

Development of a PXI Express Peripheral Module and data transfer platform

by

Matthew David Bourne

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Electronic and Computer System Engineering.

Victoria University of Wellington
2013

Abstract

Magritek, a company who specialise in NMR and MRI devices, required a new backplane communication solution for transmission of data. Possible options were evaluated and it was decided to move to the PXI Express instrumentation standard. As a first step of moving to this system, an FPGA based PXI Express Peripheral Module was designed and constructed. In order to produce this device, details on PXI Express boards and the signals required were researched, and schematics produced. These were then passed onto the board designer who incorporated the design with other design work at Magritek to produce a PXI Express Peripheral Module for use as an NMR transceiver board. With the board designed, the FPGA was configured to provide PXI Express functionality. This was designed to allow PCI Express transfers at high data speeds using Direct Memory Access (DMA). The PXI Express Peripheral board was then tested and found to function correctly, providing Memory Write speeds of 228 MB/s and Memory Read speeds of 162 MB/s. Also, to provide a test system for this physical and FPGA design, backplanes were designed to test communication between PXI Express modules.

Acknowledgments

I would like to thank my supervisor Robin Dykstra for providing me the opportunity to undertake this project and his guidance over this time.

Thank you to my parents for their support and encouragement over the years to help me get where I am today. Especially to Dad for doing a last minute proofread of my entire thesis.

Lastly, big thanks to my flatmates at House Boston for their encouragement, support, feedback and for showing great patience when I bored them senseless with my musings and whiteboard scribbles on PCI Express, Address Spaces and Root Complexes.

Contents

1	Introduction	1
1.1	Computers and Expansion buses	1
1.2	Instrumentation	3
1.3	Motivation of Project	3
1.3.1	Magritek	3
2	Background	5
2.1	Present Magritek system	5
2.1.1	Existing Backplane	6
2.1.2	Requirements for new backplane	6
2.2	Choice of new Expansion Bus	7
2.2.1	Instrumentation buses	7
2.2.1.1	VMEbus eXtensions for Instrumentation (VXI)	7
2.2.1.2	PCI Express eXtensions for Instrumentation (PXI Express or PXIe)	7
2.2.1.3	LAN eXtensions for Instrumentation (LXI) .	8
2.2.2	Evaluation of solutions	8
2.3	PXI Express Overview	9
2.4	Outcomes and goals for Project	11
2.4.1	PXI Express Peripheral Module	16
2.4.2	PXI Express System Timing Board	18
2.4.3	Backplanes to test devices	18
2.5	PXI Express description	18

2.5.1	PCI Express	18
2.5.2	Instrumentation	20
2.6	PCI Express Architecture	21
2.6.1	Transaction Layer	22
2.6.2	Data Link Layer	23
2.6.3	Physical Layer	24
2.6.3.1	Logical Sub-Block	24
2.6.3.2	Electrical Sub-Block	27
2.6.4	Configuration of Devices	30
2.6.4.1	PCI Configuration Space	30
2.6.4.2	PCI Enumeration and Configuration	34
2.6.5	Packet Transactions	37
2.6.5.1	Transaction Layer Packet	37
2.6.5.2	Data Link Layer Packet	41
2.7	FPGA	43
2.7.1	VHDL	47
2.8	High speed signal board design	47
2.8.1	Characteristic Impedance	47
2.8.2	Length Matching	49
2.9	Linux Systems	49
3	Design and Implementation	51
3.1	Description of project	51
3.2	PXI Express Peripheral Module	52
3.2.1	Mechanical requirements for board design	54
3.2.2	PXI Express Power	56
3.2.3	PCI Express Signals	56
3.2.3.1	PCI Express Transmit and Receive signals	57
3.2.3.2	Clock Signal	57
3.2.3.3	Sideband Signals	57
3.2.3.4	PCIe signals used for Peripheral Module	58

3.2.4	PXI Express Signals	60
3.2.5	Creation of Libraries, Schematics and Footprints for Device	63
3.2.6	Schematics	64
3.2.7	Completed board	68
3.3	Design of System Timing Module	69
3.4	Implementation of PXI Express system	71
3.5	FPGA Design	72
3.5.1	Spartan-6 FPGA Integrated Endpoint Block for PCI Express	73
3.5.2	Format of PCI Express Design	78
3.5.3	PCI Express Data Transfer	79
3.5.4	Programmed Input/Output	81
3.5.4.1	Example Xilinx PIO Design	82
3.5.5	Direct Memory Access Transmissions	83
3.5.5.1	Xilinx Bus Master Performance Demonstra- tion Design for the Xilinx Endpoint PCI Ex- press Solutions	84
3.5.5.2	Other PCI Express DMA solutions	86
3.5.6	Problems and improvements to be made on sample Xilinx designs	86
3.5.7	FPGA design for use on PXI Express Peripheral Mod- ule	89
3.5.7.1	Memory Unit	95
3.5.7.2	Receive Unit	98
3.5.7.3	Transmit Unit	101
3.5.7.4	Implementation on FPGA	105
3.6	Backplanes for testing devices	107
3.6.1	Power	110
3.6.2	Reset Circuit	112
3.6.3	Signal integrity	112

3.6.3.1	Impedance Matching	113
3.6.3.2	Length Matching	116
3.6.3.3	Other steps	117
3.6.4	Component Selection	119
3.6.5	Completed PCBs	119
4	Testing and Design Evaluation	121
4.1	Testing of PXI Express Peripheral Module	121
4.1.1	Electrical testing of PXI Express Peripheral Module .	121
4.1.2	Mechanical testing of PXI Express Peripheral Module	122
4.2	FPGA Designs	123
4.2.1	Test Configuration	124
4.2.1.1	Test Configuration for SP605 Board	125
4.2.1.2	Test Configuration for designed PXI Express Peripheral Module	127
4.2.2	Software for testing	128
4.2.3	PIO Design	132
4.2.3.1	Root Port Model	132
4.2.3.2	Implementation on device	134
4.2.4	XAPP1052	140
4.2.5	FPGA Design for the PXI Express Peripheral Module	143
4.2.5.1	Testing on the SP605 board	143
4.2.5.2	Implementation on designed PXI Express module	144
4.2.6	Comparison of designs	147
5	Conclusions and Future Work	149
5.1	Further Work	150

Chapter 1

Introduction

In this chapter, the historical environment for the project is given. This details the relationship between computers and expansion buses, an overview of instrumentation and lays out the motivation and reasons for the project.

1.1 Computers and Expansion buses

Computers are used in numerous areas of our lives for a number of different purposes. Each computer contains a Central Processing Unit (CPU) which acts as the brains of the device for a number of applications. A standard computer makes use of a CPU to take care of computations, random access memory (RAM) to save results of computations and data for currently used applications. Some basic Input/Output (I/O) to allow a user to interact with a computer is also used (such as a keyboard, mouse and screen). However, such a setup may not be enough for all applications as the users' needs may vary greatly. Some applications may require dedicated hardware, for example a lot of desktop computers come equipped with a video card to do the video processing. Less commonly, computers may come equipped with sound cards to take care of the sound processing. In the past, almost all computers did not have integrated video, sound or even network controllers on board. Instead, a peripheral card was re-

quired to provide this functionality. Recently, a lot of these commonly required functions have been either integrated on the computer motherboard, or even provided on the dye of the CPU. However for niche applications, such as an industrial test system, the functions and equipment required would not be provided out of the box, and need to be provided by an expansion card.

These peripheral devices require the ability to communicate with the CPU, either to be informed of operations or for memory to be read or written in either direction. The expansion bus is what takes care of this communication between peripheral devices and the CPU. In fact, many controllers embedded on the motherboard of a computer make use of the expansion bus standard for communication, even if the particular form factor is not used.

The first computer expansion bus, the S-100 bus or Altair bus, was part of the Altair 8800 which is considered to be the very first personal computer [1]. Here each board performed a particular function of the device. There were separate CPU, memory, I/O and video boards which fitted into a mainly passive backplane or motherboard.

The Altair and its variants were mainly hobbyist machines, although some small businesses made use of them. The first successful home computer, the IBM PC released in 1981 [2], used the Industry Standard Architecture (ISA) bus for expansion cards. This was the first concerted effort to standardise the interface bus. Its bandwidth and functionality were somewhat limited but it paved the way for future designs which are still used today.

ISA was later superseded by the Peripheral Communication Interconnect (PCI) standard [3]. This brought about plug and play functionality which meant no complex configuration was required to enable communication. Instead the host computer auto configures the system on start up.

This ability to send information between devices on a computer sys-

tem is an important aspect of the device. For specialist application test and measurement systems, some manufacturers choose to design custom solutions to have full control over the communication, whilst others design to standards to maintain interoperability with other devices.

Commonly, peripheral communication systems (such as PCI) use a card edge connector, plugging into a socket on the motherboard, which keeps the size small. Others use a socket-pin type connection which consumes more space, but can give a more reliable connection.

1.2 Instrumentation

For most traditional PC applications which require an expansion card (such as graphics processing), timing is not a major factor. Due to this, the major PC peripheral interfaces PCI and PCI Express (PCIe) do not come with clock and trigger signals. However, quality instrumentation equipment does require this precise timing in order to get accurate measurements. Thus for instrumentation applications, whether a standard or custom solution, the backplane or motherboard will commonly come equipped with clock and trigger lines. These keep devices synchronised and are used to signal when particular events are to start or end.

