) issues write requests. This engine exercises data (actual frame) traffic on PCIe transmit link giving a performance ~3.67 Gb/s per lane. This resembles PCIe memory write performance.

When both C2S and S2C channels are activated at the same time, the effective throughput drops and we see 3.24 Gb/s per lane for PCIe-TX and 2.9 Gb/s per lane for PCIe-RX. [Implying 12.94 Gb/s-PCIe-TX and 11.6 Gb/s-PCIe-RX for x4 Gen2 PCIe link]

From these estimates, the best case efficiency is between 72-80% when traffic is active in both directions. This implies a loss of 20-28%.

A PCIe packet has 20 bytes of TLP overhead, for 256 bytes of data, loss is 20/(256 + 20) = 7% and for 64 bytes of data it is 20/(64 + 20) = 14%. So of the total 20-28% performance loss, 7-14% is due to packet overheads. With smaller frame size (say 64 bytes), the payload will get reduced resulting in loss of 20/(64 + 20) = 24%. This means there will be additional loss of 10-17%. So, the total efficiency for 64 byte packets comes out to be ~60% (rounded off 72-80% efficiency minus additional loss).

Thus, the effective throughput for 64 byte sized packets would be around 2.4Gb/s per lane per dir.

Processing System DDR3 Performance

The PS has a Memory Controller that provides a 32-bit 533MHz DDR3 interface.

This provides a total performance of 32 x 533 x 2 = 34.112 Gb/s.

Considering 60% efficiency of the controller (due to DDR3 overheads like refresh etc), effective bandwidth is ~20 Gb/s.

The TRD aims to show processing of 60 1920 x 1080p video frames per second.

This requires 1920 x 1080 x 4 x 60 = 4 Gb/s of transfer rate in each direction.

The design has 2 VDMA paths- 2 S2MM and 2 MM2S and logiCVC-ML display controller requiring access of DDR3. This requires 4 x 4 + 19.2 Gb/s = 35.2 Gb/s of DDR3 bandwidth. Since this exceeds theoretical calculation of DDR3 bandwidth both logiCVC-ML display and MM2S interface of SOURCE VDMA will not be operating at the same time.

Measuring PCIe Performance

This section shows how performance is measured in the Zynq-7000 PCIe TRD.

It should be noted that PCIe performance is dependent on factors like Maximum Payload Size, Maximum Read Request Size, Read Completion Boundary which are dependent on the systems used. With higher MPS values, performance improves as packet size increases.

Hardware provides the registers listed in Table 4-3 for software to aid performance measurement.

Table 4-3: **Performance Registers in Hardware**

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

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

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

Measuring HP Port Performance

The AXI Performance Monitor can monitor and analyze system behavior on the AXI interface. This core is used in the TRD to measure read and write throughput on AXI slave ports of the PS (HP0 and HP2), which are used to access DDR memory from the PL. The core consists of the AXI4-Lite interface to configure and control the core.

This core is configured to measure the read and write throughput by counting the number of transactions per second. When the configured time interval expires, measured throughput in bytes is loaded into a register and read by the software application.

Performance Observations

This section summarizes the performance measured and the trends seen.

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

Video Demonstration Performance

This section summarizes performance as observed on HP0 and HP2 ports of the PS for various video processing modes on Video Path of the TRD.

