

Report of Industrial Training

at

Lightspeed Photonics private limited

Unit No. 203, SBR CV Towers Sector-I, SY No64, Huda

Techno Enclave,

Madhapur Hyderabad,

Telangana 500081 India

In partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

Submitted by

Student Name: Ankush Verma

Register Number: 200907058



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY

(A Constituent institute of MAHE, Manipal)

MANIPAL – 576104, KARNATAKA, INDIA

August 2023

ACKNOWLEDGEMENT

The internship experience I had with LIGHTSPEED PHOTONICS PRIVATE LIMITED was a tremendous opportunity for learning and professional growth. I consider myself fortunate to have been given the chance to be a part of such a valuable experience. I am also grateful for the opportunity to have met many exceptional individuals and professionals who guided me throughout the internship period.

I would like to extend my sincere thanks to Mr. JUGALKISHORE BHANDARI, the esteemed System Architect/FPGA Engineer, for his valuable input, advice, and guidance during this time. I am profoundly grateful for his contributions.

I would also like to express my heartfelt appreciation to Mrs. DIVYA SRI R for her careful and invaluable guidance, which enriched my theoretical and practical knowledge. My gratitude also extends to the officials and staff members of Bharat Electronics Limited for their assistance during my internship.

This experience marks a significant milestone in my career development, and I am determined to apply the skills and knowledge I gained in the best viable way. I will continue to work on further improving myself to achieve my career objectives.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION	1
1. INTEL ARRIA 10 FPGA	1
1.1 Application Areas	1
1.2 Advantages of Arria 10 FPGA	1
1.3 Overview of Arria 10 FPGA Block Diagram I/O Pins	2
2. INTEL QUARTUS PRIME SOFTWARE	3
2.1 Key Features and Tools	3
CHAPTER 2 : LINUX OS	4
1. INTRODUCTION TO LINUX OS	4
2. SOME IMPORTANT LINUX COMMANDS FOR THE PROJECT	4
CHAPTER 3: PCIE PROTOCOLS	6
1. PCI EXPRESS AND SLOTS	6
2. PCIE COMMUNICATION STACK	8
2.1 PCIE Communication Stack	8
3. PCIE SWITCHES	10
3.1 Introduction	10
3.2 PCI Express Gen 3 Packet Switch Board	11
CHAPTER 4: PCIE IP	12
1. IP Catalog and IP Compilation in Quartus Prime	12
2. PCIE IP	13
2.1 Introduction	13
2.2 PCIE IP Parameters Settings:	14
CHAPTER 5: FPGA CVP	16
1. INTRODUCTION	16
1.1 Benefits of Using CvP:	16
2. CVP SYSTEM	17
CHAPTER 6: SIGNAL TAP LOGIC ANALYZER	18
1. INTRODUCTION	18
2. SIGNAL TAP LOGIC ANALYZER GUI	19

2.1 Signal Tap Logic Analyzer and Simulator Integration.....	19
3. SIGNAL TAP LOGIC ANALYZER CAPABILITIES	20
4. SIGNAL TAP DEBUGGING FLOW	21
5. SIGNAL TAP FPGA IP PARAMETERS	22
CHAPTER 7: PCIE-BASED DATA TRANSFER BETWEEN HOST PC AND ARRIA 10 FPGA.....	23
1. INRODUCTION	23
1.1 Xillybus:	23
1.2 LighSiP:.....	27
2. OBJECTIVES	29
3. SCOPE OF THE PROJECT:	30
4. DESIGN DESCRIPTION:	31
4.1 Downloading the Demo Bundle and Software:	31
4.2 Creating the Bitstream for Intel Arria 10 FPGA:	32
4.3 Checking for Prerequisites:	33
4.4 Compilation of the Kernel Module:.....	34
4.5 Installing the Kernel Module:	35
4.7 The “HELLO, WORLD” test:	37
5. CONCLUSION:	40

CHAPTER 1: INTRODUCTION

1. INTEL ARRIA 10 FPGA

Intel Arria 10 FPGA is a powerful programmable logic device with versatile applications. Its advanced architecture and high performance make it popular across industries for various tasks. Arria 10 FPGA family offers up to 1.15 million logic elements, high-speed transceivers, and hardened floating-point processing units. It supports multiple I/O standards, making it suitable for interfacing with various devices and systems.

1.1 Application Areas

- **Telecommunications and Networking:** Arria 10 FPGA is used in switches, routers, baseband processing, and wireless communication systems for real-time data processing and low-latency communication.
- **Data Centres and Cloud Computing:** It accelerates algorithms and computation-intensive tasks in data centres, enhancing performance and power efficiency.
- **High-Performance Computing (HPC):** Arria 10 FPGA accelerates scientific simulations, cryptography, and data analytics in HPC applications.
- **Video and Image Processing:** It finds applications in video transcoding, image recognition, and real-time video analytics.
- **Industrial Automation:** Arria 10 FPGA is utilized in control systems, motor drives, and sensor interfacing for efficient automation.
- **Aerospace and Defence:** It is used in radar processing, software-defined radios, secure communications, and digital signal processing.

1.2 Advantages of Arria 10 FPGA

- **Reconfigurability:** The FPGA can be reprogrammed in the field, offering flexibility and adaptability.
- **Parallel Processing:** Parallel processing enables faster execution of multiple tasks.
- **Low Power Consumption:** Arria 10 FPGA is power-efficient, making it suitable for power-constrained applications.
- **High Bandwidth:** It supports high-speed data transfer and communication.
- **Customizability:** The FPGA's programmability allows hardware customization for optimized solutions

1.3 Overview of Arria 10 FPGA Block Diagram I/O Pins

- Altera FMCA/FMCB: High-speed connectors for interfacing with external FMC/FMC+ modules, expanding FPGA capabilities.
- Hilo External Memory: Interface for connecting external memory devices to the FPGA.
- PCI Express: High-speed serial interface for communication with PCIe devices.
- MaxV CPLD: Configurable logic for additional control and management functions.
- On-board USB Blaster with Micro USB: USB Blaster for programming and debugging the FPGA via Micro USB.
- Buttons/Switches: User input elements for control and configuration.
- LEDs: Visual indicators for status and debugging purposes.
- QSFP/SFP+: High-speed transceiver connectors for data communication.
- 128MB Flash: On-board flash memory for non-volatile data storage.
- EPCQ: Configuration device for storing FPGA configurations.
- Display Port: Interface for connecting displays to the FPGA.
- SMA XCVR Out: SMA connector for high-speed transceiver output.
- SMA CLK Out: SMA connector for clock signal output.

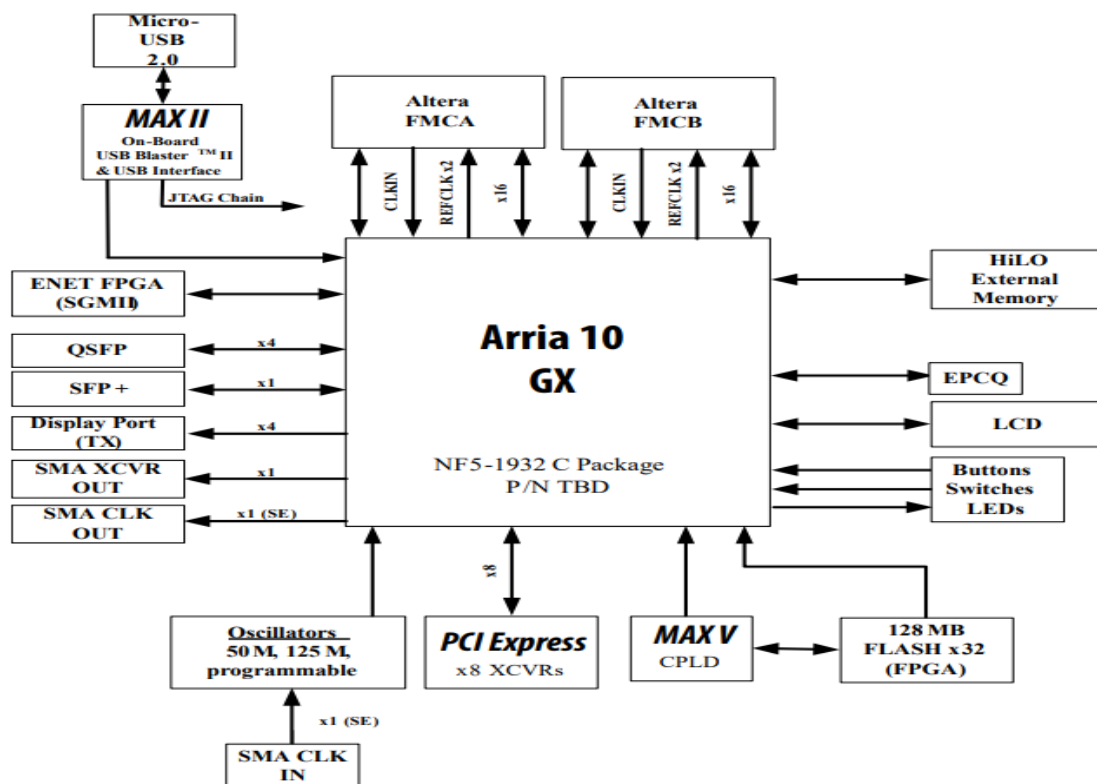


Figure 1.1 Arria 10 Gx Block Diagram

2. INTEL QUARTUS PRIME SOFTWARE

Intel Quartus Prime is a powerful and widely used software suite for designing, implementing, and programming Field-Programmable Gate Arrays (FPGAs). Developed by Intel, formerly Altera, the Quartus Prime software provides an integrated development environment (IDE) that simplifies the complex process of FPGA design and verification.

Intel Quartus Prime offers a comprehensive set of tools and features to support the entire FPGA development workflow, from design entry to programming. The software enables engineers and designers to create customized digital circuits and systems tailored to specific applications, making it a popular choice across various industries.

2.1 Key Features and Tools

- **Design Entry:** Quartus Prime supports Hardware Description Languages (HDLs) such as VHDL and Verilog for capturing the hardware design. Additionally, graphical design entry tools facilitate ease of use and faster prototyping.
- **IP Cores:** The software provides a vast library of pre-designed Intellectual Property (IP) cores, allowing designers to quickly integrate complex functionalities into their designs.
- **Synthesis and Optimization:** Quartus Prime Compiler synthesizes the HDL code into a netlist representation of the design. It optimizes the logic and resource utilization to achieve the desired performance and efficiency.
- **Platform Designer (formerly Qsys):** Platform Designer allows designers to create complex systems by interconnecting various IP cores and peripherals in a graphical user interface.
- **Timing Analysis:** Timing Analyzer performs static timing analysis to ensure that the design meets the specified timing constraints and operates reliably at the desired clock frequencies.
- **Signal Tap Logic Analyzer:** Signal Tap enables designers to capture and analyse internal signals within the FPGA during runtime, aiding in debugging and verification.
- **Model Sim-Intel FPGA Edition:** Integrated Model Sim simulator allows functional verification and debugging of the FPGA design.
- **Power Analysis:** Power Analyzer estimates the power consumption of the design, helping optimize power efficiency.
- **Programming and Configuration:** Quartus Prime Programmer is used to configure and program the FPGA with the generated bitstream files, enabling the FPGA to execute the desired functionality.
- **Enhanced Design Security:** Intel Quartus Prime incorporates advanced security features to safeguard FPGA designs against unauthorized access and tampering. It supports design encryption and device programming protection.

CHAPTER 2 : LINUX OS

1. INTRODUCTION TO LINUX OS

Linux OS, often referred to simply as Linux, is an open-source operating system based on the Linux kernel. It is a Unix-like operating system that was initially developed by Linus Torvalds in 1991 and has since grown into a powerful and widely used platform.

Linux is known for its stability, security, flexibility, and the ability to be customized to suit various needs. Unlike proprietary operating systems such as Windows or macOS, Linux is distributed under various free and open-source licenses, which allows users to access and modify the source code, making it a popular choice among developers and tech enthusiasts.

One of the defining features of Linux is its modular design, which allows users to choose from various software packages, graphical desktop environments, and other components to create a tailored system that meets their requirements.

Linux is used in a wide range of applications, from running servers, cloud computing, and supercomputers, to being the foundation of many embedded systems, smartphones, and IoT devices. Popular Linux distributions, such as Ubuntu, Fedora, Debian, CentOS, and Arch Linux, cater to different user needs and preferences.

The collaborative nature of Linux development, with contributions from individuals and organizations worldwide, has led to a robust and continuously evolving ecosystem. Its versatility, performance, and cost-effectiveness have contributed to its widespread adoption across industries and user communities.

2. SOME IMPORTANT LINUX COMMANDS FOR THE PROJECT

- cp – Copy file or files.
- rm – Remove file or files
- mv – Move file or files.
- rmdir – Remove directory
- ls – List all files in the current directory (or another one, when specified).
- lspci – List all PCI (and PCIe) devices on the bus.
- cd – Change directory
- pwd – Show current directory
- cat – Send the file (or files) to standard output. Or use standard input if no argument is given.

- `man` – Show the manual page on a certain command.
- `less` – terminal pager. Shows a file or data from standard input page by page.
- `head` – Show the beginning of a file.
- `tail` – Show the end of a file.
- `diff` – compare two text files. If it says nothing, files are identical.
- `cmp` – compare two binary files. If it says nothing, files are identical.
- `grep` – Search for a pattern in a file or standard input.
- `find` – Find a file depending on their name, age, type, or anything of which you can think.

```

File Edit View Search Terminal Help
lsai_nikhil@server1:~$ cd ankushverma
lsai_nikhil@server1:~/ankushverma$ cat > newfilecreated
content of newfilecreated
lsai_nikhil@server1:~/ankushverma$ ls emptyfolder
lsai_nikhil@server1:~/ankushverma$ cp newfilecreated emptyfolder
lsai_nikhil@server1:~/ankushverma$ ls emptyfolder
newfilecreated
lsai_nikhil@server1:~/ankushverma$ ls
adding          counter.v.bak  newfilecreated      pciecvp
addingclock     designexample newmultiplyadd      pcietest
addnewclock     emptyfolder   newmultiplyadd.ip   project
ankush.txt      emptyfolder   NEW.qpf              qdb
avalonsti1.ip  IPsimulate    NEW.qsf              synth_dumps
checkpcielastr logictapi     NEW.qws              testpcie
CHECKPCIELAST  macc         not1                 tmp-clearbox
checkPIPE       macc.ip      now                  trial.qpf
collgePCIE     mac.qpf      output_files        trial.qsf
counter.qpf    mac.qsf      pcie123              trial.qws
counter.qsf    mac.qws      pcie_a10_hip_0_example_design VERIFY
counter.qws    multiply     pcieavalon           VERIFY.ip
counter.v      multiply.ip   pciecore             xillybus
lsai_nikhil@server1:~/ankushverma$ mv newfilecread renamedfile
mv: cannot stat 'newfilecread': No such file or directory
lsai_nikhil@server1:~/ankushverma$ mv newfilecreated renamedfile
lsai_nikhil@server1:~/ankushverma$ rmdir now
lsai_nikhil@server1:~/ankushverma$ pwd
/home/lsai_nikhil/ankushverma
lsai_nikhil@server1:~/ankushverma$ ls
adding          counter.v.bak  newmultiplyadd      project
addingclock     designexample newmultiplyadd.ip   qdb
addnewclock     emptyfolder   NEW.qpf              renamedfile
ankush.txt      emptyfolder   NEW.qsf              synth_dumps
avalonsti1.ip  IPsimulate    NEW.qws              testpcie
checkpcielastr logictapi     not1                 tmp-clearbox
CHECKPCIELAST  macc         output_files        trial.qpf
checkPIPE       macc.ip      pcie123              trial.qsf
collgePCIE     mac.qpf      pcie_a10_hip_0_example_design trial.qws
counter.qpf    mac.qsf      pcieavalon           VERIFY

```

Figure 2.1 Executing Some Linux Commands

CHAPTER 3: PCIE PROTOCOLS

1. PCI EXPRESS AND SLOTS

PCIe, also known as Peripheral Component Interconnect Express, is a serial expansion bus standard employed to link a computer with one or multiple peripheral devices.