1.3 Motivation of Project

1.3.1 Magritek

Magritek is a company based in Wellington, New Zealand which uses technology developed at both Massey University and Victoria University of Wellington. Magritek develops compact Nuclear Magnetic Resonance (NMR) and Magnetic Resonance Imaging (MRI) devices for use in lab and field environments. A lot of the equipment is modular where a NMR or MRI system will make use of a number of different cards connected to the

device. These cards may include RF transmitters, receivers etc. In order for effective communication and data transfer between peripheral cards, this requires a reasonable amount of bandwidth.

NMR is a physical phenomenon in which magnetic nuclei in a magnetic field absorb and re-emit electromagnetic radiation. Valuable information about the structure of the sample is gathered from the interaction between the magnetic moment of an atomic nucleus and an externally applied magnetic field. When an oscillating magnetic field is applied at a specific frequency (the Larmor frequency), transitions between different energy states (i.e. different orientations) occur. This is usually done by placing a coil around the sample to irradiate it with radio waves at the Larmor frequency. The nuclei in the sample absorb these radio waves and re-emits them at the Larmor frequency. This emission is the NMR signal which can be detected by the antenna. Only nuclei with a non-zero magnetic moment - those with an odd number of protons or neutrons - can undergo NMR.

Magritek presently makes use of their own custom backplane mechanism for the transfer of data between peripheral devices. However, it has limited bandwidth of around 5MB/s which is not sufficient for more advanced techniques such as multichannel MRI. Also, being a custom solution gives the limitation that their devices are not inter-operable with equipment manufactured by other companies. For these reasons, Magritek is looking to move to a new system for data transport. The purpose of this project was to develop some of the tools necessary to move in this direction.

Chapter 2

Background

In this chapter, the background of the project is described. This includes the choice of the PXI Express (PXIe) system, a brief description of the design direction chosen and an overview of important topics of this project. This includes details on the PXIe standard, FPGAs, high speed board design and the Linux environment.

2.1 Present Magritek system

In order to perform the data capture for NMR, data needs to be sent between the various components of the system. A typical piece of Magritek NMR hardware consists of a chassis with add in cards which all perform a particular task. An example device is shown in Figure 2.1. This includes a DSP based host board which controls the applications on the device and can possibly interface with a PC. A typical data transfer would be when the host device sends a waveform file to the transmitter to output. Similarly, the receiver could send data after collection to the host board. Thus a transmission mechanism to perform these Memory Read, Memory Write transfers as well as control tasks is a crucial part of the system. Details on present backplane solution used by Magritek and the requirements for a replacement system are given below.



Figure 2.1: An example of a Magritek Spectrometer - The Kea 2 [4]

2.1.1 Existing Backplane

Communication between boards is presently provided using Magritek's custom backplane solution. Communication between the devices is typically provided by using address and data pins of the Digital Signal Processor (DSP). These get buffered and sent out along the backplane. This allows the DSP device to read or write to external devices. The Peripheral Modules of the Kea system meet a 3U form factor and are connected using a standard 96 pin connector. This is connected with the backplane using a socket-pin connection. For more sophisticated techniques such as multi channel MRI, the system does not provide enough bandwidth. Also the parallel format of data transfer meant the amount of expansion available was limited.

2.1.2 Requirements for new backplane

With increased bandwidth and expansion being much desired aspects, moving to a standard, well defined architecture was the approach chosen. The advantage of moving to a standard is that the communication system has already been designed so does not need to be developed from

scratch. A form factor similar to their present system was desired so it would not involve too much chassis redesign. Thus a format which used pluggable expansion cards was required. The ability to send trigger and clock signals along the backplane was also required. Magritek's present devices are portable, rugged and used out in the field so a solution which provided these attributes was also desired. The intended architecture to be built for was x86 or ARM thus a peripheral bus which was supported by these architectures was required.

2.2 Choice of new Expansion Bus

The expansion bus industry was researched to find choices that fit the criteria. Some standard solutions such as PCI and PCIe provided increased bandwidth but did not give the desired form factor, or the trigger and clock signals required for instrumentation. Thus instrumentation buses were researched and evaluated. The choice was narrowed down to three options.

2.2.1 Instrumentation buses

2.2.1.1 VMEbus eXtensions for Instrumentation (VXI)

VXI is an established open instrumentation standard which was adopted by the IEEE in March 1993 (IEEE 1155). It implements the VMEbus standard for data transfer bus which provides 320 MB/s throughput [5]. However VXI modules are either 6U or 9U in size which is larger than desired [6].

2.2.1.2 PCI Express eXtensions for Instrumentation (PXI Express or PXIe)

PXIe is an instrumentation standard which evolved from the older PXI standard [7]. PXI used the older parallel PCI architecture for data trans-

mission, whilst PXIe moved to the newer PCIe standard. Cards meet the 3U eurocard form factor which is similar to the present Magritek design. The standard provides a plethora of high quality trigger and clock signals, both differential and single-ended. The use of PCIe as the transmission mechanism was somewhat future proof in terms of bandwidth availability. Further revisions of the PCIe standard have successively increased bandwidth where a maximum 250 MB/s is provided with revision 1.1 of the specification [8] and is now at 1GB/s as of version 3.0 of the specification [9]. It is a widely supported standard and is promoted by the PXI Systems Alliance whose sponsor members include two major instrumentation companies, Agilent Technologies and National Instruments.

2.2.1.3 LAN eXtensions for Instrumentation (LXI)

LXI is an instrumentation and data acquisition standard using the Ethernet protocol to send data [10]. LXI is integrated with the PXIe standard where PXIe devices can communicate as a part of a larger LXI system. However, LXI is used more for distributed systems than standalone systems which was not the direction of this project.

2.2.2 Evaluation of solutions

All the standards are supported by major vendors so would allow add-in cards to be run across different devices. It was decided PXIe gave the best combination of portability, features and transmission speed. Also, PXIe devices are typically less cumbersome due to their 3U height rather than VXI's 6U height. As the data transport scheme of PXIe is essentially the same as PCIe, this provides greater bandwidth than the VXI system.

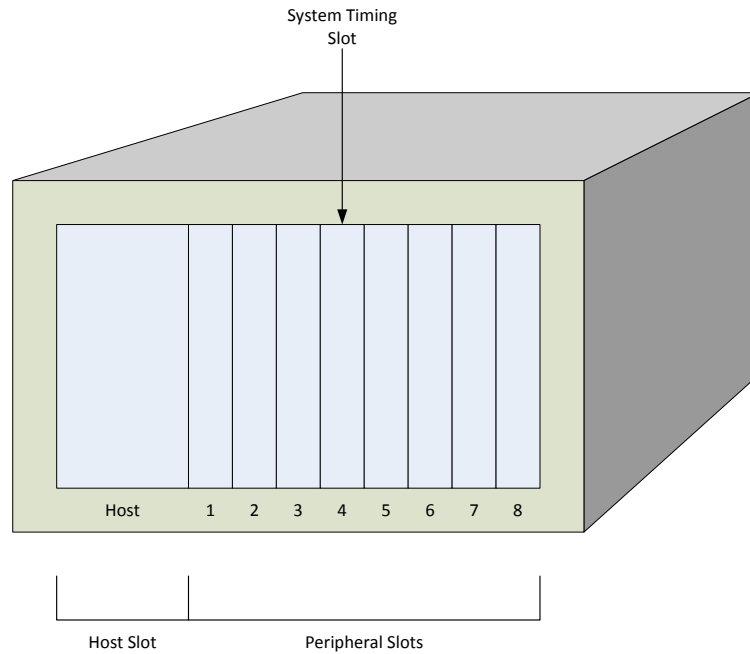


Figure 2.2: The layout of a typical PXI Express chassis

2.3 PXI Express Overview

PXIe is an instrumentation standard which combines the PCIe serial transmission protocol with a more robust form factor and the addition of clock and trigger signals. An example PXIe chassis which houses these boards and provides the mechanism for trigger and clock signals to be sent between devices is shown in Figure 2.2. Peripheral devices thus get installed in this chassis and communicate through data being sent along the backplane. The backplane of this chassis distributes all the data, clock and trigger signals to the installed modules. A diagram of the distribution of these signals along an example backplane is given in Figure 2.3. Here, slots 1 through 4 show PXIe slots whilst slots 5 through 8 show legacy PXI slots.