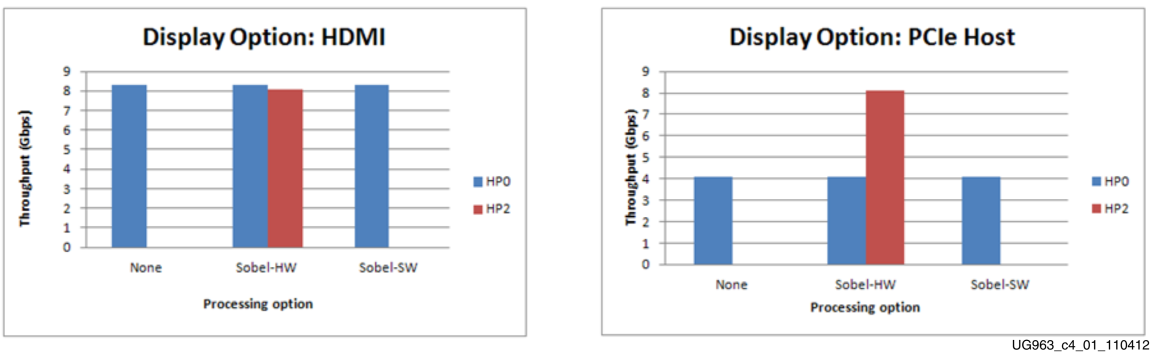


Figure 4-1: **HP Port Performance**

Gen/Check Performance

This section summarizes performance as observed with PCIe-DMA performance mode (GEN/CHK mode) on Data Path of the TRD.

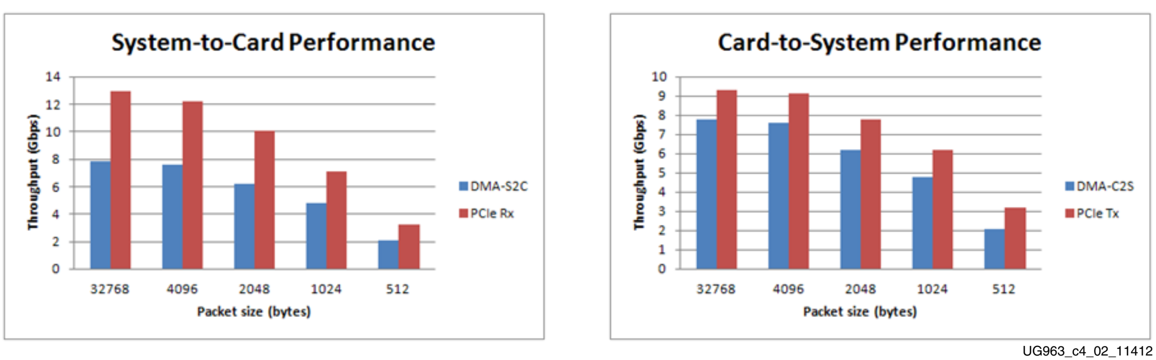


Figure 4-2: **PCIe-DMA Performance**

As can be seen:

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

Designing with the Targeted Reference Design Platform

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

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

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

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

PCIe Host System Software Modifications

This section describes modifications to the platform done directly in the software driver. The same hardware design (BIT/MCS files) works. After any software modification, the code needs to be recompiled. The Linux driver compilation procedure is detailed in the *Zynq-7000 PCIe Targeted Reference Design* wiki page: http://wiki.xilinx.com/zynq-pcie-trd.

Host Software Macro Based Modifications

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

Descriptor Ring Size

The number of descriptors to be set up in the descriptor ring can be defined as a compile time option. To change the size of the buffer descriptor ring used for DMA operations, modify DMA_BD_CNT in `sw/host/xdma/xdma_base.c`.

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

A larger descriptor ring size uses additional memory but improves performance because more descriptors can be queued to hardware.

The DMA_BD_CNT in the driver is set to 1999, increasing this number may not improve performance.

Log Verbosity Level

To control the log verbosity level:

In Linux:

- Add DEBUG_VERBOSE in the makefiles in the provided driver directories - this causes the drivers to generate verbose logs.
- Add DEBUG_NORMAL in the makefiles in the provided driver directories - this cause the drivers to generate informational logs.

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

Driver Mode of Operation

The base DMA driver can be configured to run in either interrupt mode (Legacy or MSI as supported by the system) or in polled mode. Only one mode can be selected. To control the driver:

- Add TH_BH_ISR in the makefile `sw/host/xdma` to run the base DMA driver in interrupt mode.
- Remove the TH_BH_ISR macro to run the base DMA driver in polled mode.

Driver queue depth

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

- Edit macro MAX_BUFF_INFO in `sw\host\driver\xrawdata0\sguser.c`
- Edit macro MAX_BUFF_INFO in `sw\host\driver\xrawdata1\sguser.c`

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

Setting Up Board Communications

This appendix provides a procedure for setting up communications between the ZC706 board and an Intel-processor-based computer running the Windows 7 operating system. For this procedure, the computer must have one USB port to communicate with the ZC706 board

Install the USB UART Drivers

Download and install the *Silicon Laboratories CP210x VCP drivers* on the host computer. The drivers are available for download at no cost from www.silabs.com/Support Documents/Software/CP210x_VCP_Win_XP_S2K3_Vista_7.exe.