In contrast to parallel busses like PCI and PCI-X, PCIe offers superior data transfer rates and lower latency. Each peripheral device connected to the motherboard using PCIe enjoys its dedicated point-to-point connection, eliminating bandwidth competition since they are not sharing a common bus.

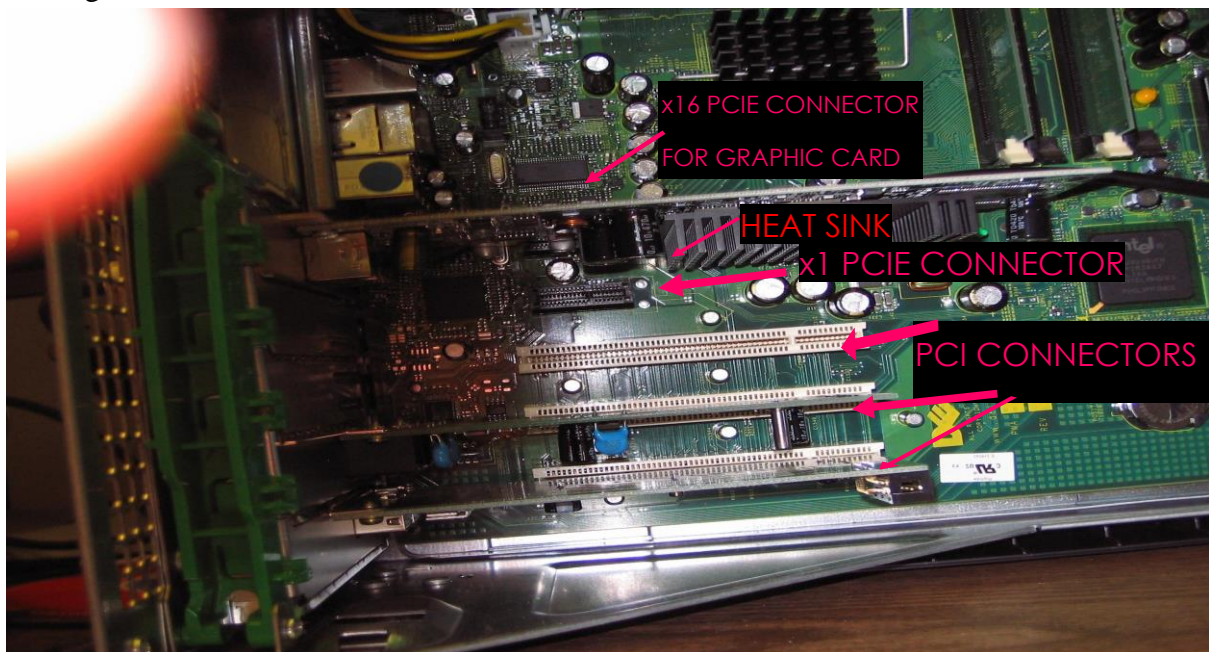


Figure 3.1 PCIe Slots In Motherboard

Various high-performance peripherals, including graphics adapter cards, network interface cards (NICs), and storage accelerator devices, utilize PCIe for efficient data transfer.

The data transfer process in PCIe involves two sets of signal pairs, consisting of two wires for transmitting and two for receiving. Each of these signal pairs is termed a "lane," with each lane simultaneously transmitting and receiving eight-bit data packets between two points.

PCIe's scalability ranges from one to 32 separate lanes, with common deployments using 1, 4, 8, 12, 16, or 32 lanes. The number of lanes in a PCIe card directly influences its performance and, consequently, its price. For instance, less expensive PCIe devices, such as NICs, might use only four lanes (PCIe x4), whereas high-performance graphics adapters may employ 32 lanes (PCIe x32) for top-speed transmission, making them pricier.

PCIe bus slots generally exhibit backward compatibility with other PCIe bus slots, allowing interfaces with fewer lanes to connect to those with more lanes. For example, a PCIe x8 card can fit into a PCIe x16 slot. However, PCIe bus slots are not backward compatible with connection interfaces for older bus standards.

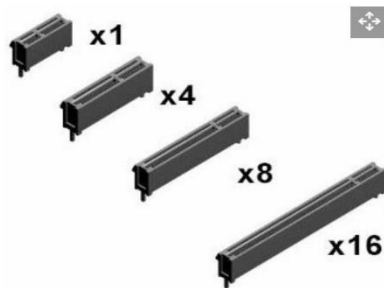


Figure 3.2 Types of PCIE Slots

PCIe Generations Compared

	Bandwidth	Gigatransfer	Frequency
PCIe 1.0	8 GB/s	2.5 GT/s	2.5 GHz
PCIe 2.0	16 GB/s	5 GT/s	5 GHz
PCIe 3.0	32 GB/s	8 GT/s	8 GHz
PCIe 4.0	64 GB/s	16 GT/s	16 GHz
PCIe 5.0	128 GB/s	32 GT/s	32 GHz
PCIe 6.0	256 GB/s	64 GT/s	32 GHz

Figure 3.3 PCIE Generations

In data centres, PCIe enables high-speed networking across server backplanes, facilitating connections to Gigabit Ethernet, RAID, and InfiniBand networking technologies beyond the server rack. Moreover, the PCIe bus serves as an interconnection for clustered computers employing Hyper Transport.

For laptops and mobile devices, mini-PCI-e cards prove useful in connecting wireless adaptors, solid-state storage devices, and other performance-enhancing components. Additionally, External PCI Express (ePCIe) allows the motherboard to link to an external PCIe interface, typically used when the computer requires an unusually high number of PCIe ports.

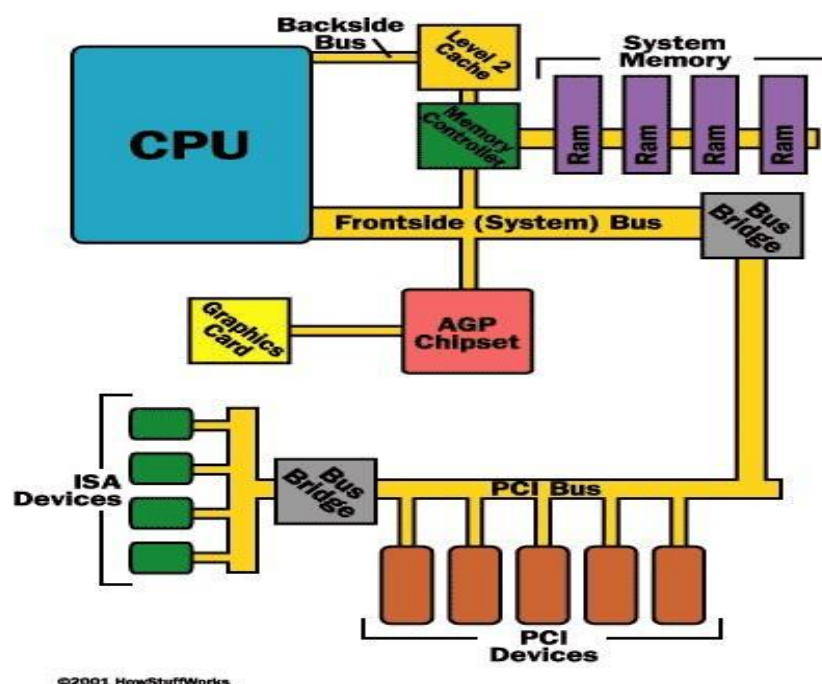


Figure 3.4 PCIE System Architecture

2. PCIE COMMUNICATION STACK

2.1 PCIE Communication Stack

It is a layered architecture responsible for managing data transfer and communication between devices using the PCIe (Peripheral Component Interconnect Express) standard. It comprises of three layers, each with distinct functionalities.

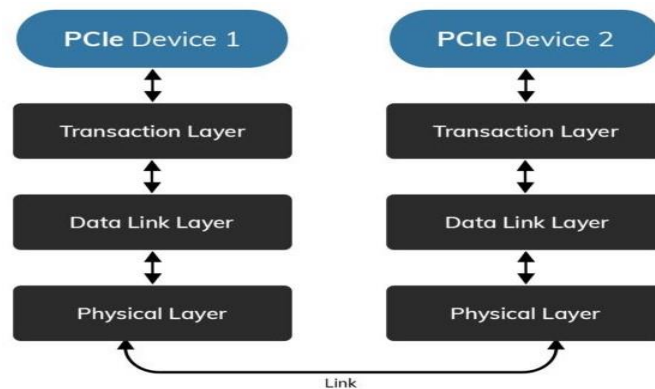


Figure 3.5 PCIe Protocol Stack

1. Physical Layer (PHY)

The Physical Layer is the foundation of the PCIe communication stack. It deals with the actual transmission of electrical signals over the PCIe bus. It finds the physical connections, voltage levels, and clocking mechanisms used to transmit and receive data between devices. This layer is responsible for ensuring reliable and accurate data transmission across the PCIe bus.

2. Data Link Layer (DLL)

The Data Link Layer sits above the Physical Layer and is crucial for reliable data transfer between devices. It handles error detection and correction, flow control, and sequence management. Within the Data Link Layer, we have two sub-layers:

- Logical Link Control and Status Register (LLC)
The LLC manages link initialization, configuration, and status reporting. It handles the establishment of connections between devices and ensures that both ends are ready for communication.
- Transaction Layer Packet (TLP)
The TLP is responsible for packetizing the data and headers for efficient data transfer. It encapsulates the payload (data) and necessary control information to route and identify the data within the PCIe bus.

3. Transaction Layer (TL)

The Transaction Layer is the highest layer in this simplified explanation. It is responsible for managing transaction-oriented communication between devices. It abstracts the data into Transaction Layer Packets (TLPs) and handles different types of transactions, such as memory reads and writes. The Transaction Layer ensures the right data is sent to the appropriate destination and manages the communication between the devices.

2.2 Packet Format in PCIE Communication

In the PCIe (Peripheral Component Interconnect Express) communication standard, data is organized into packets and transmitted across the bus. The packet format varies depending on the specific layer within the PCIe communication stack.

1. Transaction Layer (TL) - Transaction Layer Packet (TLP)

At the Transaction Layer, the packet format used is known as a Transaction Layer Packet (TLP). TLPs carry essential information about the type of transaction being executed, such as memory R/W operations. They also include the address of the data being accessed and amount of data to be transferred. TLPs are responsible for managing data exchanges between devices.

2. Data Link Layer (DLL) - Data Link Layer Packet (DLLP)

In the Data Link Layer, the packet format utilized is called a Data Link Layer Packet (DLLP). DLLPs are used to manage data flow between the sender and receiver. They contain vital information such as sequence numbers and flow control details, ensuring that data transmission is smooth and reliable.

3. Physical Layer (PHY) - Physical Layer Packet (PLP)

At the Physical Layer, the packet format used is referred to as a Physical Layer Packet (PLP). PLPs are responsible for transmitting TLPs or DLLPs across the PCIe link. They carry information such as the packet type, lane number, and data payload, facilitating the reliable transmission of data between devices.

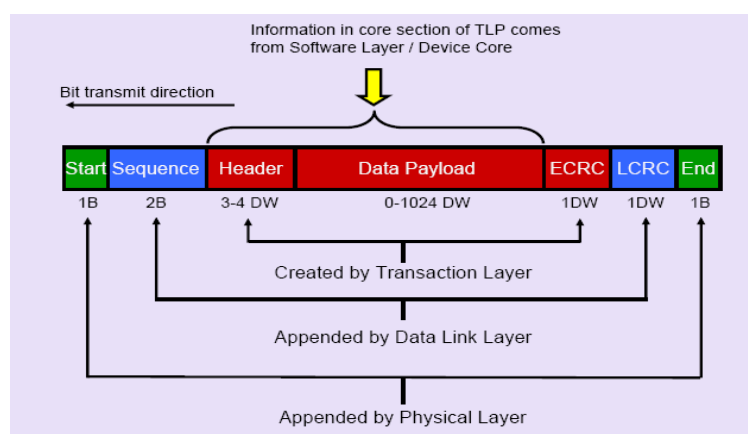


Figure 3.6 Encapsulation of Packets In PCIE

3. PCIE SWITCHES

3.1 Introduction

A PCIe switch is a specialized component that acts as a central hub, connecting multiple PCIe devices and allowing them to communicate with each other. It operates at the Data Link Layer (DLL) and Transaction Layer (TL) of the PCIe communication stack.

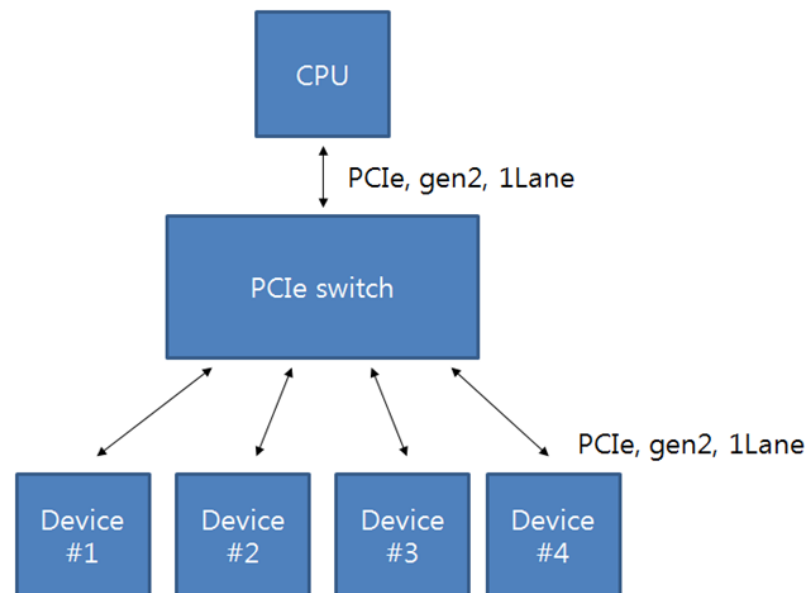


Figure 3.7 PCIe 4:1 Switch

The primary functions of a PCIe switch includes:

- **Connectivity:** PCIe switches provide multiple downstream ports, each capable of connecting to PCIe devices, such as graphics cards, network cards, storage devices, and other expansion cards.
- **Routing:** The switch intelligently routes data packets between connected devices based on their addresses and transaction types. It ensures efficient data flow and minimizes contention for the PCIe bus.
- **Bandwidth Aggregation:** PCIe switches aggregate the bandwidth of multiple PCIe lanes, allowing devices connected to different lanes to communicate with each other without bottlenecking.
- **Scalability:** By allowing multiple devices to connect to the switch, the overall system's scalability and expandability are enhanced. This is particularly beneficial in high-performance computing and server environments.
- **Peer-to-Peer Communication:** PCIe switches enable direct peer-to-peer communication between devices without involving the CPU or memory. This feature improves data transfer efficiency for specific workloads.