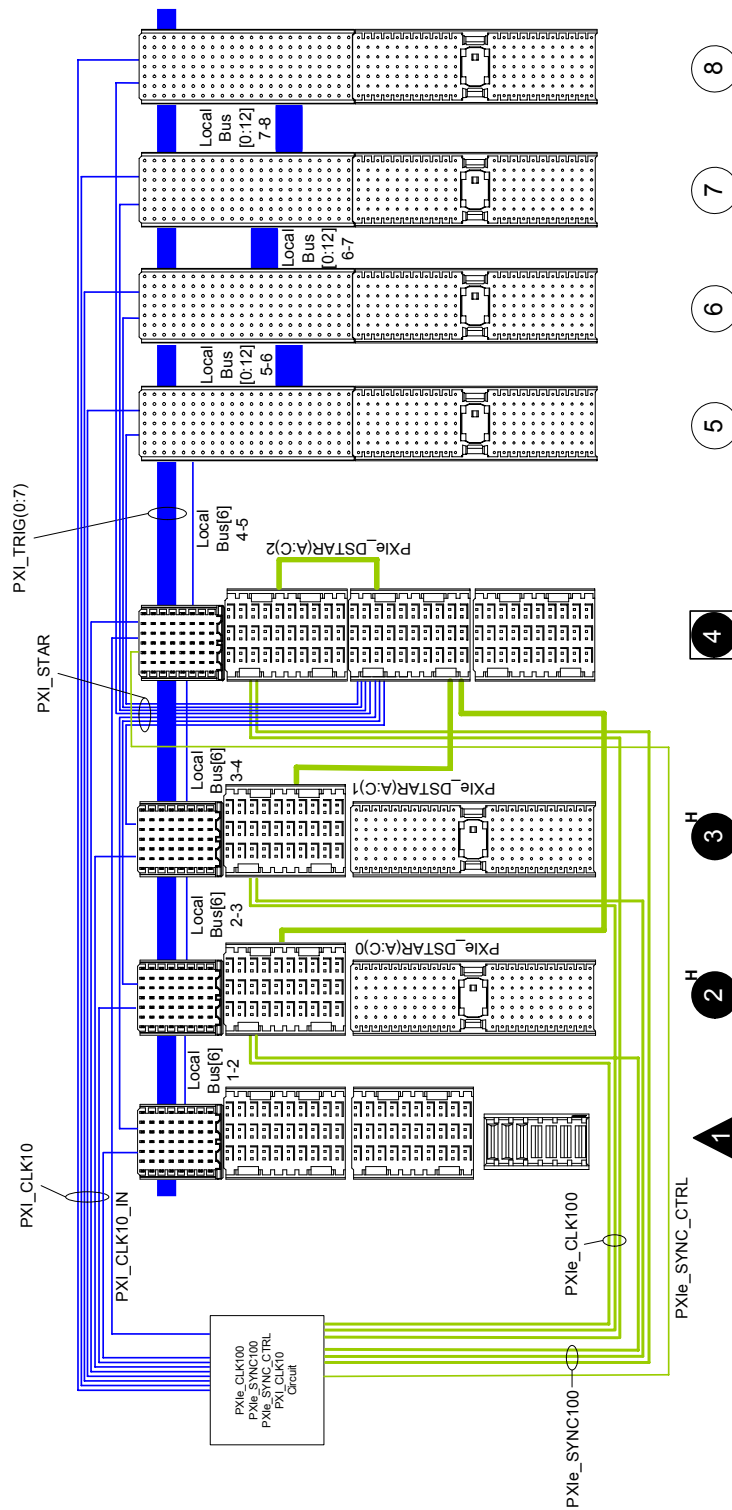


Figure 2.3: Distribution of PXI Express signals on the backplane [11]

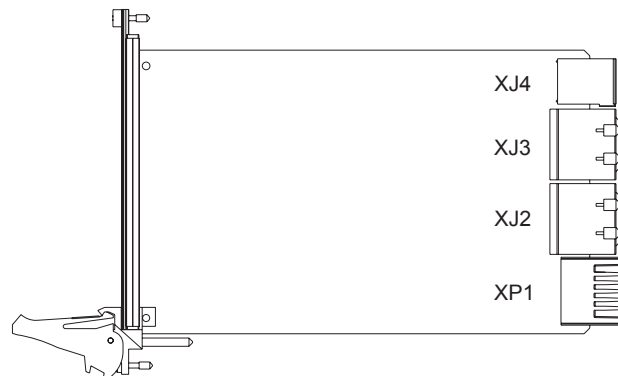


Figure 2.4: The layout of a PXI Express System Module [11]

As can be seen in Figure 2.3, a PXIe system consists of PXIe System Module (situated at the far left of the chassis) which communicates with the other peripheral devices connected in the chassis. The layout of a PXIe System Module and System Slot are shown in Figures 2.4 and 2.5.

Peripheral devices are added by plugging a Peripheral Module into a Peripheral Slot or Hybrid slot on the chassis. The basic Peripheral Module, Peripheral Slot and Hybrid Slot are shown in Figures 2.6 to 2.8 respectively. Peripheral Slots only allow PXIe Peripheral Modules to connect whilst Hybrid slots also allow legacy PXI boards to be connected. PXIe systems commonly use Hybrid slots for the flexibility they provide.

Lastly, the other allowable PXIe device is for System Timing purposes. The System Timing Slot is shown in Figure 2.9. The System Timing Board controls the timing of the device. This includes the distribution of star clock and trigger signals to the other peripheral modules.

2.4 Outcomes and goals for Project

PXIe is an excellent technology giving the best of high speed serial data transfer along with high quality clocking and trigger resources. Although it is an open standard its relatively niche use means that it is a rather un-

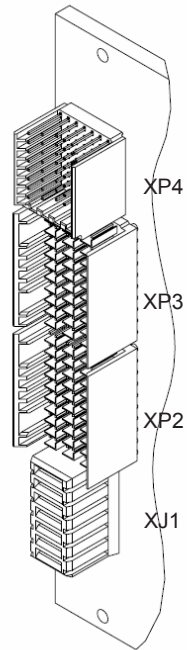


Figure 2.5: The layout of a PXI Express System Slot [11]

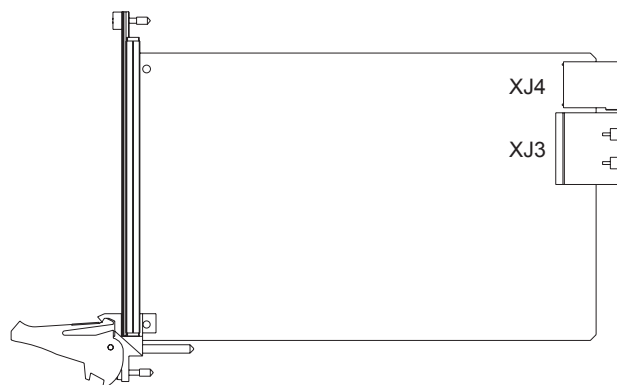


Figure 2.6: The layout of a PXI Express Peripheral Module [11]

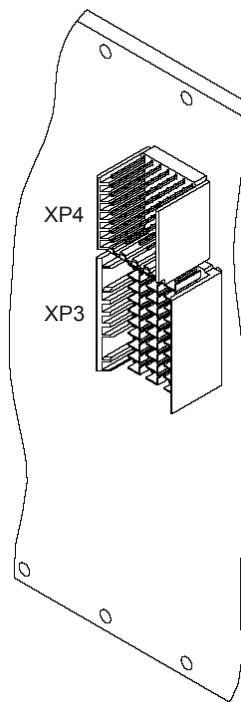


Figure 2.7: The layout of a PXI Express Peripheral Slot [11]

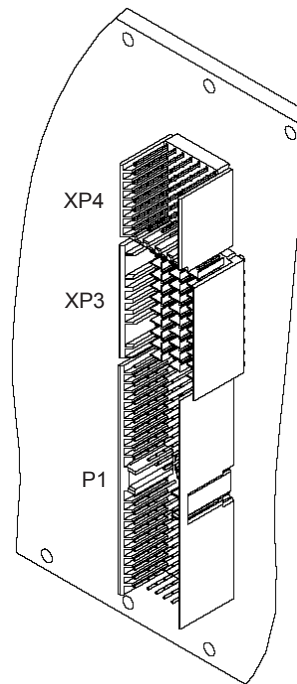


Figure 2.8: The layout of a PXI Express Hybrid Slot [11]

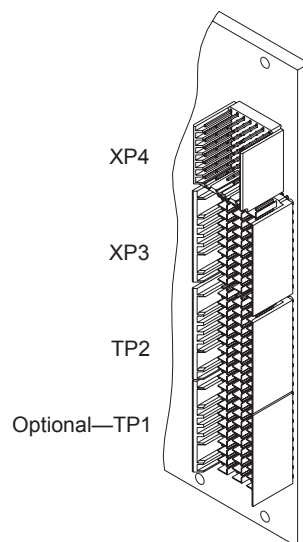


Figure 2.9: The layout of a PXI Express System Timing Slot [11]

exploited market. As it stands, National Instruments and Agilent have become the two major players where they dominate the market. Thus the challenge was providing a device that met the standard at a relatively low cost and ease of use. Presently, if custom PXIe solutions are required, National Instruments provides the FPGA based FlexRIO board. This way the PXIe functionality is provided out of box. However, these are relatively high cost devices which do not offer much flexibility. If a high sensitivity design is required this would need a custom board design which could not be achieved using a breadboard or veroboard. Also, such a solution would only be useful for one off designs.

Test and Measurement has traditionally been an area where proprietary formats have dominated. Due to the complexity of the equipment, the high prices have just been attributed as a fact of the devices. However, there is a definite opening in the market if PXIe devices could be offered at a lower price point. An open source system could be provided with a basic board design, FPGA design and device driver. The challenge was making a high performance open system where no purchased IP had to be used. Thus the major cost of the device would be the board construction and components used, rather than requiring any purchased IP from outside of the company, producing devices at the lowest cost possible.