Configure the Host Computer COM Port

The Reference design uses a terminal program to communicate between the host computer and the ZC706 board. To configure the host computer COM port for this purpose:

1. Connect the ZC706 board to the host computer and power supply as shown in Figure A-1.

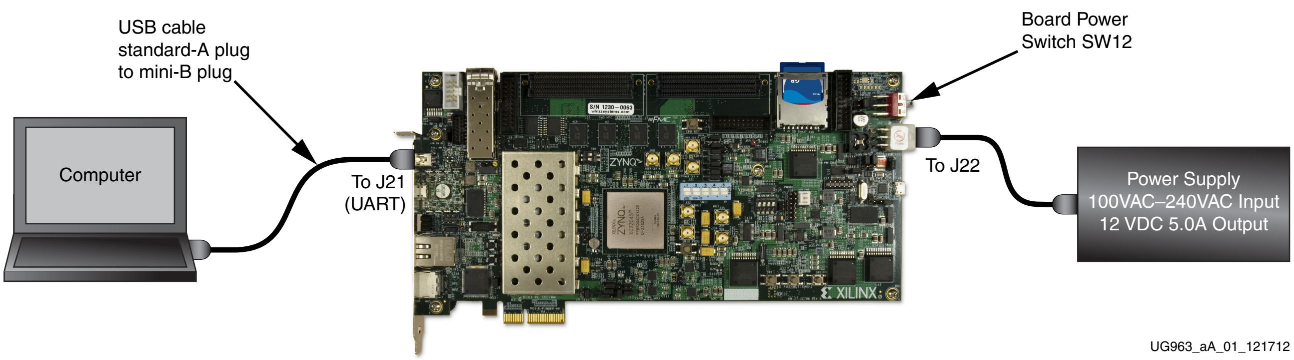


Figure A-1: **Host Computer COM Port Configuration**

2. Turn Board power on (SW12).
3. Open the host computer Device Manager (Figure A-2). In the Windows task bar, Click **Start**, click **Control Panel**, and then click **Device Manager**.

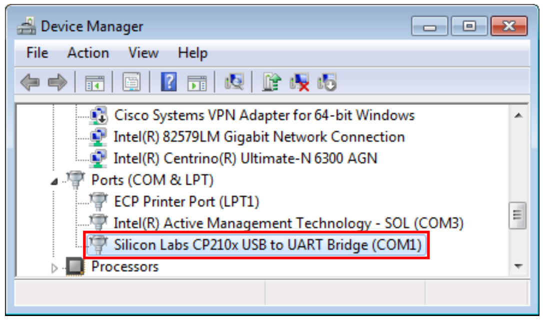


Figure A-2: **Device Manager**

4. Open UART properties. Expand **Ports (COM & LPT)**, right-click **Silicon Labs CP210x USB to UART Bridge**, and then click **Properties**.
5. In the properties window (Figure A-3), select the **Port Settings** tab, verify the settings match the values shown in Figure A-3 and then click **Advanced**.