3.2 PCI Express Gen 3 Packet Switch Board

A PCIe Gen3 switch board is a specialized hardware component designed to enable efficient and high-speed data communication between various PCIe devices within a computer system. Based on the PCIe Gen3 standard, this switch board facilitates data routing, aggregation, and scalability, enhancing the overall performance and expandability of modern computing environments.

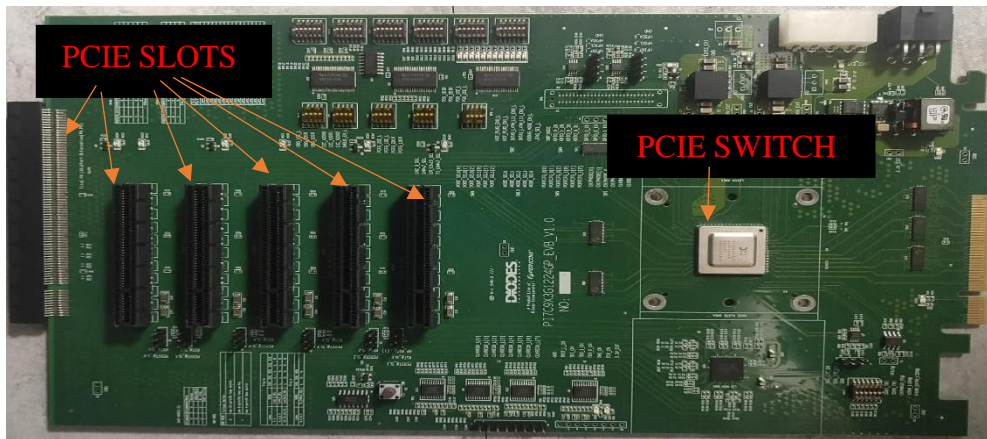


Figure 3.8 PI7C9X3G1224GP Gen 3 PCIe Switch Board

The PCIe Gen3 switch board supports three distinct modes of operation, each serving specific communication needs:

1. Base Mode (Fan-out Mode)

In Base Mode, the switch board operates as a Fan-out switch, featuring one upstream port and up to 15 downstream ports. Multiple virtual PCI-to-PCI bridges are connected via a virtual PCI bus within the switch. This mode enables the distribution of data from one upstream device to multiple downstream devices, allowing efficient communication across the interconnected devices.

2. Switch Partition Mode (Fan-out Mode)

The Switch Partition Mode further enhances flexibility by allowing the switch board to be partitioned into two independent virtual switches. Each virtual switch possesses its own dedicated PCI Express hierarchy, and they do not share ports. With two upstream ports, each attached to a separate Root Complex, this mode enables distinct and isolated communication paths between different sets of devices.

3. Cross-Domain End-Point Mode

The switch board supports a Cross-Domain Endpoint (CDEP) mode, which permits the connection of multiple hosts through the PI7C9X3G1224GP. By configuring one downstream port as a CDEP port, additional hosts can connect, facilitating seamless data exchange between different hosts through the switch board.

CHAPTER 4: PCIE IP

1. IP Catalog and IP Compilation in Quartus Prime

In Quartus Prime software, an IP catalog refers to a collection of pre-designed and pre-verified Intellectual Property (IP) cores that can be readily used in FPGA (Field-Programmable Gate Array) designs. An IP core is a reusable hardware module or function that is designed to perform specific tasks or operations. Using IP cores from the IP catalog simplifies the design process, reduces development time, and ensures reliable functionality.

1. IP Catalog:

The IP catalog in Quartus Prime contains a wide range of IP cores developed by Intel (formerly Altera) and third-party vendors. These IP cores cover various functionalities, such as arithmetic operations, memory controllers, communication protocols (Ethernet, USB, PCIe), digital signal processing (DSP), video processing, and more. By utilizing IP cores from the catalog, designers can integrate complex functionalities into their FPGA designs without having to implement the entire functionality from scratch.

2. IP Compilation in Quartus Prime:

IP cores in the Quartus Prime IP catalog are provided as encrypted RTL (Register Transfer Level) files, which contain the design description in a hardware description language like VHDL or Verilog. When an IP core is used in a design, the Quartus Prime software performs IP compilation, which includes several essential steps:

- **Importing the IP Core:** The designer selects the desired IP core from the IP catalog and imports it into their Quartus Prime project.
- **IP Core Configuration:** The designer configures the IP core based on the specific requirements of the design. Configuration parameters may include clock frequencies, data widths, and other settings.
- **Synthesis:** The Quartus Prime software synthesizes the IP core's RTL code to generate a gate-level netlist. This process involves converting the RTL code into specific FPGA logic elements and interconnections.
- **Placement and Routing:** The synthesized netlist, along with other design components, undergoes placement and routing to determine the physical location of logic elements within the FPGA and establish the connections between them.
- **Design Generation:** After successful compilation, the Quartus Prime software generates programming files (e.g., .sof or .pof files) that can be used to program the FPGA with the configured design.

2. PCIE IP

2.1 Introduction

PCIe IP (Peripheral Component Interconnect Express Intellectual Property) refers to a pre-designed and pre-verified hardware module or core provided by semiconductor manufacturers, such as Intel (formerly Altera) or Xilinx, for implementing the PCIe (PCI Express) communication protocol in FPGA (Field-Programmable Gate Array) designs.

PCIe IP cores are used to integrate PCIe functionality into FPGA designs, enabling communication and data exchange between the FPGA and other PCIe devices in a computer system. These IP cores abstract the complex details of the PCIe protocol, making it easier for designers to incorporate high-speed data transfer capabilities into their FPGA-based apps.

By utilizing PCIe IP cores, FPGA designers can take advantage of the advanced features of the PCIe standard, such as high data transfer rates, low latency, and point-to-point communication between devices. This allows for efficient and reliable communication between the FPGA and peripherals like graphics cards, storage devices.

PCIe IP cores are typically offered with various configuration options, allowing designers to tailor the core to their specific design requirements, including lane width, data rate, and interface type. These IP cores are thoroughly tested and verified, ensuring reliable and robust operation in FPGA-based systems.

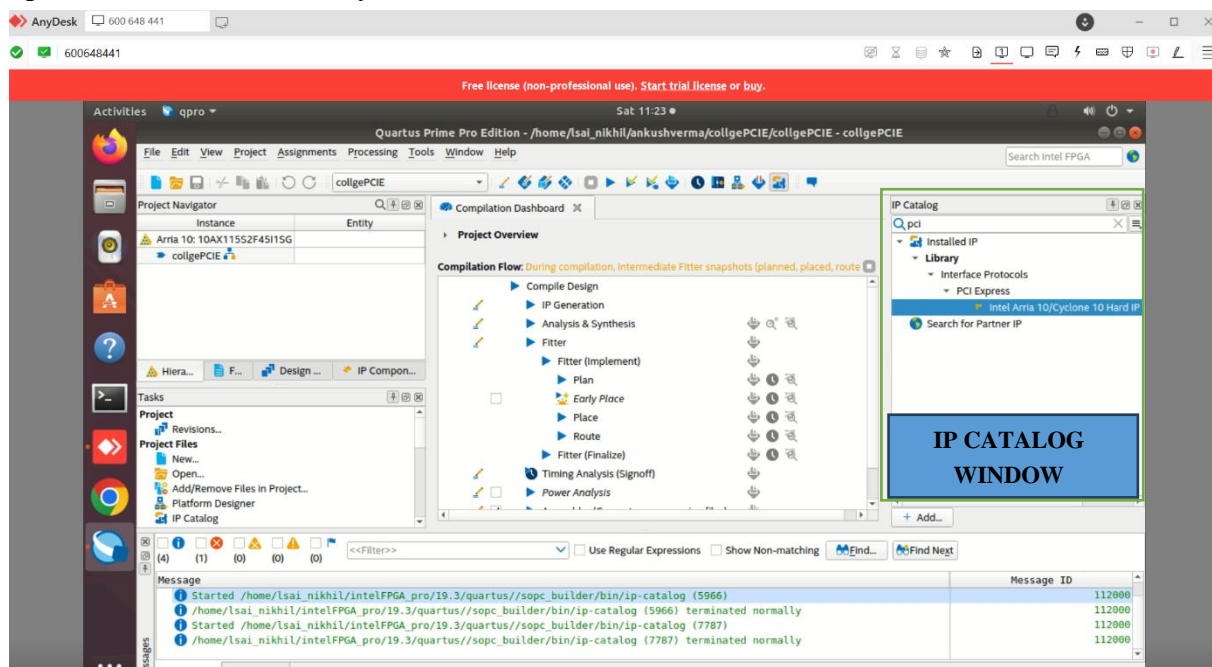


Figure 4.1 PCIe IP Catalog Window

2.2 PCIE IP Parameters Settings:

Design Environment Parameter:

It is used to identify the environment in which the PCIe IP core operates.

- **Standalone Environment:** When set to Standalone, the PCIe IP operates independently, and all its interfaces are exported. In this mode, the IP is in a standalone state, allowing it to function as a separate entity without being integrated into a larger system.
- **System Environment:** When set to System, the PCIe IP is instantiated within a Platform Designer system (formerly Qsys). In this mode, the IP is integrated into a larger FPGA design, allowing it to interact with other IP cores and components within the system.

System Settings Parameter:

The "System Settings" parameter provides configuration options for the PCIe IP core when it is instantiated in a Platform Designer system (System environment).

a. **Application Interface Type:** This parameter allows selecting the interface type to the Application Layer. Options include Avalon-ST, Avalon-MM, Avalon-MM with DMA, and Avalon-ST with SR-IOV. When the Design Environment parameter is set to System, all four interface types are available. However, in Standalone mode, only Avalon-ST and Avalon-ST with SR-IOV are available.

b. **Hard IP Mode:** This parameter allows choosing the lane data rate (Gen1, Gen2, Gen3), the width of the data interface between the hard IP Transaction Layer and the Application Layer implemented in the FPGA fabric, and the Application Layer interface frequency. Different configurations are available based on the FPGA device, and for example, Intel Cyclone 10 GX devices support up to Gen2 x4 configurations.

c. **Port Type:** This parameter specifies the type of port for the PCIe IP core.

- **Native Endpoint:** The IP core acts as an endpoint, and its parameters are stored in the Type 0 Configuration Space. This is suitable where the IP core is a peripheral device.
- **Root Port:** The IP core acts as a root port, and its parameters are stored in the Type 1 Configuration Space. This mode is used when the IP core connects to other PCIe devices as a host controller.

d. **RX Buffer credit allocation - performance for received requests:**

The "RX Buffer Credit Allocation" parameter in the PCIe IP core determines how credits are allocated in the 16 KB receive (RX) buffer for handling incoming data requests and completions. This allocation setting can be adjusted to optimize system performance.

There are three available options for credit allocation:

- Minimum: This configuration allocates credits according to the minimum PCIe specification allowed for non-posted and posted request credits. The majority of the RX buffer space is reserved for received completion header and data. This option is suitable when the application logic generates many read requests but only infrequently receives single requests from the PCIe link.
- Low: With this setting, a slightly larger amount of RX buffer space is allocated for non-posted and posted request credits, but most of the space is still dedicated to received completion header and data. It is ideal for variations where the application logic generates many read requests and infrequently receives small bursts of requests from the PCIe link. This option is recommended for typical endpoint applications where most of the PCIe traffic is generated by a DMA engine.
- Balanced: This configuration evenly distributes the RX buffer space between received requests and received completions. It is a suitable option when the number of received requests and received completions are roughly equal.

e. RX Buffer Completion Credits:

This refers to the count of completion credits in the 16 KB Receive (RX) buffer. These credits indicate the available space in the buffer to receive incoming data.

- Header Credits: Each header credit corresponds to 16 bytes of space in the RX buffer.
- Data Credits: Each data credit corresponds to 20 bytes of space in the RX buffer.

The number of completion credits in the RX buffer is determined by the credit allocation parameter, which helps manage and allocate buffer space efficiently for incoming data.

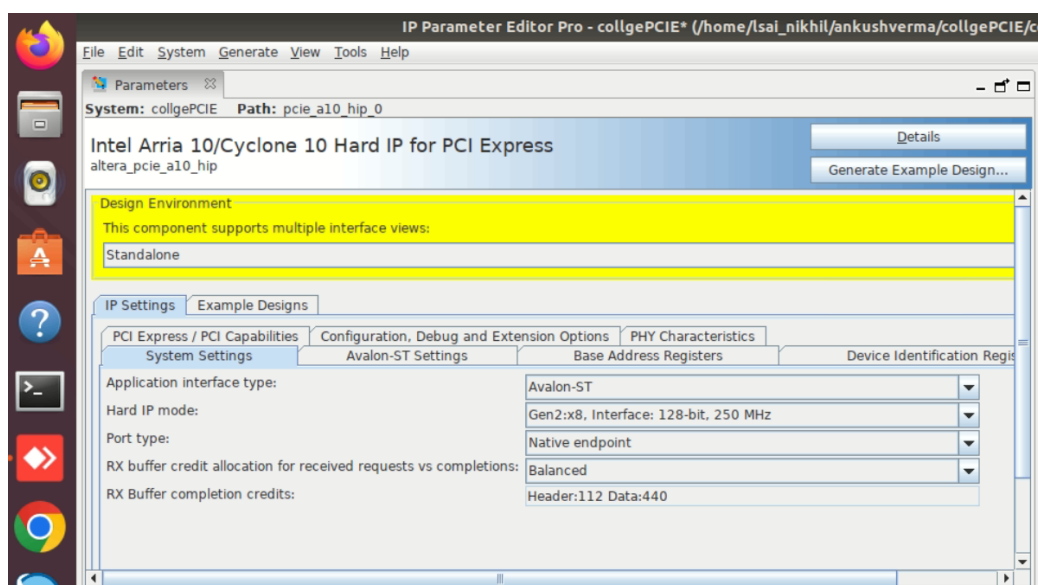


Figure 4.2 IP Parameter Editor

CHAPTER 5: FPGA CVP

1. INTRODUCTION

Quartus Prime, Configuration by Protocol (CVP) is a feature that allows us to configure Intel FPGA devices using a non-standard configuration protocol. It enables us to load FPGA configurations through interfaces other than the standard configuration options like JTAG, AS (Active Serial), or Passive Serial (PS) configuration.

CVP is particularly useful in applications where we need to reconfigure the FPGA in-system using specialized protocols or interfaces that are not natively supported by Quartus. By leveraging CVP, we can customize the configuration flow and use our preferred protocol for FPGA configuration.