PXIe devices include a system controller card which includes the CPU, RAM and I/O which the user makes use of to communicate with the device. Magritek's present devices use a similar format where the user communicates with a host board which in turn communicates with the peripheral devices. The move to PXIe would allow for an ARM processor based PXIe System Module to be designed in the future. ARM processors are low cost, low power devices for which PCIe solutions are available [12].

A PXI Chassis complete with a PXIe System Module was required to communicate with peripheral devices. For this, a National Instruments (NI) PXIe-1062Q chassis fitted with a NI PXIe-8101 Embedded Controller card was sourced. The PXIe-1602Q chassis is shown in Figure 2.10. The



Figure 2.10: National Instruments NI PXIe-1062Q Chassis [13]

connectors on the backplane indicate this chassis came equipped with slots for two Peripheral Modules and a System Timing Board. It also had slots for older, legacy PXI cards. However, these were not the direction of the project and were largely ignored.

The move to this open PXIe standard involves a number of steps including the development of a PXIe chassis, controller card and Peripheral Module. The first step in moving to this system was producing a PXIe Peripheral Module for NMR use which is detailed below.

2.4.1 PXI Express Peripheral Module

For this project, the design and implementation of a PXIe Peripheral Module was focussed on. The Peripheral Module would operate as a NMR

transceiver. The device was based off a present Magritek device which makes use of the custom backplane. The device were adapted to meet the PXIe standard as well as adding improved Digital to Analog Converters (DACs), Analog to Digital Converters (ADCs) and other features.

As well as developing the physical PXIe Peripheral Module, the PXIe communication on board had to be implemented. The base PCIe functionality can be provided using a dedicated IC [14]. In this case, the required PCIe signals get connected directly to the IC which takes care of the PCIe link. However, this would then have to be interfaced with the rest of the logic on board which would complicate the board design.

The PXIe configuration can also be performed by an FPGA. FPGAs (which will be covered in further detail later) allow dedicated logic blocks to be implemented. No FPGA vendor provide complete PXIe solutions, however Xilinx and Altera both provide PCIe solutions for their devices [15, 16]. These have the potential to reduce development time and simplify the board design as few external ICs would be required, and everything could be configured inside the FPGA.

Implementing the PXIe protocol on an FPGA was decided as the best design option. It simplified the board design and made development of the PXIe system simpler. Magritek's present devices all make use of FPGA designs so this would allow the fastest development time for a PXIe Peripheral Module for NMR applications.

The physical device along with the FGPA design was intended to meet to following requirements.

- Used a Xilinx FPGA (specifically the Spartan-6) as the controller of the board.
- An open PCIe FPGA design system where no purchased or node-locked IP was used and all source code was available.
- Open source device driver and application used.

2.4.2 PXI Express System Timing Board

A PXIe System Timing Board was also designed, which would control the timing, clocks and triggers to the peripheral devices. It also met the requirements given for the Peripheral Module.

2.4.3 Backplanes to test devices

Along with the boards, test backplanes were designed to test designed modules along with FPGA based PCIe development kits. These would allow communication between a host and peripheral system to test that the devices were working as required. The test backplanes were designed to test PXIe modules as well as standard PCIe boards.

2.5 PXI Express description

As explained earlier, PXIe can be seen as the instrumentation extension of the PCIe serial transmission protocol. Thus data is sent in the same way as a PCIe transaction. The description of the standard has been broken into a description of the PCIe data transmission system and the instrumentation signals which make up the PXIe standard.

2.5.1 PCI Express

For more than a decade in computer technology, the preferred computer bus for hardware devices was Peripheral Component Interconnect, most commonly known as PCI. This became the standard for expansion cards in the mid 1990s where it provided much higher bandwidth than the older ISA standard of the time [17]. PCI is a parallel architecture which has a max transfer rate of 133 MB/s, 266 MB/s or 533 MB/s depending on the frequency and bus width [3]. Peripheral devices are connected to this PCI

bus which can host up to five devices reliably. Whilst this provided adequate bandwidth up until the late 1990s, the turn of the decade brought about the requirement for dedicated high bandwidth which PCI could not provide. This was due to number of reasons, one being the limitations inherent in the parallel format. This meant that frequencies could not be pushed much higher than that which were presently used without massive changes to board design. Also, the shared bus meant that dedicated bandwidth was not provided. The maximum bus transfer rate was shared between all devices.

To get around these problems in PCI, a new transmission system was devised in PCIe which was released in 2004 [8]. Here, devices are connected through a high speed serial connection. Each device has a connection to the central controller, known as the Root Complex. This is provided either directly or through a bridge. They do not share data and address lines as seen in PCI, so there are no limitations on concurrent access across multiple Endpoints. The serial scheme uses dedicated upstream and downstream links which use differential signalling. Data can be sent and received simultaneously, known as full-duplex transmission.

PCIe is a packet based technology where all information is sent across the same link. There is no need for separate address, data, control, interrupt or clock signals resulting in much more efficient use of ports. Also, the design of PCIe allows for more bandwidth when required. A PCIe link is built around couples of differential serial point-to-point connections known as lanes. More lanes can be provided to give more bandwidth when the application requires it. Revision 1.1 of the specification states that along one lane, 250 MB/s is provided [8]. If only one of these lanes is used, this would give a x1 PCIe link. A graphics card usually makes use of a 16 lane (or x16) link which gives 4GB/s bandwidth in the 1.1 specification. This scalability allows a variety of devices, each with their own bandwidth demands, to be used over a PCIe link. For instance, a device may not require extremely high data speeds, but does need the dedicated

bidirectional bandwidth which PCIe provides. For such a device, a x1 link would be adequate.

Though PCIe was designed to succeed PCI, it was also designed so the two maintained compatibility. As of 2012, PCI devices are still being produced. Some devices do not require the increased bandwidth PCIe provides and have opted to stay with the older PCI system. The two remain interoperable through software compatibility. PCI was the industry standard for around ten years and brought with it a lot of strengths. PCIe performs configuration in a similar method to PCI. They both make use of the PCI Configuration Space. This is the underlying way that devices get automatically configured when inserted into the bus. This is known as plug-and-play. However it is implemented in such a way that compatible devices can also make use of the more advanced PCIe features.

2.5.2 Instrumentation

Where a PXIe design differs from a standard PCIe design is that it is an instrumentation module which comes in a much different form factor and includes additional signals. PXIe uses the form factor specified by CompactPCI Express (cPCIe) standard and provides additional instrumentation signals. The cPCIe form factor uses metric spacing and gives a more robust mechanical form factor than desktop PCI.

The connector design defined in the standard specifications of PCI and PCIe is fit for use in standard desktop computers. However, such a design does not meet the requirements for industrial applications. Due to this, the form factor CompactPCI was developed as an extension to PCI and the specifications released in 1997 [18]. The advantages over conventional PCI include a vertical card orientation for improved cooling, improved shock and vibration characteristics, high performance, and robust connectors.

When used for instrumentation applications, the connectivity provided by CompactPCI still had some limitations. Whilst the CompactPCI form

factor is a lot more rigorous than PCI, a lot of measurement and automation systems require precise timing which CompactPCI cannot provide. For this reason, the PCI eXtensions for Instrumentation (PXI) standard was developed and released, also in 1997. This format takes the advantages of the CompactPCI form factor and provides timing, synchronization and triggers to be sent along the backplane [19].

With the development of PCIe, the benefits of the new technology were desired in a more robust form factor. The cPCIe specification was released on June 27, 2005 [20] which was soon followed by the PXIe (PXIe) standard [11]. PXIe also improves on the clocking and triggers provided in the original PXI standard by providing higher performance differential clocking and trigger signals.

PXI and PXIe provide various clocking and triggering signals to the system for use in instrumentation applications. These devices usually include a central System Timing card which controls length matched (or star) trigger and clock signals to the other peripheral cards where this is required.

As well as the form factor, these described systems also differ from a traditional PC as the CPU and memory are not found on the motherboard. Instead, cPCI, cPCIe, PXI and PXIe all include a System Card which houses the CPU, memory and hard drive for the system. This card also drives the clock source to the peripheral cards for their PCIe interconnect. Thus the backplane in these systems is much more passive than a standard PC motherboard.

2.6 PCI Express Architecture

The system architecture can be explained using a three layer abstraction model which is shown in Figure 2.11. The three layers which build a PCIe transaction are the Transaction Layer, the Data Link Layer and the Physical Layer. The serial architecture is packet based where each layer deals with

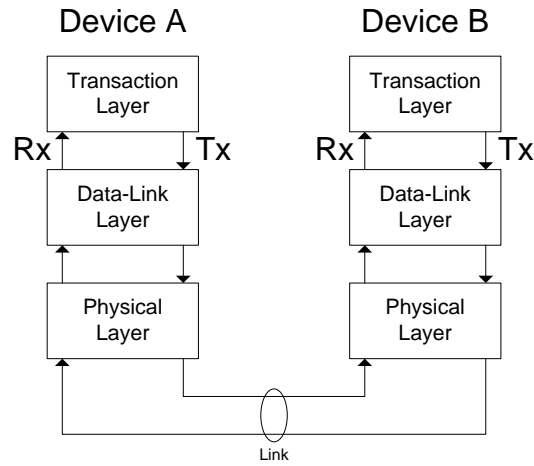


Figure 2.11: The three-layer abstraction of the PCI Express protocol

a different level of the packet, like a typical network stack. These layers become encapsulated in one another which is shown in Figure 2.12. These layers and their interactions are detailed below.