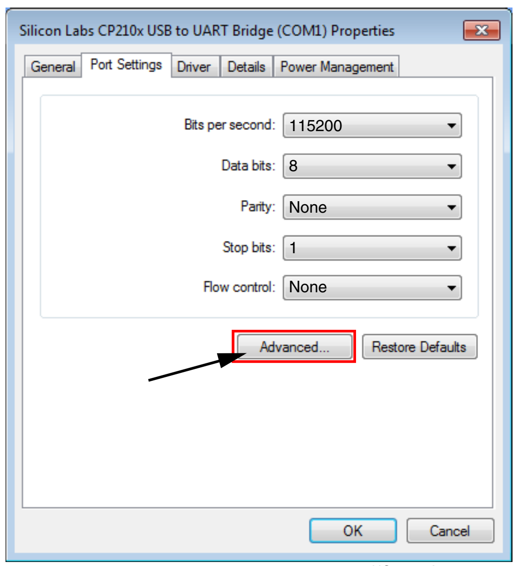
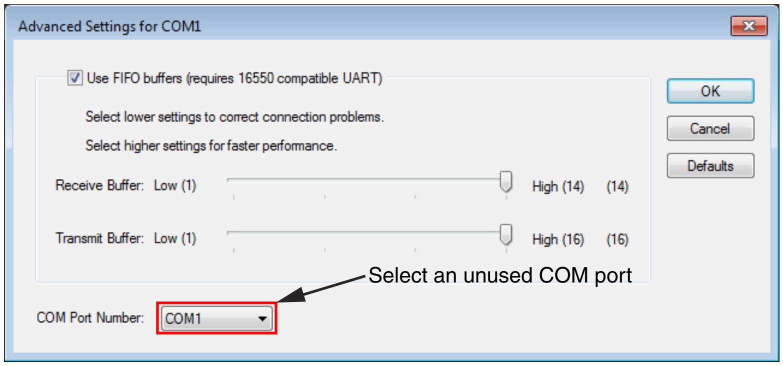


Figure A-3: **Port Settings**

6. Select an unused COM Port Number and then click **OK**. Figure A-4 shows **COM1** as the selected COM port number.



UG963_aA_04_102412

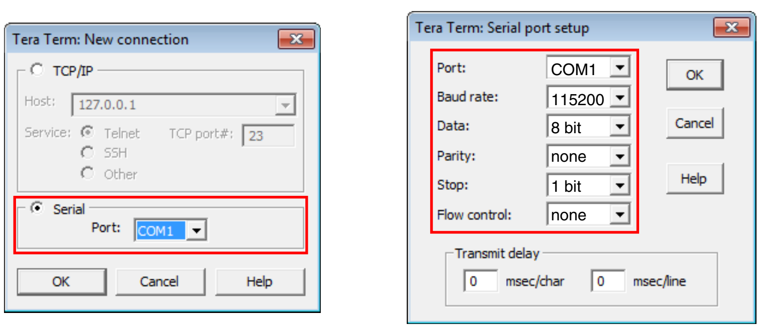
Figure A-4: **Advanced Settings**

7. Click **OK** in the properties window (Figure A-3), then close the Device Manager and the Control Panel.

Install the Terminal Program

Download and install the TeraTerm Pro terminal program on the host computer. TeraTerm Pro is available for download at no cost from http://www.ayera.com/teraterm/.

To communicate with the ZC706 board, configure the New Connection and Serial Port settings as shown in Figure A-5. These settings must match the host computer COM port settings shown in Figure A-3 and Figure A-4.



UG963_aA_05_102412

Figure A-5: **TeraTerm Pro Settings**

Communications setup between the ZC706 board and computer is now complete.

Register Description

This appendix describes the registers most commonly accessed by the host software and the PS software drivers shown in Figure B-1.

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

The hardware registers are mapped to the base address register (BAR0) in PCIe Endpoint. Table B-1 shows the mapping of multiple DMA channel registers across the BAR.

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

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

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

Figure B-1 shows the layout of registers.