- Configuration via Protocol (CvP) is a configuration scheme supported in Arria® V, Cyclone® V, Stratix® V, and Intel® Arria 10 device families.
- CvP creates separate images for the periphery and core logic. The periphery image can be stored in a local configuration device, while the core image can be stored in the host memory.
- This separation reduces system costs and increases security for the proprietary core image.
- CvP configures the FPGA fabric through the PCI Express* (PCIe) link and is available for Endpoint variants only.
- Using CvP, the PCIe endpoint can wake up within 200 ms, enabling faster system response times.

1.1 Benefits of Using CvP:

- By minimizing the size of the local flash device required for storing the periphery configuration data, it effectively lowers system expenses.
- Enhanced security measures are implemented to safeguard the proprietary core bitstream, allowing exclusive access to the FPGA core image for the PCIe host.
- The configuration process is made more straightforward, leading to a simplified software model. A smart host can utilize the PCIe protocol and application topology to efficiently initialize and update the FPGA fabric.
- Enables seamless hardware acceleration capabilities.

2. CVP SYSTEM

In a CvP (Configuration via Protocol) system, the following components are required:

- **FPGA:** This is the main Field-Programmable Gate Array where the configuration takes place.
- **PCIe Host:** The FPGA is connected to a PCIe host, which facilitates the configuration process.
- **Configuration Device:** The configuration device is connected to the FPGA through a conventional configuration interface. The interface can be active serial (AS), passive serial (PS), or fast passive parallel (FPP) based on the chosen configuration scheme. The configuration device stores the periphery image.
- **PCIe Hard IP Block for CvP and PCIe Applications:** The FPGA includes PCIe Hard IP blocks for both CvP and other PCIe applications. The CvP scheme utilizes the bottom left PCIe Hard IP block, which must be configured as an Endpoint.
- **PCIe Hard IP Block only for PCIe Applications:** The other PCIe Hard IP blocks on the FPGA can only be used for standard PCIe applications and cannot be utilized for CvP.

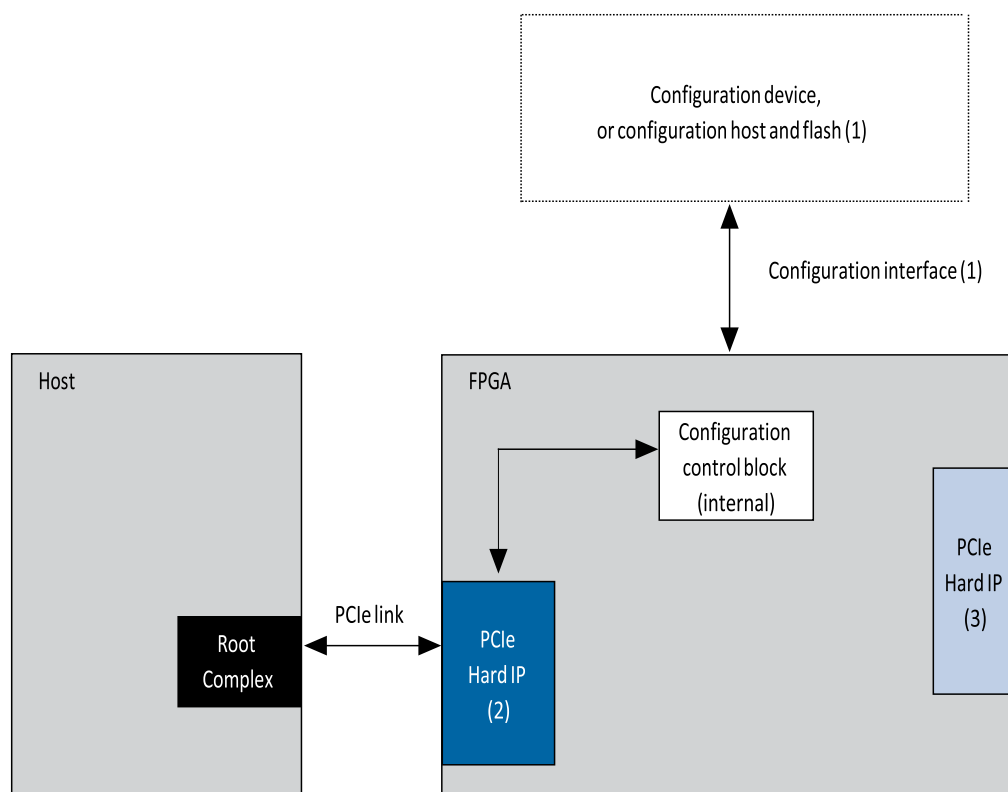


Figure 5.1 CvP Block Diagram

CHAPTER 6: SIGNAL TAP LOGIC ANALYZER

1. INTRODUCTION

The Signal Tap logic analyzer, a feature available within the Intel Quartus Prime software, captures and presents real-time signal behaviour within an Intel FPGA design. We use the Signal Tap logic analyzer to probe and debug the behaviour of internal signals during normal device operation, without requiring extra I/O pins or external lab equipment.

By default, the Signal Tap logic analyzer continuously captures data from specified signals while it is running. However, to focus on specific signal data, we can define trigger conditions that initiate the start or stop of data capture. When these trigger conditions are met, the analysis halts, and the relevant data is displayed. Additionally, we have the option to save the captured data in the device memory for future analysis and apply filters to exclude irrelevant data.

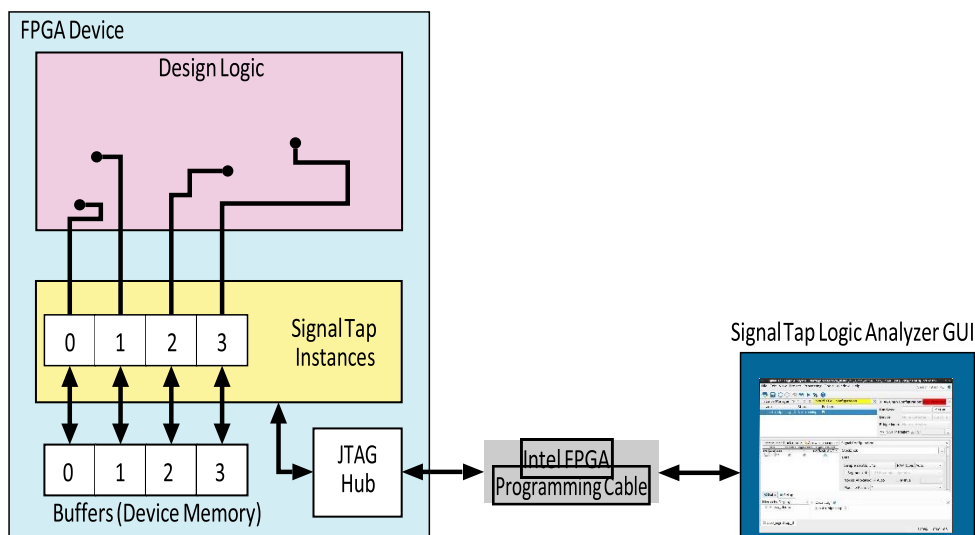


Figure 6.1 Signal Tap Logic Analyzer Block Diagram

2. SIGNAL TAP LOGIC ANALYZER GUI

The Graphical User Interface (GUI) of the Signal Tap logic analyzer offers a convenient platform for swiftly defining and adjusting Signal Tap signal configurations and JTAG connection parameters. It allows users to visualize captured signals during the analysis, initiate and halt the analysis process, as well as view and record signal data. Whenever a Signal Tap instance is configured using the GUI, the instance settings are saved in a reusable Signal Tap Logic Analyzer file (.stp) by Signal Tap.

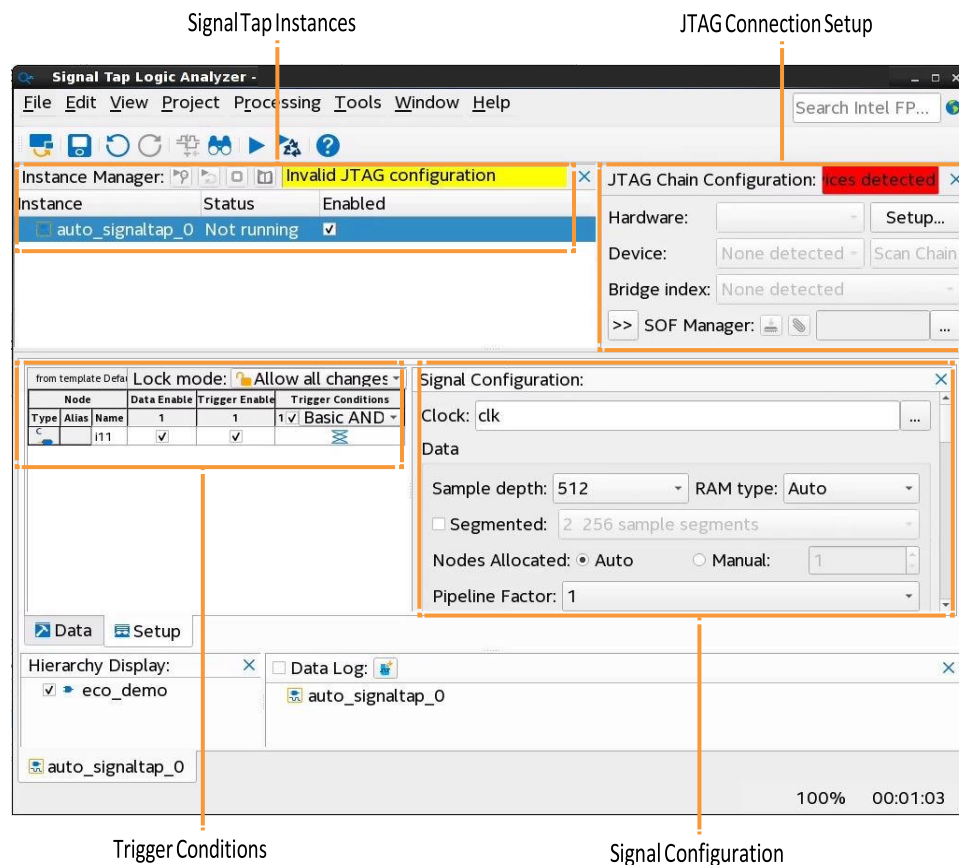


Figure 6.2 Signal Tap Logic Analyzer GUI

2.1 Signal Tap Logic Analyzer and Simulator Integration

Signal Tap logic analyzer can be integrated with a supported simulator environment. It generates a list of "simulator-aware" nodes for tapping into any design hierarchy, providing full visibility into internal signal states in the RTL simulator. Additionally, Signal Tap supports automatic RTL simulation testbench creation, allowing direct export of acquired hardware data into the RTL simulator to observe signals beyond the ones initially specified for tapping. This enables the replication of simulation events using live data traffic in the simulator.

3. SIGNAL TAP LOGIC ANALYZER CAPABILITIES

The Signal Tap logic analyzer offers various powerful capabilities:

- **Multiple Logic Analyzers:** It supports multiple logic analyzers in a single device or across multiple devices in a chain. This enables data capture from multiple clock domains and devices simultaneously.
- **Multiple Trigger Conditions:** Up to 10 trigger conditions can be set for each analyzer instance, allowing complex data capture commands for accurate debugging and problem isolation.
- **Power-Up Trigger:** It captures signal data for triggers occurring after device programming but before manually starting the logic analyzer, helping to catch specific events.
- **Custom Trigger HDL Object:** Users can define custom triggers in Verilog HDL or VHDL and tap specific instances of modules throughout the design hierarchy, simplifying connections.
- **State-Based Triggering Flow:** Organize triggering conditions to precisely define data capture.
- **Flexible Buffer Acquisition Modes:** Provides precise control over data written into the acquisition buffer, allowing discarding of irrelevant data samples.
- **MATLAB Integration:** Capture data can be collected into a MATLAB integer matrix, facilitating further analysis.
- **RTL Simulator Integration:** Easy creation of nodes to tap in the design hierarchy and observe internal signal states directly in the RTL simulator. Automatic testbench creation enables direct injection of Signal Tap data into the RTL simulator.
- **High Channel Count:** Up to 4,096 channels per logic analyzer instance can be sampled, including wide bus structures.
- **Large Sample Depth:** Each instance can capture up to 128K samples per channel, ensuring extensive data capture.
- **Fast Clock Frequencies:** Synchronous sampling of data nodes using the same clock tree driving the logic under test.
- **Compatibility with Other Debugging Utilities:** Can be used alongside JTAG-based on-chip debugging tools, like In-System Memory Content editors, to change signal values in real-time.
- **Floating-Point Display Format:** Supports single-precision (IEEE754 Single) and double-precision (IEEE754 Double) floating-point display formats for numerical data representation.

4. SIGNAL TAP DEBUGGING FLOW

The Signal Tap debugging flow involves the following steps:

- Compile your design, which includes one or more Signal Tap instances that you define.
- Configure the target device accordingly.
- Run the logic analyzer to capture and analyze signal data from the design.

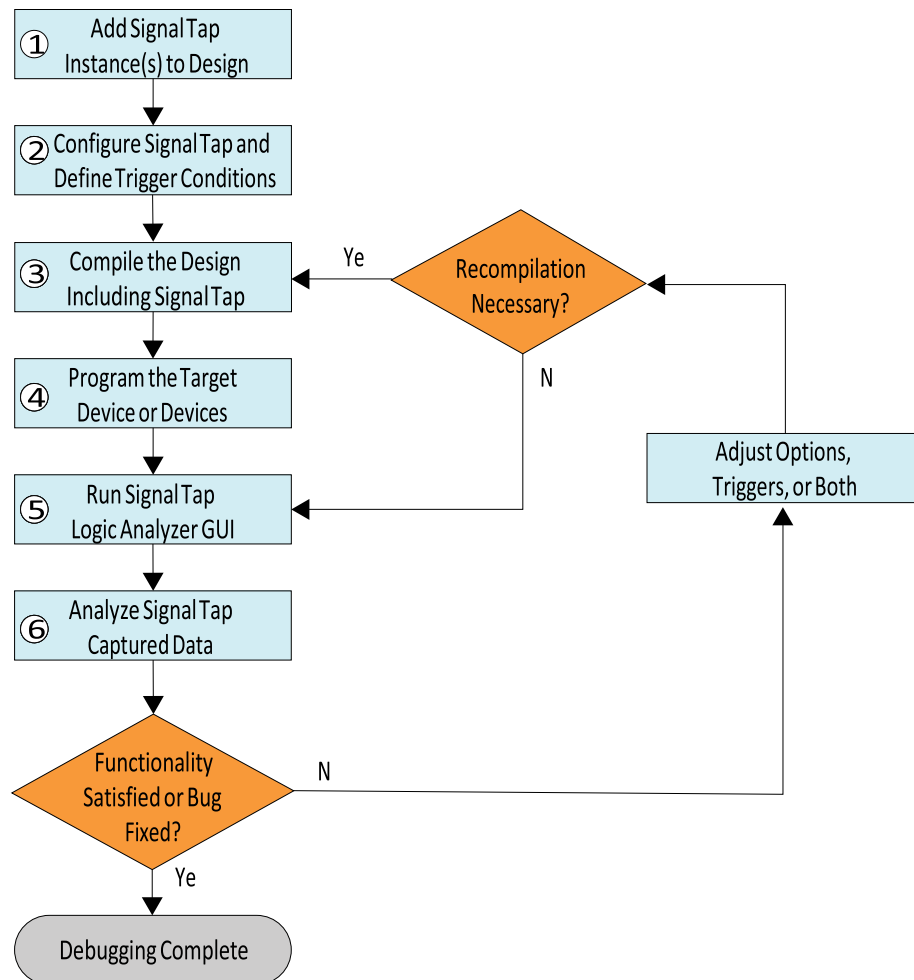


Figure 6.3 Signal Tap Debugging Flow

5. SIGNAL TAP FPGA IP PARAMETERS

The Signal Tap FPGA IP offers several configurable parameters :

Data:

- Data Input Port Width: This parameter can be set from 1 to 4096. The default value 1.
- Sample Depth: It defines the number of samples to be collected, ranging from 0 to 128K. The default value is 128.
- RAM Type: This parameter determines the memory type used for sample collection and storage. Multiple options are available, including Auto (default), M20K/M10K/M9K, MLAB/LUTRAM, and M144K.