2.6.1 Transaction Layer

The top layer of the PCIe protocol is the Transaction Layer. It receives requests or data packets from the device core and serves as the starting point for turning these into PCIe transactions. The Transaction Layer (TL) does this by creating a Transaction Layer Packet (TLP) as shown in Figure 2.12. This comprises of a header which outlines the details of the packet, a data payload and an optional end-to-end cyclic redundancy check (CRC) which is used to test the integrity of the TLP. As well as starting the process for PCIe transactions on the transmission side, it also provides the reverse process on the receiving side where it passes the data on to the device core in the correct format.

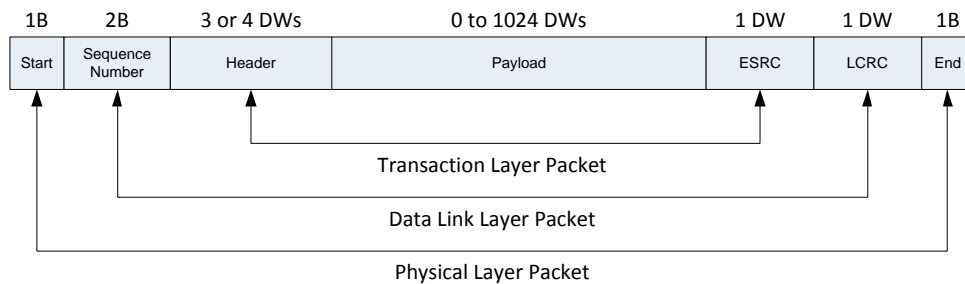


Figure 2.12: The encapsulated design of the PCI Express packet structure

For example say the core of some device A decides it wants to read the memory of device B. The Transaction Layer turns this information into a Memory Read Request and sends this packet onto the Data Link Layer. Later, the Transaction Layer of device A receives a completion packet from device B. This is passed up from the receive side of Data Link Layer which the Transaction Layer then decodes and passes this data on to the device A core.

2.6.2 Data Link Layer

The middle layer of the PCIe protocol is the Data Link Layer (DLL). The function of the DLL is to keep the link performing well and provide error detection and correction. After packets are created in the Transaction Layer, they get passed along to the DLL which adds a sequence number and LCRC error checker. This then gives a Data Link Layer Packet (DLLP). Thus, when packets get received, the Data Link Layer checks their integrity by verifying the CRC. The sequence number is also checked to ensure the packets have arrived in the correct order.

As well as further encapsulating TLPs, the DLL also generates its own packets to perform particular tasks it is responsible for. These tasks performed in the DLL include flow control, power management and ensuring

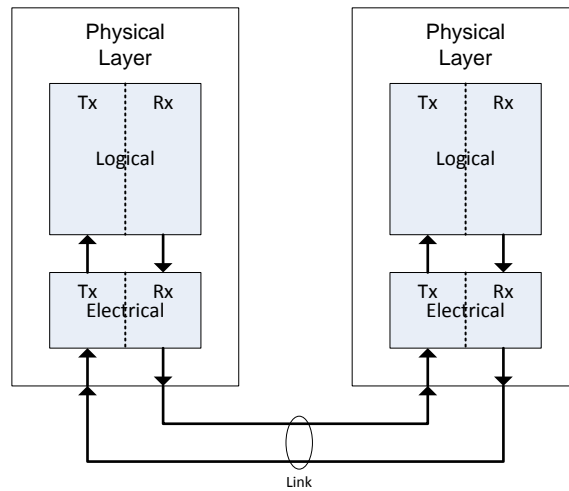


Figure 2.13: Physical Layer of PCI Express

reliable delivery of TLPs between Endpoints via acknowledgements.

2.6.3 Physical Layer

The Physical Layer provides the digital and analogue circuitry necessary to configure and maintain the link [17]. The Physical Layer comprises of two different sub-blocks: the Logical and the Electrical sub-blocks as shown in Figure 2.13. These are detailed below.

2.6.3.1 Logical Sub-Block

The transmit function of the logical sub-block provides four important processes: data scrambling, 8-bit/10-bit encoding, packet framing and serialisation. The receive function is to provide the reverse processes.

Data Scrambling Data scrambling is used to reduce the possibility of electrical resonance on the link. Resonance is usually caused by repeated patterns or bit sequences. Scrambling breaks up these repeated patterns

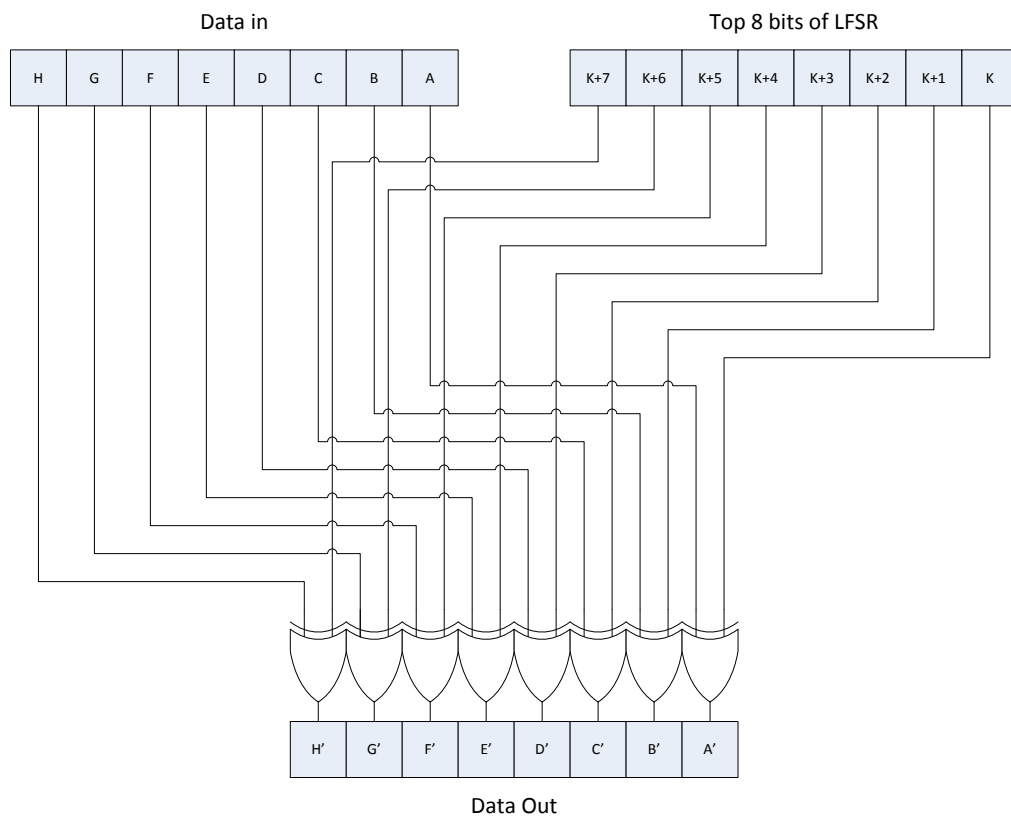


Figure 2.14: PCI Express Data Scrambler

meaning no single frequency component is transmitted for a long period of time. This spreads the frequency spectrum resulting in a 'whitening' of the bit stream (known as spread-spectrum). The scrambling is done by performing an XOR operation on the data with a pseudo-random number generator, implemented using a Linear Feedback Shift Register (LFSR). On the receive side this data is then descrambled using another LFSR. Synchronisation between LFSRs so that the same data is received on both sides of link is performed on start up. This scrambler is shown in Figure 2.14.

8b/10b Encoding Encoding is also applied to the data in the logical sub-block. This is used to embed the clock cycle into the data stream which

eliminates the need for an external clock signal to recover the data. Due to this embedded clocking, the same stringent routing required in PCI is no longer necessary. In a parallel system such as PCI, there is a strict requirement that all signals must arrive at the same time or a mismatch will occur. To meet this requirement, signal lines are snaked across the board to keep the track lengths the same meaning signals reach the destination at the same time. The problem becomes exacerbated at higher frequencies, where very small differences in track lengths can correspond to signals being one or two bits out of phase with one another. It is for this reason that PCI cannot be pushed much further than 33 MHz it operates at. With the serial, embedded clock system of PCIe, the same timing problems are not apparent which allows much higher frequencies to be used. It also improves board routing, as signals do not have to be snaked which saves board space and simplifies the routing process.

The encoding also provides DC balance on the data line as it keeps the number of 1s and 0s as equal as possible. Removing the DC component prevents capacitance along the lane from being over charged. This means the ability to change from one logic level to another is not hindered which therefore reduces inter-bit interference.