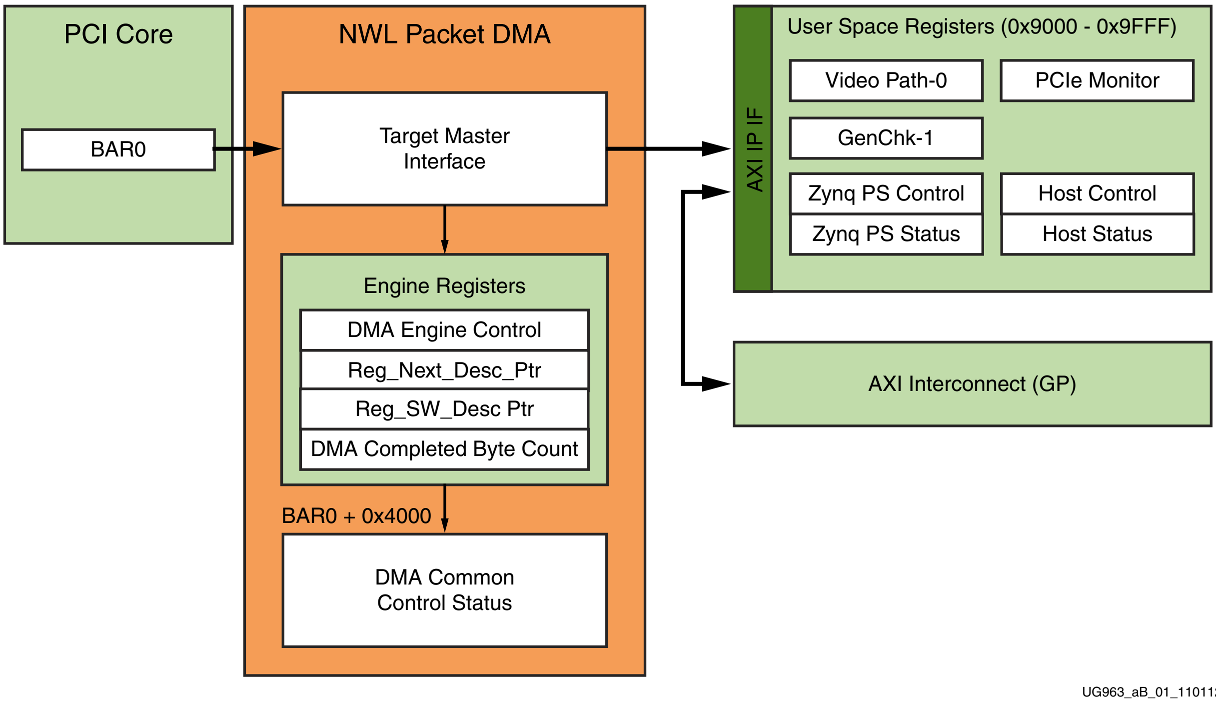


Figure B-1: **Register Map**

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

IP/User Register	Features Controlled/Status Provided	Document Reference
DMA (always BAR0)	• DMA Setup • Performance Statistics	#W LUG v4.12
User Space Registers	• Design Behavior - Loopback mode - Independent traffic generator mode ◦ Independent GEN/CHK mode - Sobel filter ON/OFF mode - HW or SW Sobel processing mode - HDMI display or DMA C2S data transfer mode • PCIe Specific - Statistics	

PCIe DMA Registers

This section describes DMA registers used frequently by the software driver. For a detailed description of all registers available, refer to the NWL DMA user guide.

Channel Specific Registers

The registers described in this section are present in all channels. The address of each register is channel address offset from BAR0 (Refer to Table) plus the register offset.

Engine Control (0x0004)

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

Bit	Field	Mode	Default Value	Description
0	Interrupt Enable	RW	0	Enables interrupt generation
1	Interrupt Active	RW1C	0	Interrupt active is set whenever an interrupt event occurs. Write ‘1’ to clear.
2	Descriptor Complete	RW1C	0	Interrupt active was asserted due to completion of descriptor. This is asserted when descriptor with interrupt on completion bit set is seen.
3	Descriptor Alignment Error	RW1C	0	This causes interrupt when descriptor address is unaligned and that DMA operation is aborted
4	Descriptor Fetch Error	RW1C	0	This causes interrupt when descriptor fetch errors i.e., completion status is not successful
5	SW_Abort_Error	RW1C	0	This is asserted when 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	Assertion of this bit resets the DMA engine and issues a reset to user logic

Next Descriptor Pointer (0x0008)

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

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

Software Descriptor Pointer (0x000C)

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

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

Completed Byte Count (0x001D)

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

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

Host Software Common Registers

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

Common Control & Status (0x4000)

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

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

User Space Registers

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

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

Design version and status Registers

Design Version (0x9000)

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

Bit Position	Mode	Default Value	Description
3:0	RO	0000	Minor version of the design
7:4	RO	0001	Major version of the design
15:8	RO	0100	NWL DMA Version
19:16	RO	0001	Device 0001 – ZC706 board

Transmit Utilization Byte Count (0x900C)

Table B-9: **PCIe Performance Monitor – Transmit Utilization Byte Count Register**

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

Receive Utilization Byte Count (0x9010)