Segmented Acquisition:

- Segmented: When enabled, the memory space is divided into separate buffers. Each buffer acts as an individual FIFO with its own set of trigger conditions, behaving as a non-segmented buffer. By default, this option is turned off.
- Number of Segments: This parameter sets the number of segments in each memory space. The default value is 2.
- Samples per Segment: It specifies the number of samples captured by Signal Tap per segment. The default value is 64.

Storage Qualifier:

- Continuous or Input Port: This parameter allows selecting the method for storage - continuous or based on input port.
- Record Data Discontinuities: It determines whether data discontinuities should be recorded or not.

Trigger:

- Trigger Input Port Width: This parameter can be configured from 1 to 4096. The default width is set to 1.
- Trigger Conditions: This parameter sets the number of trigger conditions or levels to be implemented, ranging from 1 to 10. The default value is 1.
- Trigger In: When enabled, this option creates and enables a port for the Trigger In.
- Trigger Out: When enabled, this option creates and enables a port for the Trigger Out.

Pipelining:

The Pipeline factor parameter defines the levels of pipelining added to potentially improve fMAX (maximum frequency). It can be set from 0 to 5, with the default value being 0.

CHAPTER 7: PCIE-BASED DATA TRANSFER BETWEEN HOST PC AND ARRIA 10 FPGA

1. INTRODUCTION

1.1 Xillybus:

Xillybus is a technology and solution designed to facilitate data flow handling for FPGA (Field-Programmable Gate Array) engineers. It offers a streamlined and efficient way to handle data transfer between FPGA-based hardware and other systems like PCs.

The primary purpose of Xillybus is to address the challenges and complexities often faced by FPGA engineers when dealing with data flow. Managing data streams and ensuring smooth communication between FPGA-based devices and external systems can be challenging, leading to bugs and difficulties in the development process.

Xillybus provides a comprehensive solution that goes beyond simply providing a basic wrapper for data transport, such as a PCIe DMA engine. Instead, it offers several end-to-end stream pipes specifically designed for application data transport. This approach aims to solve data flow issues in FPGA projects "once-and-for-all," making it easier for engineers to handle data streams and maintain reliable communication.

The technology has undergone extensive stress testing on various FPGA platforms and configurations to ensure robustness and dependability. By offering a straightforward and efficient solution for data transfer, Xillybus simplifies the integration of FPGA-based devices with other systems and helps FPGA engineers focus more on their core design tasks rather than dealing with data flow intricacies.

Features of Xillybus:

- **End-to-End Stream Pipes:** Xillybus provides multiple end-to-end stream pipes that are specifically designed for application data transport. These dedicated channels ensure efficient and optimized data transfer between the FPGA and external systems.
- **Simplified Data Flow Handling:** Xillybus simplifies data flow handling for FPGA engineers, addressing one of the challenging aspects of FPGA development. It offers a straightforward and user-friendly interface for managing data streams.
- **Robustness and Dependability:** Xillybus has undergone extensive stress testing on various FPGA platforms and IP core configurations. This testing ensures that the solution is robust, reliable, and capable of handling real-world data transfer scenarios.

- **Low-Latency Communication:** By offering optimized stream pipes, Xillybus facilitates low-latency communication between the FPGA and external systems. This is especially beneficial for applications requiring real-time data processing or interactive communication.
- **Flexible Integration:** Xillybus can be easily integrated into FPGA projects, making it a seamless addition to existing designs. The integration process is straightforward and low risk, allowing engineers to evaluate its value quickly.
- **Customizable Data Transfer:** Xillybus allows for customization of data transfer configurations, enabling engineers to tailor the data flow to match the specific requirements of their applications.
- **Versatile Compatibility:** Xillybus is designed to be compatible with various FPGA platforms and IP core configurations, making it a versatile solution that can be utilized across different projects.
- **High-Speed Data Transfer:** Leveraging efficient stream pipes and optimized data handling, Xillybus facilitates high-speed data transfer between the FPGA and external systems, maximizing overall system performance.
- **Real-Time Processing Support:** With its low-latency communication and customizable data transfer, Xillybus is suitable for applications that require real-time data processing and immediate response.
- **Efficient Resource Utilization:** Xillybus is designed to utilize FPGA resources efficiently, minimizing resource overhead and leaving more room for other critical components of the FPGA design.

IP Core Functionality:

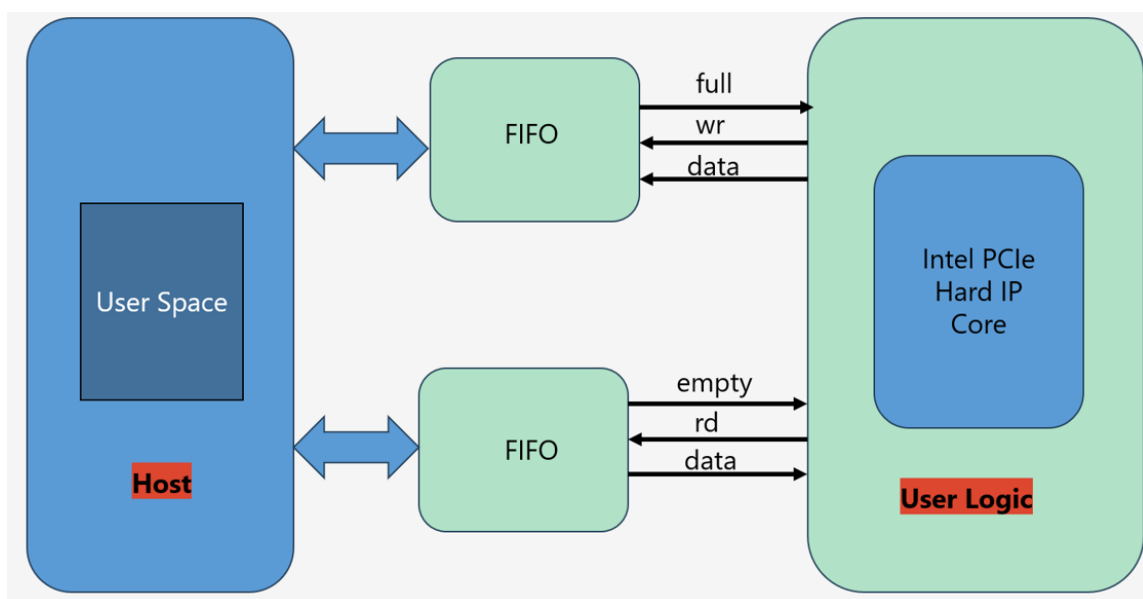


Figure 7.1 Generic Data and Control Flow from PCIe-to-Host

The IP core, as depicted in the simplified block diagram above, facilitates data communication between the user logic and the Xillybus IP core through a standard FIFO, known as the "Application FIFO." The FPGA designer has the flexibility to determine the depth of this FIFO and its interface with the application logic, granting them full control over the data flow.

This design approach liberates the FPGA designer from managing data traffic with the host. Instead, the Xillybus core efficiently checks the FIFOs' "empty" and "full" signals in a round-robin manner, initiating data transfers as needed until the FIFOs become empty or full again, depending on the data stream's direction.

On the other side, the Xillybus IP core seamlessly interfaces with the PCIe core supplied by Xilinx or Intel (formerly Altera), as illustrated in the diagram.

To maintain simplicity, the Xillybus IP core operates without knowledge of the expected data rate or when the user logic will supply or fetch data. It only monitors the FIFO's state, responding appropriately when it becomes empty or full. While this approach may lead to some inefficiency with low data rates, it becomes insignificant as the data rate increases. Overall, Xillybus effectively utilizes bus transport resources when necessary and allows for flexibility when they are not fully utilized.

Despite the desire of FPGA engineers to control DMA transactions and use resources optimally, this design approach demonstrates that there is little or no advantage in doing so. Instead, a plain FIFO drain, or source facilitates a more adaptable design, free from stiffness and capable of accommodating future modifications without compromising performance.

The host driver provides essential functionalities:

- **Device File Generation:** The host driver generates device files that emulate named pipes, acting like regular files but functioning as communication channels between processes or TCP/IP streams. These files allow reading and writing, and on the host side, they connect to a FIFO in the FPGA. This design facilitates high-rate data transfers and accommodates occasional single-byte data exchanges, like TCP/IP streams.
- **Configuration Versatility:** A single driver binary is capable of supporting any Xillybus IP core configuration. Upon loading into the host's operating system, the driver automatically detects the streams and their attributes, creating corresponding device files. For Windows systems, these files are accessed as `\\.\xillybus_something`, while on Linux, they appear as `/dev/xillybus something`.
- **DMA Buffer Allocation:** During driver load, DMA buffers are allocated within the host's memory space, and their addresses are communicated to the FPGA. Each stream has a separate set of parameters, including the number and size of DMA buffers. These

parameters are hardcoded in the FPGA IP core for a specific configuration and are retrieved by the host during the discovery process.

- **Illusion of Continuous Data Stream:** A handshake protocol between the FPGA and host creates the impression of a continuous data stream. Behind the scenes, DMA buffers are filled with data, passed to the other side, and acknowledged. Similar to techniques used in TCP/IP streaming, this approach optimizes DMA buffer utilization and maintains responsiveness, particularly for handling small data pieces.

Schematic of Xillybus:

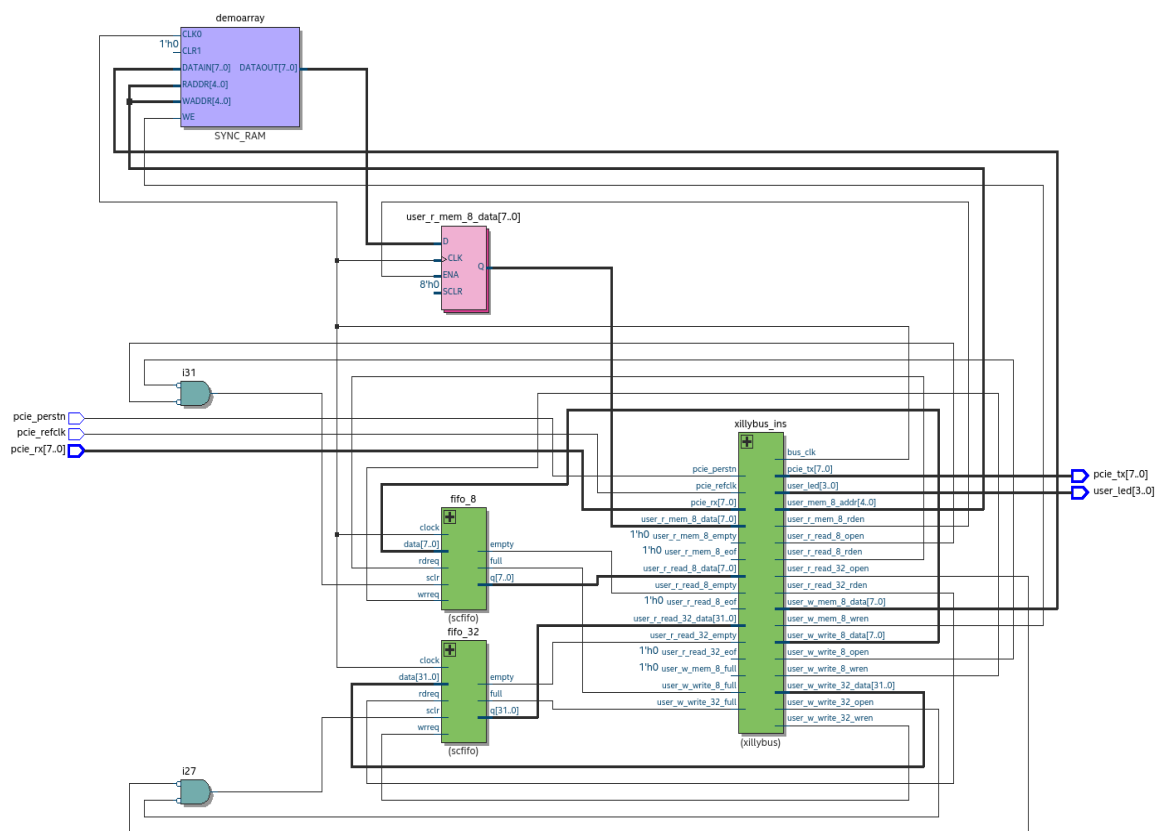


Figure 7.2 Schematic of Xillybus

Conclusion:

In conclusion, Xillybus provides an abstraction interface that caters to the preferences of both FPGA engineers and host application programmers, allowing them to work comfortably. Although there might be some instances of PCIe bandwidth wastage, the system's efficiency improves significantly as the demand for bandwidth increases, rendering these occurrences of wasted bandwidth nearly imperceptible.

1.2 *LighSiP*:

LightSiP is an advanced hexagonal optically interconnected network processor that brings significant improvements to data interface, performance, and efficiency. Compared to traditional processors, this innovative solution enhances data interface speeds by an impressive 100 times, while boosting overall performance by 40 times per watt. Furthermore, it manages to reduce volume by five times and decrease costs by a notable two times, making it a highly cost-effective choice for various applications.

One of the standout features of LightSiP is its adaptability and scalability. Multiple LightSiPs can be effortlessly configured in a pre-defined orientation, enabling the creation of a powerful array of processors for distributed computing applications. This means that tasks can be efficiently distributed among the processors, enhancing overall computational capabilities and enabling parallel processing of data-intensive tasks.

The versatility of LightSiP extends even further, as these System in Package (SiP) units can be arranged in either 1U or 2U racks. This allows for flexible deployment options, making it easier to integrate LightSiP into existing systems or design new setups to suit specific application requirements.

In summary, LightSiP is an innovative and high-performance optically interconnected network processor. Its exceptional data interface improvements, remarkable performance-to-watt ratio, reduced form factor, and cost-effectiveness make it a compelling choice for a wide range of computing applications. Whether utilized individually or combined in an array, LightSiP promises to elevate computational capabilities and enhance overall system efficiency, making it a valuable solution for modern computing needs.