In the PCIe 1.1 specification the encoding is performed using 8b/10b encoding where every 8 bits of data was encoded to a 10 bit data stream. This forces a minimum number of bit-level transitions within a particular signal. The 8-bit/10-bit encoding format allows at most five bits of the same polarity to be transmitted before a bit transition must occur. This gives sufficient 0-to-1 and 1-to-0 transition density so that the clock signal can be recovered on the receive device using a phase-locked loop (PLL). It is performed by breaking a byte up in to a five-bit block and a three-bit block which get encoded to a six-bit block and four-bit block respectively using table lookups. The encoding, whilst adding a 25% overhead to the transmission, is worth it for the benefits gained from embedded clocking. A description of the 8b/10b encoding scheme is given in Figure 2.15. The

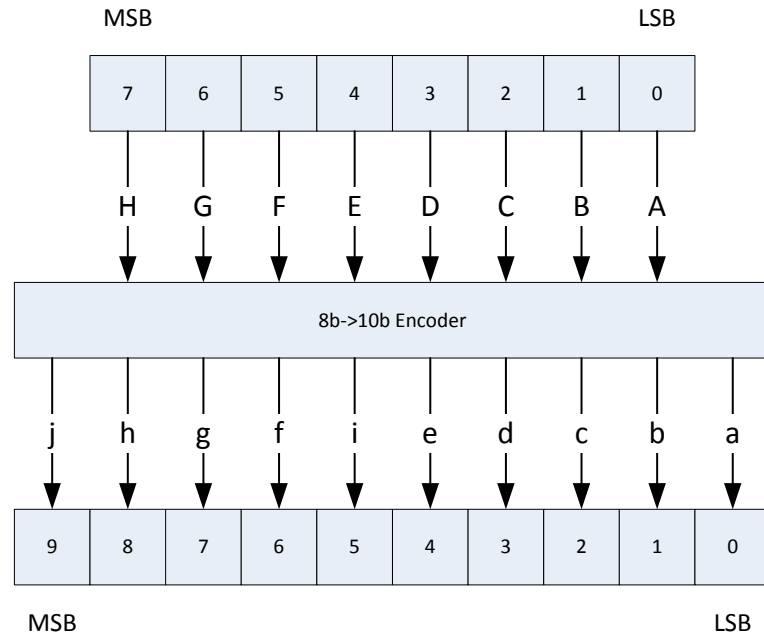


Figure 2.15: 8b/10b Encoding

reverse of this process is applied on the receive side to decode the original data. Encoding is used in all versions of the PCIe protocol. A similar process, but with less overhead, is used in PCIe 3.0 which uses 128b/130b encoding [9].

2.6.3.2 Electrical Sub-Block

The electrical sub-block functions as the delivery mechanism along the physical link. On the transmit side this consists of converting the serial bit stream to electrical signals to send on along the link. The receive side detects this electrical signalling and recreates the bitstream.

One of the major limitations of PCI is its use of single ended signalling where all signals are referenced to ground. Noisy ground planes can effect the signal integrity. At high frequencies the noise generated increases as do the attenuation effects, thus PCIe employs a different type of signalling.

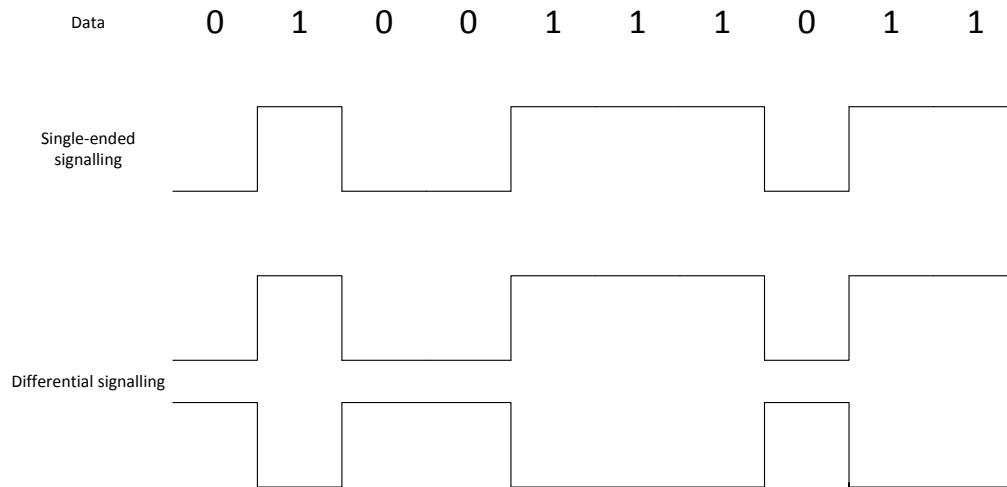


Figure 2.16: Comparison of single ended and differential signalling

The transmission scheme is known as high-speed Low-Voltage Differential Signalling (LVDS). This is where the signal is represented by the difference between two signals (a differential pair). The difference between single ended signalling and differential signalling is shown in Figure 2.16. A PCIe differential pair consists of two signals, D+ and D-. A logical one is sent out by driving the D+ lane high and the D- lane low and a logical zero is signalled by driving the D+ signal low and the D- signal high. In this way, no ground reference is required which provides greater signal integrity and noise immunity.

The high frequency differential signals also make use of a balanced transmission line. This is employed in the PCIe LVDS transmission scheme. The characteristic impedance of a link is $100\ \Omega$. Balancing the transmission lines helps the rejection of external noise in the circuit and minimise reflections along the link. This is matched to the transmitter and receiver.

To prevent any DC offset voltage appearing on these differential lines, PCIe requires a decoupling capacitor to be used on the transmit side of the differential pair. The decoupling is provided using a 75 to 200 nF capac-

itor either on the PCIe device itself or on the motherboard. This greatly simplifies the buffer design for PCIe devices as they do not receive signals which have a large DC offset.

In the PCIe Revision 1.1 specification, the bitrate of transmission is 2.5Gb/s. To meet requirements this clock needs to be accurate to 300 ppm jitter about its centre frequency [8]. This clock is usually derived from a 100 MHz clock signal distributed along the system board which is then multiplied to 2.5GHz by way of a PLL. This 2.5GHz clock does create some EMI radiated noise however, and so the option of Spread spectrum clocking (SSC) is given. This prevents a noise spike at 2.5GHz by spreading over a small frequency range about 2.5GHz by way of a small 0% to -0.5% modulation using a modulation rate between 30KHz and 33KHz.

As PCIe operates at very high frequencies, the period of each bit is very small. Thus the capacitive effects on the link are very apparent. This can cause inter-symbol interference (ISI) where a change in the polarity of the bit can be effected by the previous bit(s). In order to overcome this, a method known as de-emphasis is used. As detailed earlier, the number of consecutive bits allowed is 5. Say for example that five logical 0s are sent across the link, followed by a 1 and then another 0. Because of capacitance on the link, transition to the 1 may not happen quickly enough to register the logical 1 occurred before the next 0. The use of de-emphasis means that every new bit (i.e. one that was different from the one that came before it) would be transmitted at a higher power than any subsequent bits of the same polarity. For example, the signal transmitted five 0s in a row and then a 1, the four 0s that follow the first one would be at reduced power and then a boosted power 1 would be sent. This allows the system to discharge faster and thus the 1 can be transmitted successfully, overcoming any issues with ISI. An example of this de-emphasis system is shown in Figure 2.17. As can be seen, on every bit transition from 0 to 1 or 1 to 0, a boosted voltage is used. Then for any bit which is the same polarity as the previous bit, the standard voltage level is used.

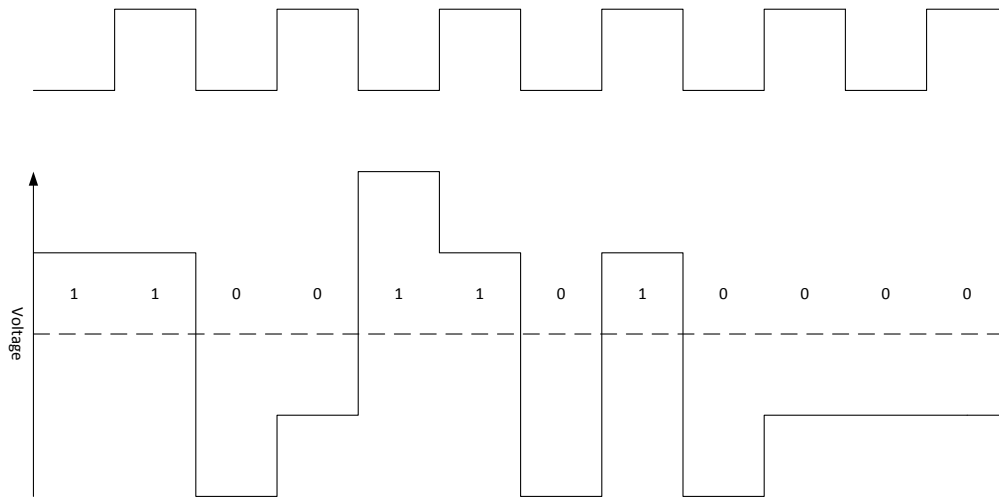


Figure 2.17: Description of de-emphasis system

2.6.4 Configuration of Devices

One major strength of PCI was its software mechanism which brought plug-and-play, the ability to add a new component to a system without requiring manual configuration. PCIe kept this system which makes use of the PCI Configuration Space. The PCI Configuration Space, enumeration and configuration are detailed below.