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

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

Upstream Memory Write Byte Count (0x9014)

Table B-11: **PCIe Performance Monitor – Upstream Memory Write Byte Count Register**

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

Downstream Completion Byte Count (0x9018)

Table B-12: **PCIe Performance Monitor – Downstream Completion Byte Count Register**

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

Initial Completion Data Credits for Downstream Port (0x901C)

Table B-13: **PCIe Performance Monitor – Initial Completion Data Credits Register**

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

Initial Completion Header Credits for Downstream Port (0x9020)

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

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

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

Table B-15: **PCIe Performance Monitor – Initial NPD Credits Register**

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

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

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

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

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

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

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

Demonstration Mode: Video Path

This section lists the registers to be configured in performance mode for enabling generator/checker or loopback mode.

PCIe Performance Module #0 Sobel Control Register (Offset- 0x100, Physical-0x40029100)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	Enable Sobel filter mode 1-Sobel filter ON 0-Sobel filter OFF

PCIe Performance Module #0 Offload Control Register (Offset- 0x104, Physical-0x40029104)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	Sobel filter processing in Hardware or PS 1-Sobel processing in PS 0-Sobel processing in PL

PCIe Performance Module #0 Display Control Register (Offset- 0x108, Physical-0x40029108)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	Video data transfer control 1-Transfer video data back to host 0-Display video data through HDMI

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

This lists the registers to be configured in performance mode for enabling generator/checker or loopback mode.

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

PCIe Performance Module #0 Packet Length Register (0x9204)			
Bit Position	**Mode**	**Default Value**	**Description**
15:0	RW	16'd4096	**Packet Length** to be generated. Maximum supported is 32KB size packets. (C2S1)

Module #0 Enable Loopback/Checker Register (0x9208)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	Enable traffic checker – S2C1
1	RW	0	Enable Loopback – S2C1 <-> C2S1

PCIe Performance Module #0 Checker Status Register (0x920C)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW1C	0	**Checker error** – indicates data mismatch when set (S2C1)

PCIe Performance Module #0 Count Wrap Register (0x9210)			
Bit Position	**Mode**	**Default Value**	**Description**
31:0	RW	511	**Wrap Count** – value at which sequence number should wrap around

Host and PS Communication Registers

The Host Control and Status registers are read only registers for PS and PS Control and Status registers are read only registers for host.

Host Control Register (Offset- 0x300, Physical-0x40029300)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	Host soft reset- 1-Soft reset asserted 0-Soft reset de-asserted

Host Status Register (Offset- 0x304, Physical-0x40029304)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	Host ready status
1	RW	0	Test start in host status 1-Test started 0-Test stopped
2	RW	0	Host test error status 1-Error detected 0-No error

PS Control Register (Offset- 0x400, Physical-0x40029400)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	PS soft reset- 1-Soft reset asserted 0-Soft reset de-asserted

PS Register (Offset- 0x404, Physical-0x40029404)			
Bit Position	**Mode**	**Default Value**	**Description**
0	RW	0	PS ready status
1	RW	0	TPG DMA error status 1-Error detected 0-No error

Troubleshooting

This appendix provides some troubleshooting suggestions to try when things do not work as expected.

Table C-1 is based on these assumptions:

- User has followed instructions as explained in Getting Started chapter.
- User has made sure that PCIe link is up and the endpoint device is discovered by the host and can be seen with lspci
- Visual indicators (LEDs) as listed are up as per the functionality

Table C-1: **Troubleshooting Tips**

Problem	Possible Resolution
Performance is low	Check if the design linked at x4 5Gb/s rate
Test does not start in an installed F-16 installed OS on Intel motherboard on PCIe host system.	Check dmesg command if user is getting nommu_map_single then user can bring up by followings ways. • If OS is installed on hard disk user can edit `/etc/grub2.cfg`, add mem = 2g to kernel options. • If its live CD stop at Live CD boot up prompt and add mem = 2g to kernel boot up options.
Not able to install drivers	Error message pops up when trying to install if there is some problem in installation. Popup message will mention the reason but user can select View Log option for detailed analysis. This will create and open driver_log file.

Appendix D

Additional Resources

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

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

Create a Xilinx user account and sign up to receive automatic e-mail notification whenever this document is updated.