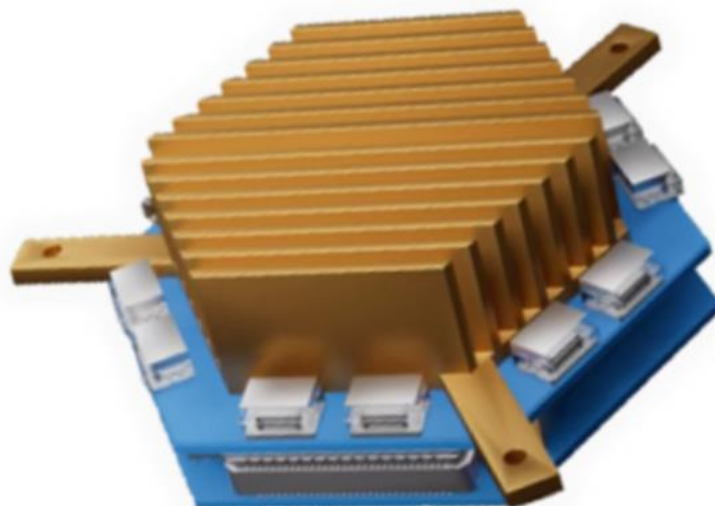


Figure 7.3 LightSiP Prototype

The features of this LightSiP include:

- Two FPGAs with 4 high-speed lanes for intercommunication, providing a combined processing power of 3000 GFLOPS and a throughput of 1200 Gbps across 72 available channels, all while operating at 100W power consumption.
- Equipped with a PCIe external user interface for seamless integration with other devices.
- Compact 55mm Hexagon Package for space-efficient design.
- Flexibility for either pluggable or solderable packaging options based on specific needs and preferences.

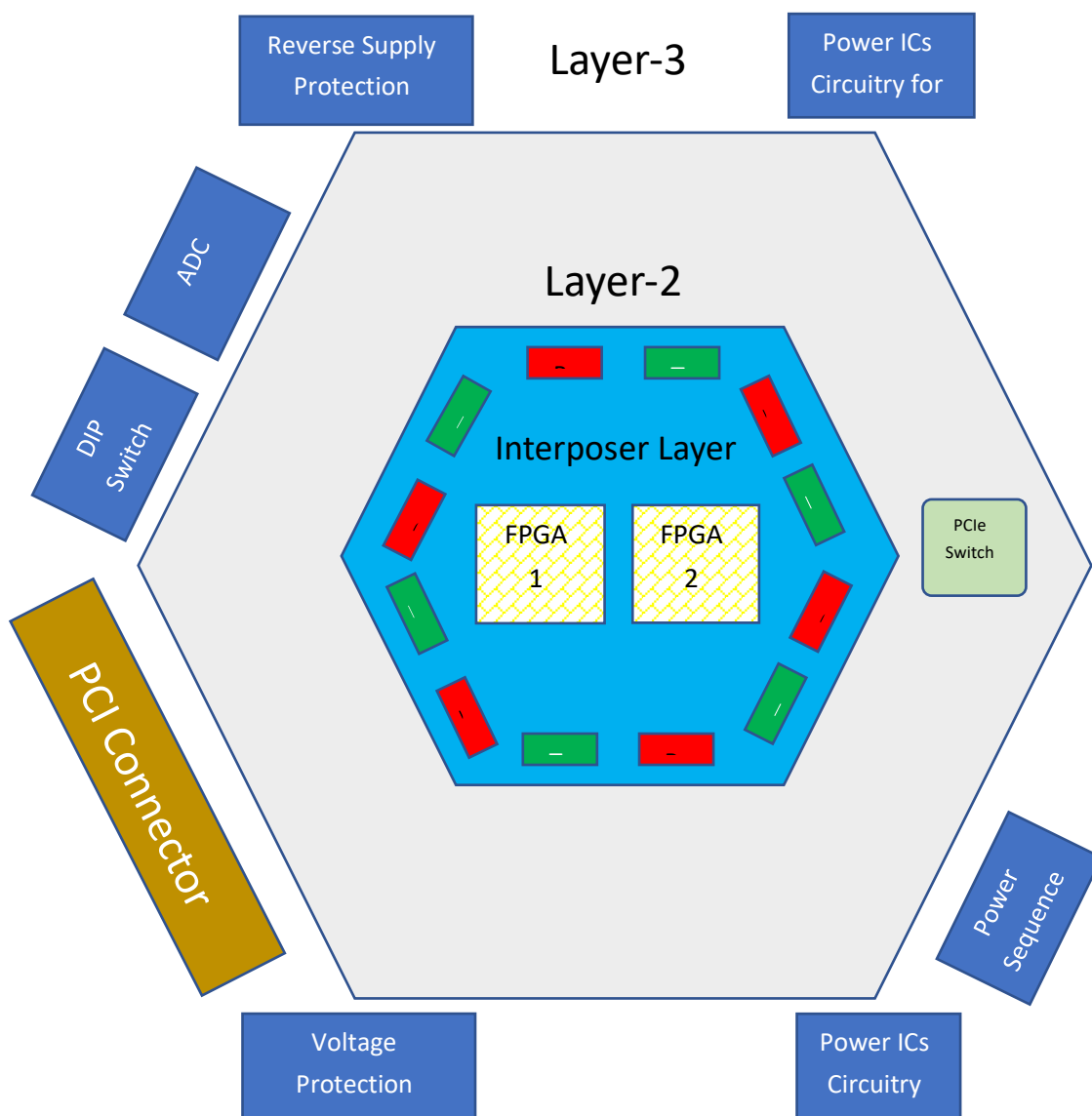


Figure 7.4 LightSiP Internal Preliminary Architecture

2. OBJECTIVES

The objective of the project to enable data transfer between the host PC and Arria 10 FPGA using PCIe can be summarized in three major points:

1. **Establishing High-Speed Communication:** The primary objective is to establish a reliable and high-speed communication channel between the host PC and the Arria 10 FPGA using the PCIe (Peripheral Component Interconnect Express) interface. PCIe is a widely used interface for connecting peripheral devices to a computer's motherboard, offering high bandwidth and low latency. By leveraging PCIe, the project aims to achieve efficient and fast data transfer between the PC and FPGA.
2. **Enabling Bi-Directional Data Transfer:** Another key objective is to enable bi-directional data transfer between the host PC and the Arria 10 FPGA. This means that not only will the PC be able to send data to the FPGA for processing or storage, but the FPGA will also be able to send data back to the PC. Enabling this bi-directional data flow is crucial for applications that require real-time processing or interactive data exchange between the PC and FPGA.
3. **Ensuring Compatibility and Scalability:** The project also focuses on ensuring compatibility and scalability between the host PC and the Arria 10 FPGA. Compatibility involves ensuring that the PCIe interface of the FPGA is compatible with the PC's PCIe slots and the associated software drivers. Scalability, on the other hand, involves designing the system to support different data transfer rates, accommodating future upgrades or expansions. By addressing compatibility and scalability, the project aims to create a flexible and adaptable solution for data transfer between the PC and FPGA.

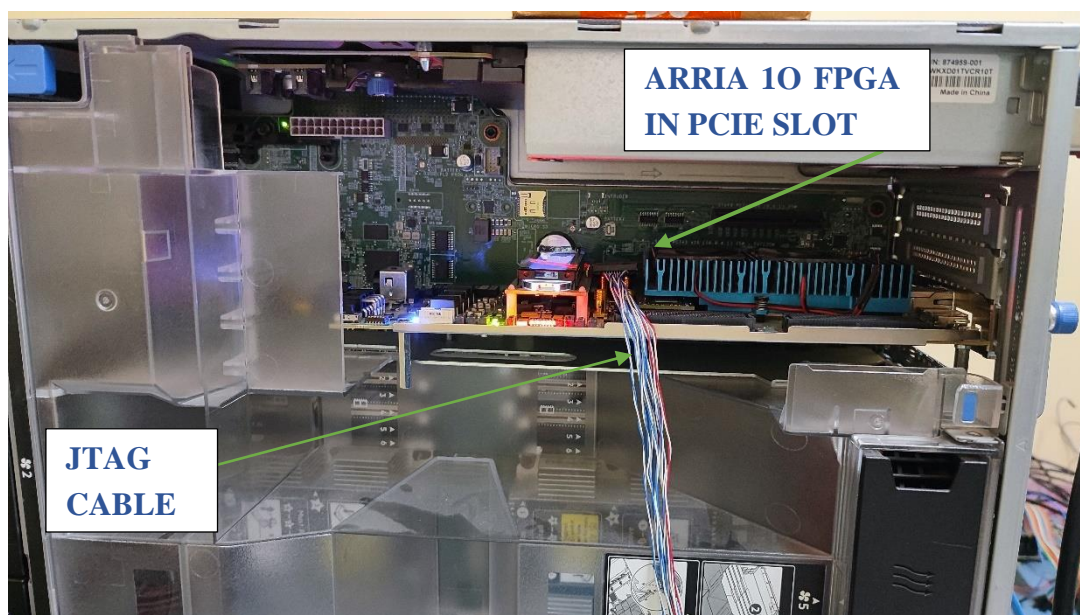


Figure 7.5 Arria 10 PCIe Accelerator Integration in Host PC

3. SCOPE OF THE PROJECT:

The scope of the project is to develop a PCIe-based data transfer system between a host PC and an Arria 10 FPGA. This system will be later integrated with the company's product named "LightSIP." The overarching concept of "LightSIP" involves splitting a single processor into smaller units known as "LightSIPs." These individual "LightSIPs" communicate with each other using high-speed light signals instead of traditional wires. Each "LightSIP" unit comprises two FPGAs that facilitate communication with the host PC using the PCIe interface.

The primary objectives of the project are as follows:

- **PCIe-Based Data Transfer:** The core focus of the project is to establish a high-speed and reliable data transfer mechanism between the host PC and the Arria 10 FPGA. Leveraging the capabilities of PCIe, the project aims to enable efficient and fast data exchange.
- **Integration with "LightSIP" Product:** The developed PCIe data transfer system will be seamlessly integrated into the company's product "LightSIP." This integration will enhance the product's overall capabilities, enabling efficient communication between the individual "LightSIP" units and the host PC.
- **Bi-Directional Data Transfer:** The system will be designed to support bi-directional data transfer. It will allow data to flow from the host PC to the Arria 10 FPGA for processing or storage, and also enable the FPGA to transmit data back to the PC. This bi-directional communication is crucial for real-time processing and interactive data exchange.
- **Compatibility and Scalability:***The project will ensure that the PCIe interface of the Arria 10 FPGA is compatible with the PCIe slots on the host PC and the associated software drivers. Additionally, the system will be designed with scalability in mind to support different data transfer rates, accommodating future upgrades and expansions.

The successful implementation of this PCIe-based data transfer system will contribute to the realization of the "LightSIP" product's vision, allowing seamless communication between multiple "LightSIP" units via high-speed light signals. This innovative technology has the potential to revolutionize the way processors communicate with each other, opening new possibilities for advanced computing and data processing applications.

4. DESIGN DESCRIPTION:

The project aims to implement PCIe-based data transfer between a host PC and an Arria 10 FPGA using Xillybus on a Linux operating system. Xillybus will serve as the interface for efficient and high-speed data communication between the two devices. The FPGA will be responsible for handling data processing tasks, while the host PC will act as the controller and data source/sink. The design of the Project includes the following steps:

4.1 Downloading the Demo Bundle and Software:

To access Xillybus' fully functional software and demo bundle, individuals can download them directly from their website without requiring an activation code or any additional requests. However, it's important to note that the legal permission to use the IP core is restricted to evaluation purposes only. If one intends to use it beyond evaluation, a proper license purchase is necessary to comply with legal regulations. It is essential to understand and adhere to these conditions.

To proceed with the download, users should visit the relevant webpage based on their connection type, either Xillybus or XillyUSB (Xillybus in our case). From there, they can download the following items:

- The demo bundle specifically designed for their development board.
- The appropriate driver for their operating system, whether it's Linux or Windows. Even if using Linux, the driver needs to be obtained, as it might not be included in the Linux distribution (Linux in our case).
- If users are Windows users, they should download the Xillybus package for Windows.
- For Windows users utilizing XillyUSB, they should acquire the diagnostic utility for Windows.

All the files mentioned above are in either .zip or .tar.gz format. Users should extract their content onto their disk by uncompressing them.

If a user's development board is not listed in the available demo bundles, they might need to make modifications within the bundle. In such cases, they should select the demo bundle that corresponds to the same FPGA family as their board and follow the guidelines provided in the documentation for Xilinx or Intel FPGA

Important links for downloading demo bundle and software:

- Xillybus IP core Demo bundle for Arria 10 FPGA board:
<http://xillybus.com/downloads/xillybus-eval-arria10-2.1a.zip>
- Linux driver, udev file and a sample C application:
<http://xillybus.com/downloads/xillybus.tar.gz>

4.2 Creating the Bitstream for Intel Arria 10 FPGA:

It requires the user to have Quartus installed on their computer. If Quartus is already installed, they can proceed to use it. However, if the version is unsuitable, the procedure detailed below will fail, showing an error message indicating the need for a newer version. If there are no such errors, the user's version of Quartus is appropriate.

The steps to create the bitstream file are as follows:

- The user needs to locate and double-click the "xillydemo.qpf" file, which is located in either the "verilog/" or "vhdl/" subdirectory. If they are unsure, they should choose "verilog/".
- Upon double-clicking, Quartus will open the FPGA project in response to the user's action.
- They should click on "Compile Design" to initiate the implementation of the project.
- After the implementation process is complete, Quartus will generate a bitstream file named "xillydemo.sof," which is ready for use.
- The implementation process might produce several warnings in Quartus, but it should not encounter any Critical Warnings or errors.