2.6.4.1 PCI Configuration Space

The PCI Configuration Space is how PCI and PCIe devices are initialised and configured for use in the system. This is a space with standardised registers which allows the auto configuration of Peripheral Modules through the reading and writing of particular registers. The format of the PCI Configuration Space depends on the type of device. The Configuration Space Header for a PCI Type 0 (Non-Bridge) device is shown in Table 2.1.

The Device ID and Vendor ID are useful for the operating system to detect the particular device installed. The Vendor ID is a particular code which PIC-SIG members are provided to identify their devices. The De-

Bits [31-24]		Bits [23-16]		Bits [15-8]		Bits [7-0]		Reg
Device ID				Vendor ID				0x00
Status				Command				0x04
Class Code						Revision ID		0x08
BIST	Header Type		Latency Timer		Cache Line Size			0x0C
Base Address Register 0 (BAR0)								0x10
Base Address Register 1 (BAR1)								0x14
Base Address Register 2 (BAR2)								0x18
Base Address Register 3 (BAR3)								0x1C
Base Address Register 4 (BAR4)								0x20
Base Address Register 5 (BAR5)								0x24
Cardbus CIS Pointer								0x28
Subsystem ID				Subsystem Vendor ID				0x2C
Expansion ROM Base Address								0x30
Reserved						Revision ID		0x34
Reserved								0x38
Max latency	Min Grant		Interrupt		Cache Line Size			0x3C

Table 2.1: PCI Configuration Space Header for a PCI Type 0 (Non-Bridge) Device

vice ID is a description of the function of the device. These together give the "PCI ID" and together identify the device model. The Vendor ID is the chip manufacturer and the Subsystem Vendor ID is the card manufacturer.

The Header Type describes the remaining 48 bytes of the header, depending on the function of the device. Type 1 headers are for the Root Complex, switches and bridges. PCIe Endpoints are Type 0.

The Base Address Registers (BARs) are used to inform the device of its address mapping by writing configuration commands to the PCI controller.

PCIe makes use of the standard PCI Configuration Space and extends it to include special PCIe functions. The standard 256 bytes of PCI Configuration Space can be used by legacy operating systems which do not support the advanced features. All PCIe devices must implement the PCI 3.0 Compatible Configuration Space Header and the PCIe Capability structure inside this 256 bytes of configuration space. As well as the PCI Express Capability structure there are also further registers known as the PCI Express Configuration Space, past the first 256 bytes. These are known as the PCI Express Extended Capabilities. This must start immediately after the 256 byte PCI Configuration Space. Here pointers are included to the rest of the registers which include, serial number and power management registers. This configuration space (including the 256 bytes PCI Configuration Space) is 4KB in total. A diagram of this layout is given in Figure 2.18

Inside the PCI Express Capability structure are parameters that are necessary for PCIe transactions, that are not required for standard PCI devices. As with the standard PCI Configuration Space, the format of the header depends on the type of device. For PCIe Endpoints, the format is as shown in Table 2.2.

Thus when the root complex communicates with PCIe Endpoints rather than simple PCI devices, it also reads these registers to enable communication with the devices. Important parameters such as the maximum payload size are read to enable successful PCIe transmission which is not

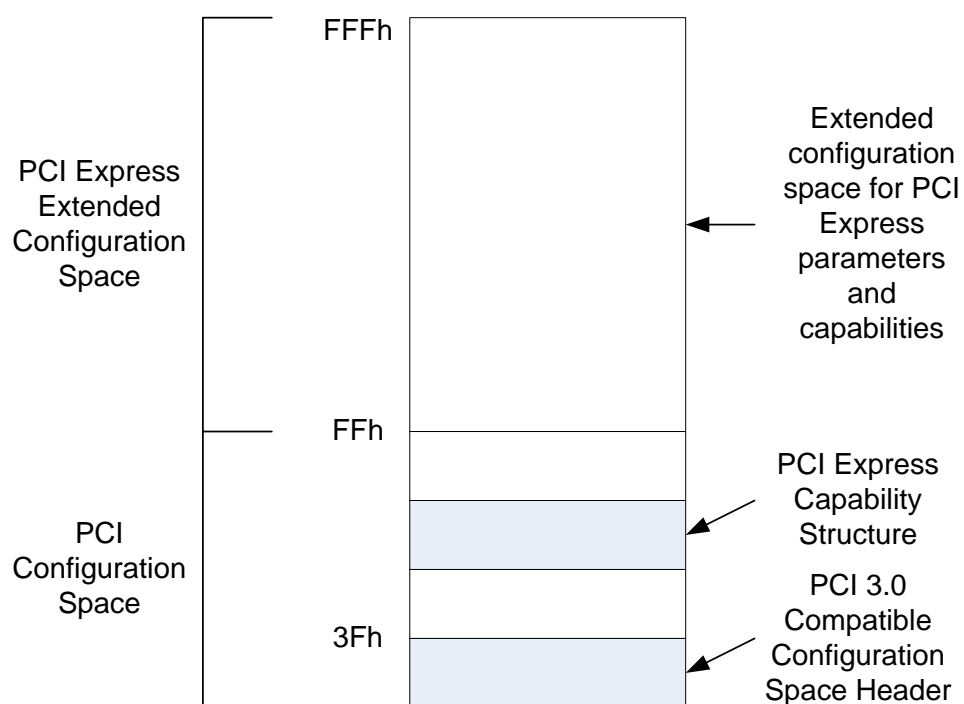


Figure 2.18: Layout of the PCI Express Configuration Space

Bits [31-24]	Bits [23-16]	Bits [15-8]	Bits [7-0]	Byte Offset
PCI Express Capabilities Register		Next Cap Pointer	PCI Express Cap ID	0x00
Device Capabilities				0x04
Device Status		Device Control		0x08
Link Capabilities				0x1C
Link Status		Link Control		0x10

Table 2.2: PCI Express Capability Structure of Endpoint Devices

required for PCI. The maximum payload size must be communicated between all devices to ensure that an unacceptable packet size is not sent. PCIe devices must ensure a maximum payload size of at least 128 bytes is provided so packets up to this size can be guaranteed to be accepted.

2.6.4.2 PCI Enumeration and Configuration

The configuration of the devices is accomplished by making use of the standard format of the PCI Configuration Space. During start up, the configuration space of each device is attempted to be read to detect its presence.

Initially only Bus 0 in the Root Complex has a bus number assigned. The header of each device is read to learn whether it exists or not. If it does and finds a bridge, it will search further down, then after reaching the bottom will traverse back up the tree. It performs a depth first search of all devices. On startup the system is unaware of the devices installed and thus the system looks as shown in Figure 2.19. There is no direct method for the BIOS or OS to determine which slots have peripherals or devices installed so the PCI buses need to be enumerated. In order to enumerate the peripheral devices, first the Device and Vendor ID registers are read. If nothing is read, the slots are taken as empty. The bus master performs an abort and returns all 1s in binary (0xFFFFFFFF). This is an invalid Vendor ID/Device ID combination so the device driver can resolve that the device does not exist. When a successful read occurs, it then writes to the Base Address Registers (BARs) and reads back to investigate the memory locations on the Endpoint.

The lower bytes of a BAR indicate the type of memory device initialised. What these lower bytes indicate is shown in Table 2.3. For memory elements, BARs must be at least 128 bytes wide. BARs are 32 bits wide, so for 64 bit memory address, this takes up two adjacent bars.

The Base Address Register configuration is performed as follows. Once a device is discovered, the host writes 0xffffffff to the BAR and the BAR is