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

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

www.xilinx.com/company/terms.htm.

Solution Centers

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

Further Resources

The most up to date information related to the ZC706 board and its documentation is available on the following websites.

The ZC706 Evaluation Kit Kit product page:

http://www.xilinx.com/products/boards/zc706

These Xilinx documents provide supplemental material useful with this guide:

DS190, *Zynq-7000 Extensible Processing Platform Overview*

DS406, *LogiCORE IP Processor System Reset Module Product Specification*

DS768, *LogiCORE IP AXI Interconnect*

PG020, *LogiCORE IP AXI Video Direct Memory Access Product Guide*

PG054, *7 Series FPGAs Integrated Block for PCI Express User Guide*

UG585, *Zynq-7000 Extensible Processing Platform Technical Reference Manual*

UG673, *Quick Front-to-Back Overview Tutorial: PlanAhead Design Tool*

UG798, *Xilinx Design Tools: Installation and Licensing Guide*

UG821, *Zynq-7000 EPP Software Developers Guide*

UG873, *Zynq Concepts, Tools, and Techniques Guide*

UG882, *Kintex-7 FPGA Base Targeted Reference Design User Guide*

UG883, *Kintex-7 FPGA Base Targeted Reference Design Getting Started Guide*

UG925, *Zynq-7000 EPP ZC702 Base Targeted Reference Design User Guide*

UG926, *Zynq-7000 EPP ZC702 Evaluation Kit Getting Started Guide*

References

The following websites provide supplemental material useful with this guide:

1. Using Git:
   http://wiki.xilinx.com/using-git
2. git: the fast version control system home page:
   http://git-scm.com/
3. Zynq Linux: Downloading the Kernel Tree:
   http://xilinx.wikidot.com/zynq-linux#toc7
4. Zynq Linux: Configuring and Building the Linux Kernel:
   http://xilinx.wikidot.com/zynq-linux#toc8
5. Xilinx Open Source Linux:
   http://wiki.xilinx.com/open-source-linux
6. Xilinx Device Tree Generator:
   http://xilinx.wikidot.com/device-tree-generator
7. Device Tree general information:
   http://devicetree.org/Main_Page
8. AMBA AXI4-Stream Protocol Specification:
   http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html

9. PCI-SIG Documentation:
http://www.pcisig.com/specifications
10. PG020, *LogiCORE IP AXI Video Direct Memory Access Product Guide*
11. Xilinx Open Source ARM git Repository: http://git.xilinx.com/
12. Xilinx ARM GNU Tools: http://wiki.xilinx.com/zynq-tools
13. Using Git: http://wiki.xilinx.com/using-git
14. git: the fast version control system home page: http://git-scm.com/
15. Zynq Linux: Downloading the Kernel Tree: http://xilinx.wikidot.com/zynq-linux#toc7
16. Zynq Linux: Configuring and Building the Linux Kernel:
http://xilinx.wikidot.com/zynq-linux#toc8
17. Xilinx Open Source Linux: http://wiki.xilinx.com/open-source-linux
18. Xilinx Device Tree Generator: http://xilinx.wikidot.com/device-tree-generator
19. Device Tree general information: http://devicetree.org/Main_Page
20. AMBA AXI4-Stream Protocol Specification:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html
21. PCI-SIG Documentation: http://www.pcisig.com/specifications
22. Xilinx PlanAhead Design and Analysis Tool website:
http://www.xilinx.com/tools/planahead.htm
23. Xylon IP Cores - logiCVC-ML Compact Multilayer Video Controller description:
www.logicbricks.com/Products/logiCVC-ML.aspx
24. Qt Online Reference Documentation. Qt is a toolkit for creating GUIs:
http://doc.qt.nokia.com/
25. logiCVC-ML Compact Multilayer Video Controller Data Sheet:
http://www.logicbricks.com/Documentation/Datasheets/IP/logiCVC-ML_hds.pdf
26. Silicon Labs CP210x USB to UART Bridge VCP Drivers:
http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx
27. Northwest Logic DMA back-end core: http://www.nwlogic.com/packetdma/
28. Fedora project: http://fedoraproject.org