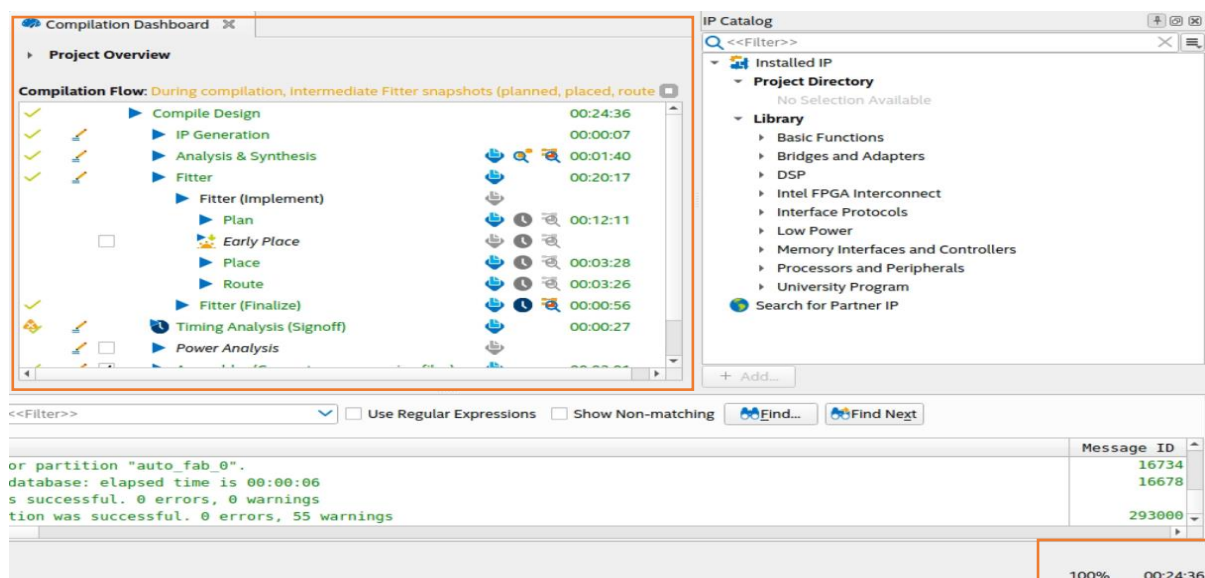


Figure 7.6 Successful Compilation of xillydemo.qpf File

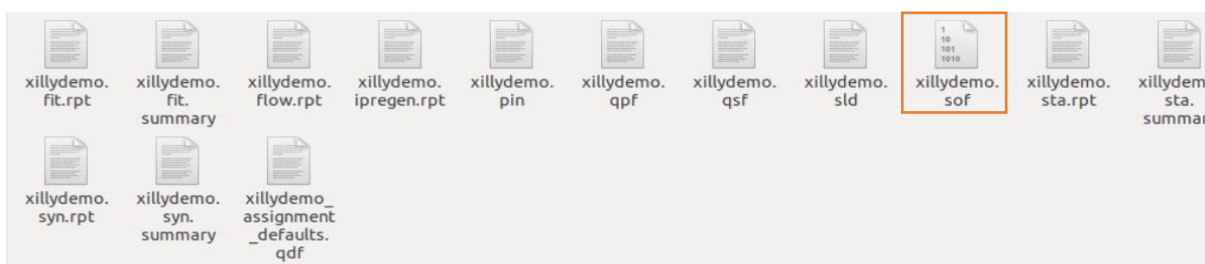


Figure 7.7 Generation of xillydemo.sof File

4.3 Checking for Prerequisites:

In some Linux installations, essential tools for kernel module compilation may be absent. A simple method to check for these tools is by attempting to run them. To compile the kernel module for Xillybus, ensure that three prerequisites are already installed on the computer:

- The Gcc compiler:
To check if "gcc" is installed, the user should run "gcc" in the terminal. If it's installed, they will see a message about "no input files." If not, they'll get "b gcc: command not found."
- The "make" utility:
To check if "make" is installed, users can run "make" in the terminal. If it's installed, they will see a message about "No targets specified." If not, they'll get "make: command not found."
- The kernel headers:
Verifying its installation is slightly more complex. Usually, if the kernel headers are missing, the kernel compilation will fail with an error indicating the absence of a header file.

For Red Hat derivatives like Fedora, RHEL, and CentOS, we can prepare the system by running the following command:

```
--- # yum install gcc make kernel-devel-$(uname -r) ---
```

For Ubuntu and other Debian-based distributions, use the following command:

```
--- # apt install gcc make linux-headers-$(uname -r) ---
```

These commands will ensure that all necessary prerequisites, including the kernel headers, are in place for successful kernel module compilation.

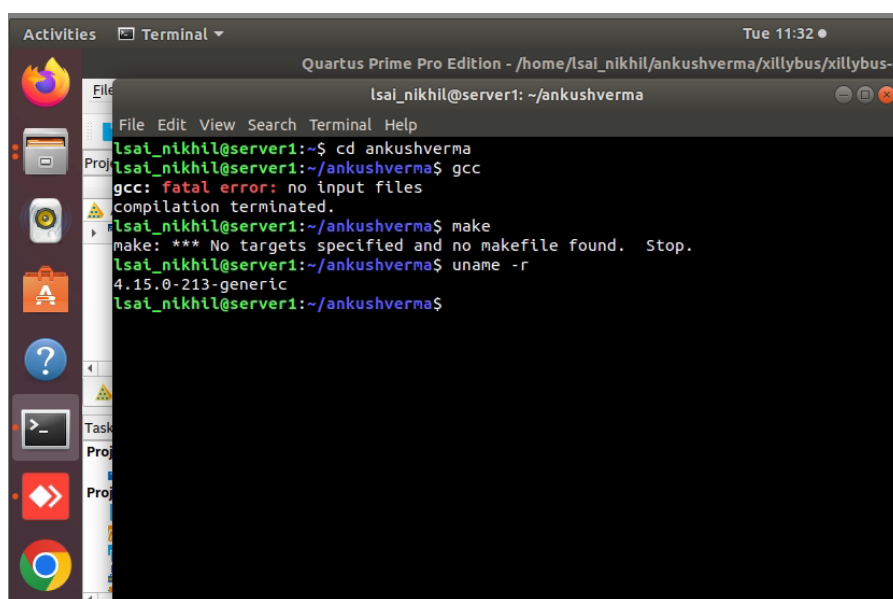


Figure 7.8 Successful Check for Prerequisites

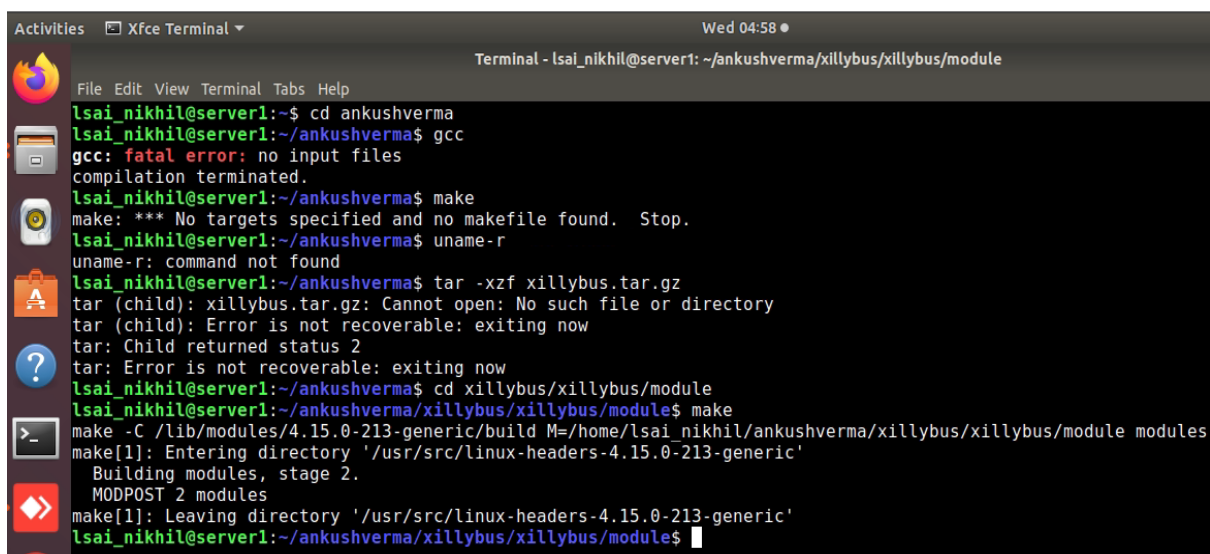
4.4 Compilation of the Kernel Module:

Kernel module compilation is the process of converting device driver source code into a loadable kernel module (LKM). It involves using the Linux build system and a Makefile to generate a .ko file. Once compiled, the module can be loaded into the kernel at runtime to extend its functionality, supporting new devices or features without the need to recompile the entire kernel.

The compilation process involves the following steps:

- Extracting the contents of "xillybus.tar.gz" archive using the below Linux command:
\$ tar -xzf xillybus.tar.gz
- Navigate to the "xillybus/module/" directory using the following Linux command:
\$ cd xillybus/module/
- To compile the driver, use the following Linux command:
\$ make

During compilation, the system will use the specified kernel headers and generate two modules: "xillybus_core.ko" and "xillybus_pcie.ko". The successful compilation will result in the creation of these kernel modules.



```
Activities  Xfce Terminal  Wed 04:58
Terminal - lsai_nikhil@server1: ~/ankushverma/xillybus/xillybus/module
lsai_nikhil@server1:~$ cd ankushverma
lsai_nikhil@server1:~/ankushverma$ gcc
gcc: fatal error: no input files
compilation terminated.
lsai_nikhil@server1:~/ankushverma$ make
make: *** No targets specified and no makefile found. Stop.
lsai_nikhil@server1:~/ankushverma$ uname-r
uname-r: command not found
lsai_nikhil@server1:~/ankushverma$ tar -xzf xillybus.tar.gz
tar (child): xillybus.tar.gz: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
lsai_nikhil@server1:~/ankushverma$ cd xillybus/xillybus/module
lsai_nikhil@server1:~/ankushverma/xillybus/xillybus/module$ make
make -C /lib/modules/4.15.0-213-generic/build M=/home/lsai_nikhil/ankushverma/xillybus/xillybus/module modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-213-generic'
Building modules, stage 2.
MODPOST 2 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-213-generic'
lsai_nikhil@server1:~/ankushverma/xillybus/xillybus/module$
```

Figure 7.9 Successful Compilation of Kernel Module

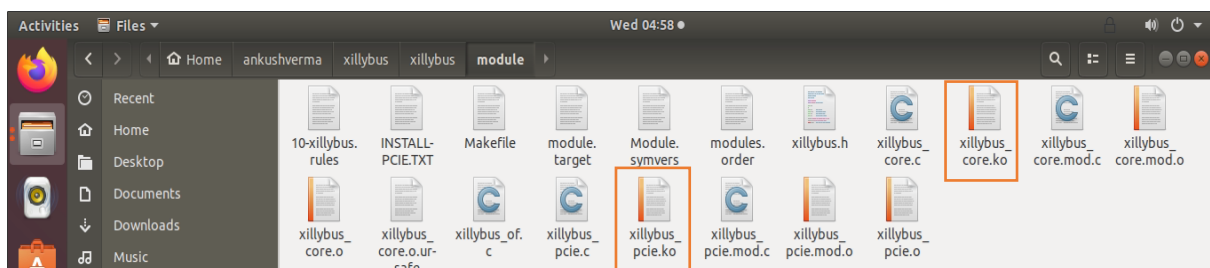


Figure 7.10 Generation of xillybus_core.ko and xillybus_pcie.ko module

4.5 Installing the Kernel Module:

The process of installing the kernel module involves the user staying in the same directory, becoming the root user, and executing the command "make install". This action can take a few seconds and is not expected to generate any errors. In case of a failure, the user is advised to manually copy the generated *.ko files from the compilation process to an existing kernel driver subdirectory. Afterward, the user should run the "depmod" command to update module dependency information.

For instance, the user can perform the following steps to copy the files for the PCIe driver, assuming the relevant kernel version is 3.10.0:

```
--- # cp xillybus_core.ko /lib/modules/3.10.0/kernel/drivers/char/ ---  
--- # cp xillybus_pcie.ko /lib/modules/3.10.0/kernel/drivers/char/ ---  
--- # depmod -a ---
```

It's important to note that the installation process does not immediately load the module into the kernel. The module will be loaded automatically during the next system boot if a Xillybus peripheral is discovered.

Instructions for manually loading and unloading the Xillybus modules are as follows:

To load the module and start working with Xillybus, the user should execute the following command as the root user:

```
--- # modprobe xillybus_pcie ---
```

Executing the appropriate "modprobe" command will result in the appearance of Xillybus device files, provided that the corresponding hardware is present on the bus.

It is important to note that manual loading of the module is not necessary if a Xillybus PCIe/AXI peripheral was already present during the system's boot process and the driver was installed as described earlier.

To view a list of modules in the kernel, the user can use the command "lsmod". If there is a need to remove the module from the kernel (in the PCIe case), the user can use the following command:

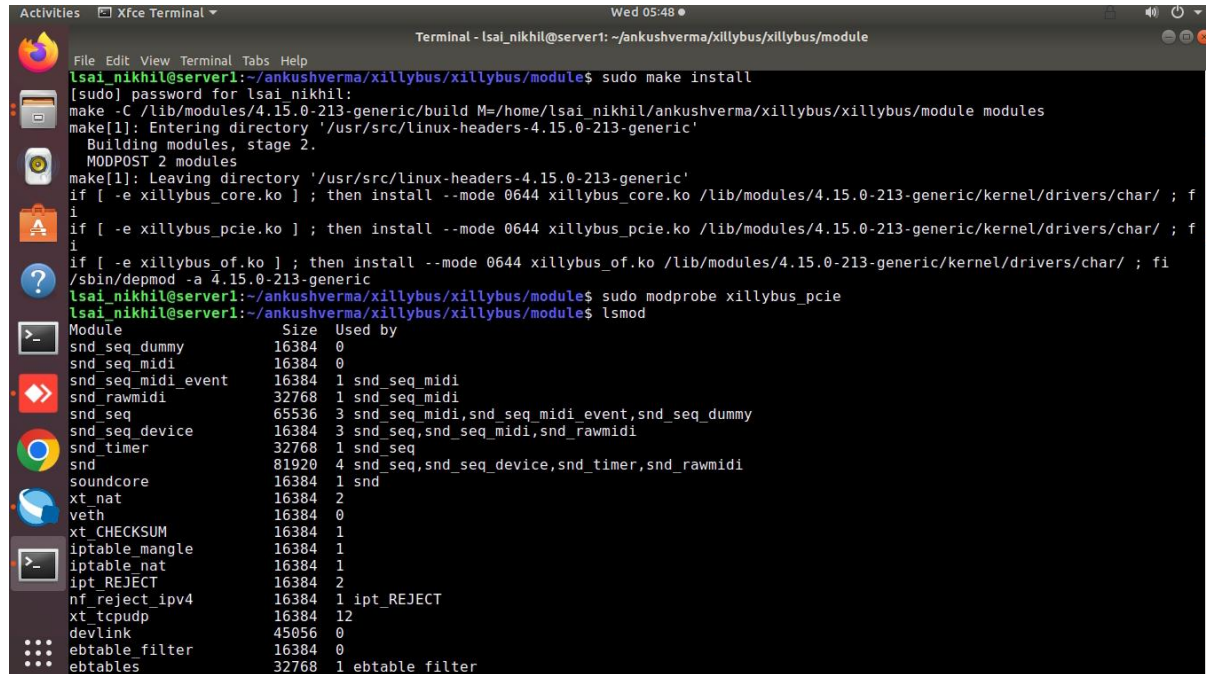
```
--- # rmmod xillybus_pcie xillybus_core ---
```

This will make the device files associated with Xillybus vanish.

If any issues occur during the process, it is recommended to check the "/var/log/syslog" log file for messages containing "xillybus" or "xillyusb" keywords, depending on the context.

Valuable clues can often be found in this log file. Alternatively, the "dmesg" command can be used to access the same log information.

In situations where no "/var/log/syslog" log file is present, it is likely that the information is available in "/var/log/messages" instead.



```
lsai_nikhil@server1:~/ankushverma/xillybus/xillybus/module$ sudo make install
[sudo] password for lsai_nikhil:
make -C /lib/modules/4.15.0-213-generic/build M=/home/lsai_nikhil/ankushverma/xillybus/xillybus/module modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-213-generic'
Building modules, stage 2.
MODPOST 2 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-213-generic'
if [ -e xillybus_core.ko ]; then install --mode 0644 xillybus_core.ko /lib/modules/4.15.0-213-generic/kernel/drivers/char/ ; fi
if [ -e xillybus_pcie.ko ]; then install --mode 0644 xillybus_pcie.ko /lib/modules/4.15.0-213-generic/kernel/drivers/char/ ; fi
if [ -e xillybus_of.ko ]; then install --mode 0644 xillybus_of.ko /lib/modules/4.15.0-213-generic/kernel/drivers/char/ ; fi
lsai_nikhil@server1:~/ankushverma/xillybus/xillybus/module$ sudo modprobe xillybus_pcie
lsai_nikhil@server1:~/ankushverma/xillybus/xillybus/module$ lsmod
Module                  Size  Used by
snd_seq_dummy           16384  0
snd_seq_midi            16384  0
snd_seq_midi_event      16384  1 snd_seq_midi
snd_rawmidi             32768  1 snd_seq_midi
snd_seq                 65536  3 snd_seq_midi,snd_seq_midi_event,snd_seq_dummy
snd_seq_device          16384  3 snd_seq,snd_seq_midi,snd_rawmidi
snd_timer               32768  1 snd_seq
snd                     81920  4 snd_seq,snd_seq_device,snd_timer,snd_rawmidi
soundcore               16384  1 snd
xt_nat                  16384  2
veth                    16384  0
xt_CHECKSUM             16384  1
iptable_mangle          16384  1
iptable_nat             16384  1
ipt_REJECT              16384  2
nf_reject_ipv4          16384  1 ipt_REJECT
xt_tcpudp               16384  12
devlink                 45056  0
ebtable_filter          16384  0
ebtables                32768  1 ebtable_filter
```

Figure 7.11 Successful Installation of the Kernel Module

4.6 COPYING THE UDEV RULE FILE:

Installing Xillybus' udev file is advisable, even if the driver is already installed. By doing so, users can access Xillybus without requiring root privileges. This udev file is included in the package that contains the driver, which is xillybus.tar.gz.

To modify the permissions of Xillybus device files and allow access to all users, the user can utilize the udev mechanism and follow these steps:

- As the root user, navigate to the same directory where the udev rule file is located.
- Copy the udev rule file to the directory where all rules are typically stored, which is often /etc/udev/rules.d/, using the following command:

```
--- # cp 10-xillybus.rules /etc/udev/rules.d/ ---
```

The content of the udev rule file, 10-xillybus.rules, should be as follows:

```
--- SUBSYSTEM=="xillybus", MODE="666", OPTIONS="last_rule" ---
```

This rule indicates that any file generated by the Xillybus device driver should be granted permission mode 0666, allowing read and write access to all users.

4.7 The “HELLO, WORLD” test:

Xillybus functions as a development kit, and its true potential is best uncovered through practical application in real designs. To facilitate this, the initial FPGA code is designed with a basic implementation as a foundation for further work. The core is connected to two pairs of interfaces using two FIFOs, which store and loop back any data sent to the core to the host. Additionally, a RAM is connected to the core, demonstrating memory or register access.

Requirements:

- The FPGA must be loaded with the bitstream obtained from the appropriate demo bundle.
- For platforms other than Xilinx: Ensure that the computer has booted correctly and the PCIe interface has been detected (verify using commands like "lspci").
- Familiarity with the Linux/UNIX command-line interface is necessary.

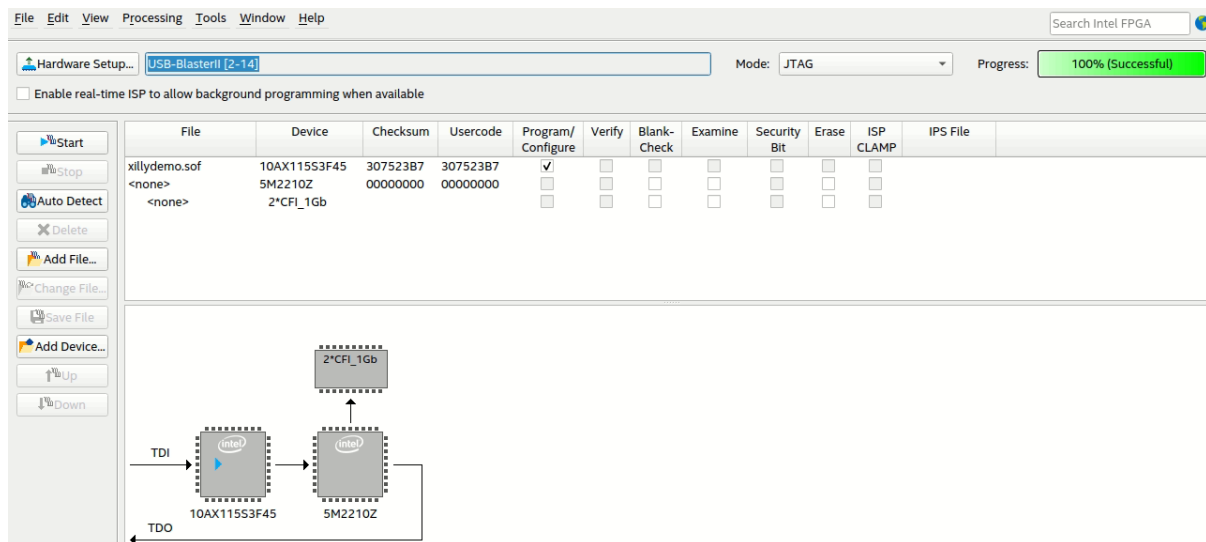


Figure 7.12 Successful Loading Of Bitstream From xillybus.sof File On Arria 10 FPGA

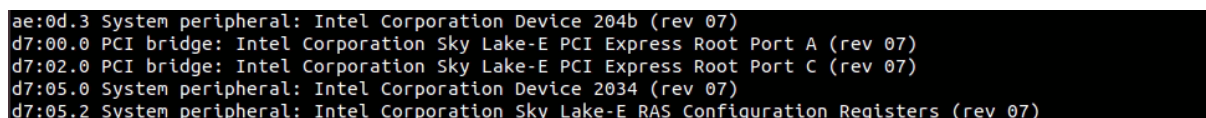


Figure 7.13 PCIe Interface Detected

The trivial loopback test:

The trivial loopback test allows users to verify the loopback functionality of Xillybus. This can be achieved by utilizing the "cat" command, a standard UNIX command-line utility.

Guidelines for Performing Text Transmission Test:

- Open two terminal windows, assigning one for each part of the test.
- In the first terminal, enter the command:
--- \$ cat /dev/xillybus_read_8 ---
- In the second terminal, type the command:
--- \$ cat > /dev/xillybus_write_8 ---
- Type any desired text in the second terminal and press ENTER. The text will then appear in the first terminal, indicating a successful loopback.

HOST RECEIVING

HOST TRANSMITTING

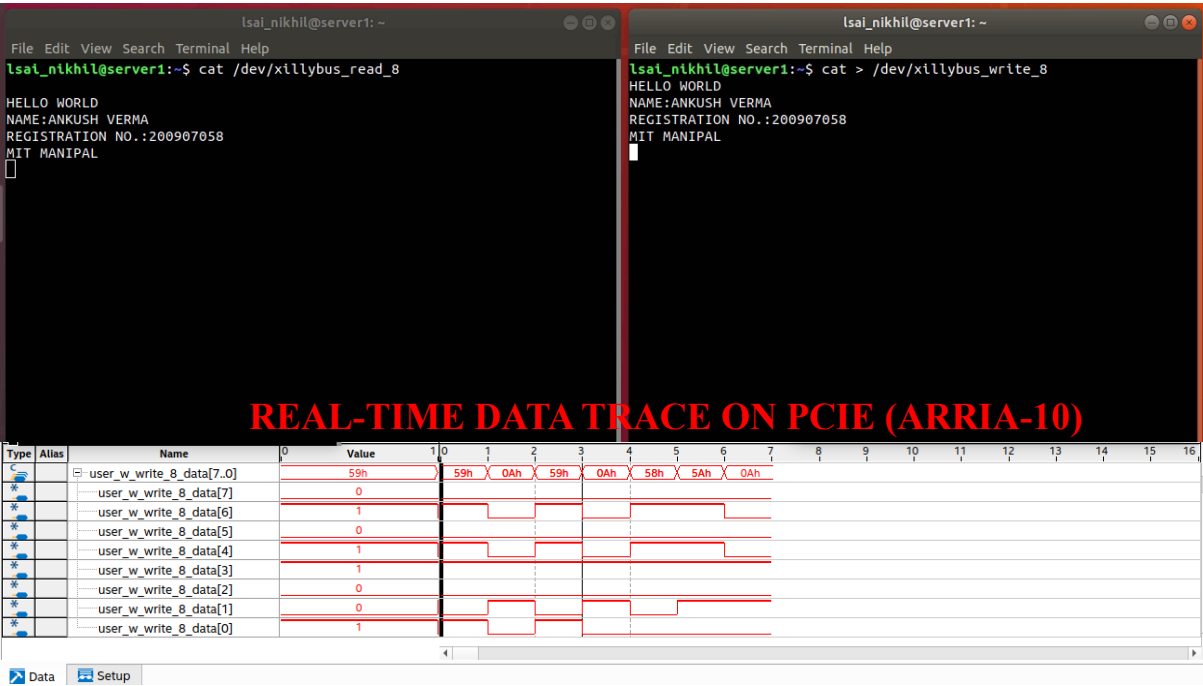


Figure 7.14 Transmitting And Tracing The Text Data Using “Signal Tap Logic Analyzer”

Guidelines for Performing File Transmission Test:

- Open two terminal windows, assigning one for each part of the test.
- In the first terminal, enter the command:
--- \$ sudo cat /dev/xillybus_read_8 ---
And then enter the required Password.
- In the second terminal, type the command:
--- \$ sudo cat ankushverma.txt > /dev/xillybus_write_8 ---
And then enter the required Password.
- Enter any file name followed by ".txt" in the second terminal and press ENTER. The file content will then appear in the first terminal, indicating a successful loopback.

HOST RECEIVING

HOST TRANSMITTING

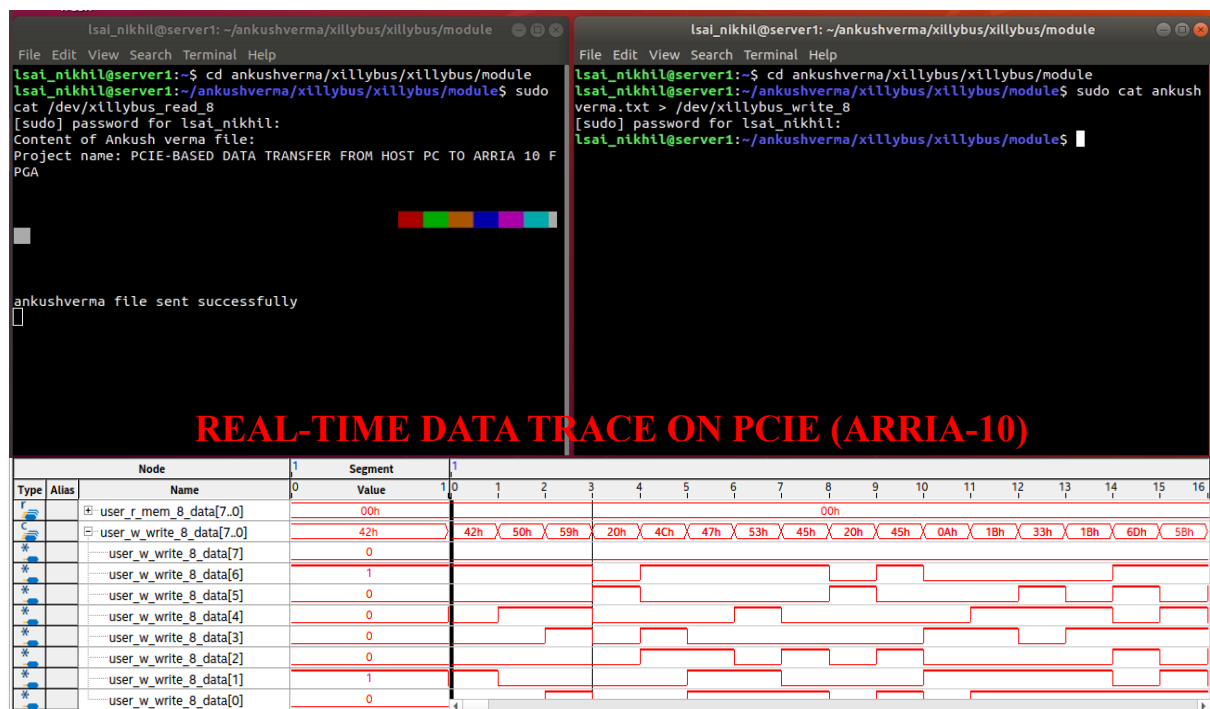


Figure 7.15 Transmitting And Tracing The File Data Using “Signal Tap Logic Analyzer”

It's important to note that the FIFOs within the FPGA are designed to prevent overflow or underflow by respecting the 'full' and 'empty' signals. Consequently, the Xillybus driver will ensure that the application waits until the FIFO is ready for I/O, resulting in a block or sleep of the user space program.

Additionally, there is another pair of device files, namely /dev/xillybus_read_32 and /dev/xillybus_write_32, which function with a 32-bit word granularity, similar to the FPGA FIFO. Performing the same test as described above with these files will yield similar behavior, but data will be transmitted in 4-byte chunks.

5. CONCLUSION:

In conclusion, the project successfully implemented PCIe-based data transfer between a host PC and an Arria 10 FPGA using Xillybus on a Linux operating system. Xillybus served as the interface for efficient and high-speed data communication between the two devices, with the FPGA handling data processing tasks and the host PC acting as the controller and data source/sink.

The project's journey began with downloading the necessary demo bundle and software from the Xillybus website, ensuring adherence to the legal permissions for evaluation purposes. The bitstream for the Intel Arria 10 FPGA was then created using Quartus, and the compilation of the kernel module followed, generating the essential "xillybus_core.ko" and "xillybus_pcie.ko" modules.

To ensure successful kernel module compilation, the project checked for prerequisites such as the Gcc compiler, "make" utility, and kernel headers, installing them when required. The installation of the kernel module was straightforward, and the user was provided with the flexibility to manually load and unload the Xillybus modules if needed.

By copying the udev rule file, users were able to modify the permissions of Xillybus device files, granting access to all users without requiring root privileges. With these preparations in place, the project proceeded to perform a "Hello, World" test, the trivial loopback test, which confirmed the successful loopback functionality of Xillybus.

Throughout the project, users were guided on how to troubleshoot any encountered errors, with valuable clues often found in log files such as "/var/log/syslog" or by using the "dmesg" command.

The successful implementation of PCIe-based data transfer using Xillybus on an Arria 10 FPGA demonstrated the effectiveness and efficiency of the solution for high-speed data communication between a host PC and FPGA. This project serves as a foundation for more complex and real-world applications utilizing Xillybus as a powerful development kit for FPGA-based designs. As technology advances, users can explore further possibilities and extend the capabilities of this system to address a wide range of data-intensive tasks and applications.