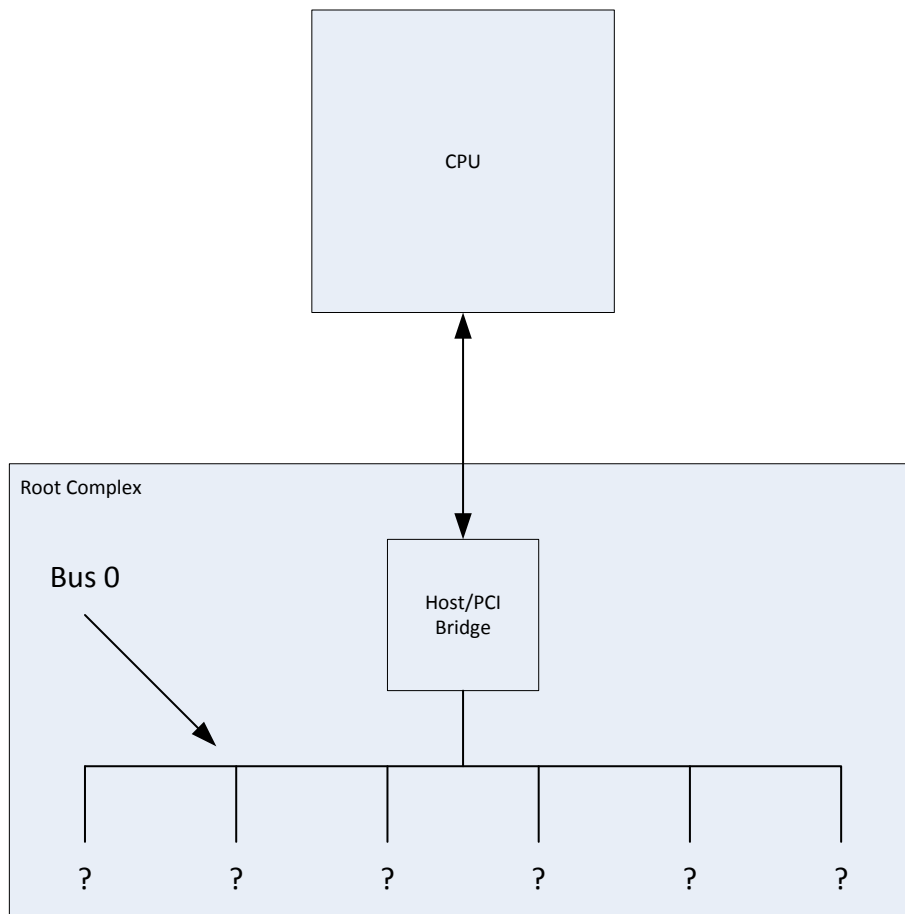


Figure 2.19: System Topology on Start-Up

Bits	Description	Values
For all PCI BARs		
0	Region Type	0 = Memory 1 = I/O (deprecated)
For Memory BARs		
2-1		00 = any 32 bit 10 = any 64 bit
3	Prefetchable	0 = no 1 = yes
6-4	Reserved	
31-7	Base Address	128-byte aligned
For I/O BARs (Deprecated)		
1	Reserved	
31-2	Base Address	4-byte aligned

Table 2.3: Base Address Register Bit Assignments

then read back. For example, after 0xffffffff gets written to a BAR, say 0xfffff800 is read back. The zero bits give the assigned memory size where $16 \times 16 \times 8 = 2048$ bytes. Once this memory space is known, the BIOS then programs the memory mapped and I/O port addresses the function will respond to into the BAR. These addresses are valid whilst the computer stays on. All settings are lost once power is removed and on next boot the procedure is repeated over again. This auto configuration on startup is how plug and play is implemented which means the user does not have to manually change settings to add hardware such as modifying DIP switches.

The enhanced PCIe configuration access mechanism utilises a flat memory-mapped structure. The mapping from memory address space to PCI Express Configuration Space address is defined in Table 2.4.

Memory Address	PCI Express Configuration Space
A[(20 + n - 1):20]	Bus Number $1 \leq n \leq 8$
A [19:15]	Device Number
A [14:12]	Function Number
A [11:8]	Extended Register Numbers
A [7:2]	Register Number
A [1:0]	Along with size of access, used to generate Byte Enables

Table 2.4: Enhanced Configuration Address Mapping

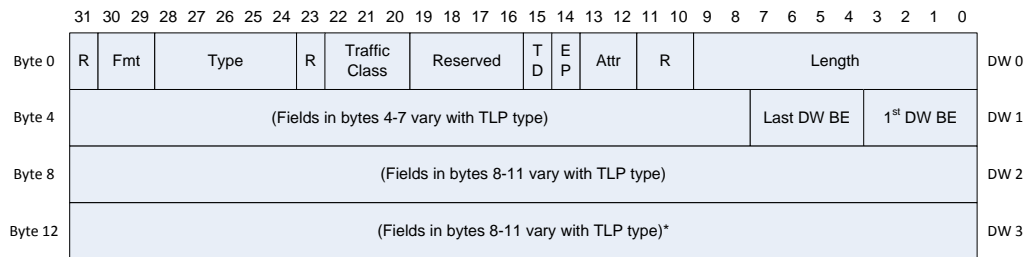


Figure 2.20: The standard Transaction Layer Packet structure

2.6.5 Packet Transactions

As described earlier, PCIe is a packet based technology. The encapsulation follows that shown in Figure 2.12. TLPs and DLLPs follow a standard packet structure where this and the transactions of these packets is detailed below.

2.6.5.1 Transaction Layer Packet

The most common packet is one sourced from the Transaction Layer. The header for such a packet meets the format shown in Figure 2.20. The header can be three or four DWs long. The headers of 32 bit memory packets are 3 DWs long whilst 64 bit memory packets are 4 DWs long.

The 'R' bits are the reserved bits which are set to '0' and do not do anything. 'Fmt' indicates the format of the packet. 'Type' indicates what type of the transaction it is such as a memory write transaction. 'TC' designates

Fmt[1:0] Field	TLP Format
00	3 DW header, no data
01	4 DW header, no data
10	3 DW header, with data
11	4 DW header, with data

Table 2.5: Fmt Field Values

the traffic class of the packet. 'TD' describes the presence of a TLP digest and 'EP' indicates whether the TLP is poisoned. 'Attr' (or attributes) allows modification of the default handling of transactions such as the ordering and hardware coherency management.

From reading the 'Fmt' bits, the type of packet is read. This can be either a 3 DW header with no data ("00"), a 4 DW header with no data ("01"), a 3 DW header with data ("10") or a 4 DW header with data ("11"). If for example a memory read transaction is occurring, no payload will be appended as it will be expecting data in return. So the 'fmt' bits will be either "x0" or "x1" depending if 32 or 64 bit addressing is used.

The Fmt[1:0] field gives the size of the header and whether there is a data payload or not. The list of Fmt[1:0] codes is given in Table 2.5. The full list of Fmt and Type codes and their corresponding packet types are shown in Table 2.6.

Transactions are known as either "posted" or "non-posted". A posted transaction does not require a response. For example, a Memory Write request is considered a posted transaction as the data gets sent to a particular location, and no response is required to complete the transaction. A non-posted transaction is one which requires a response. For example, a Memory Read transaction is classified as a non-posted transaction as it starts a transaction and requires a completion packet to complete this transaction.

Although Memory Write requests do not require completion packets, Configuration Writes do require a completion packet. This is simply a

TLP Type	Fmt[1:0] Field	Type[4:0] Field
Memory Read Request (MRd)	00 01	0 0000
Memory Read Request-Locked (MRdLk)	00 01	0 0001
Memory Write Request (MWr)	10 11	0 0000
I/O Read Request (IORd)	00	0 0010
I/O Write Request (IOWr)	10	0 0010
Configuration Read Type 0 (CfgRd0)	00	0 0100
Configuration Write Type 0 (CfgWr0)	10	0 0100
Configuration Read Type 1 (CfgRd1)	00	0 0101
Configuration Write Type 1 (CfgWr1)	10	0 0101
Message Request (Msg)	01	1 0rrr
Message Request with Data (MsgD)	11	1 0rrr
Completion without Data (Cpl)	00	0 1010
Completion with Data (CplD)	10	0 1010
Completion for Locked Memory Read without Data (CplLk)	00	0 1011
Completion for Locked Memory Read (CplDLk)	10	0 1011

Table 2.6: Description of TLP Fmt and Type codes

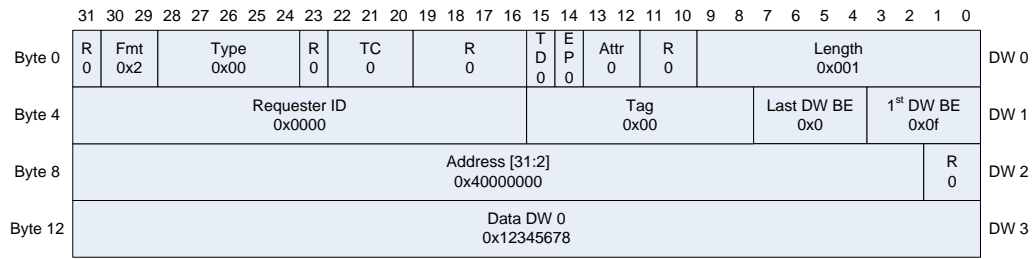


Figure 2.21: Memory Write Packet

completion with no data to confirm that it successfully completed as this is paramount to the system working successfully. For example, when programming the BARs, confirmation is needed that it has been written successfully before a read is signalled.

An example of a posted Memory Write packet is shown in Figure 2.21. As can be seen, the fmt and type fields correspond to a 32 bit memory write request. This packet is of length 1, so contains one DW of data. Here, the address to be sent to is 0x10000000. However, the lower two bits of the address are reserved. Hence this is equivalent to a 30 bit hex value of 0x40000000. DW4 of this packet is the data to be written to the target. No completion is required to be returned for this transaction. The data of PCIe packets is Big Endian. However, x86 processors are little endian. Thus data of 0x12345678 is interpreted as 0x78563412 on an x86 processor.

An example of a non-posted Memory Read packet is shown in Figure 2.22. As can be seen, the fmt and type fields correspond to a memory read request. The length of the packet is 1 so a completion with 1DW of data is expected. Again the lower two address bits are reserved so the address gets multiplied by four to get the actual address. This packet then gets passed along to the target. The target then retrieves the data and sends a completion packet back with the desired data. This packet is shown in Figure 2.23.

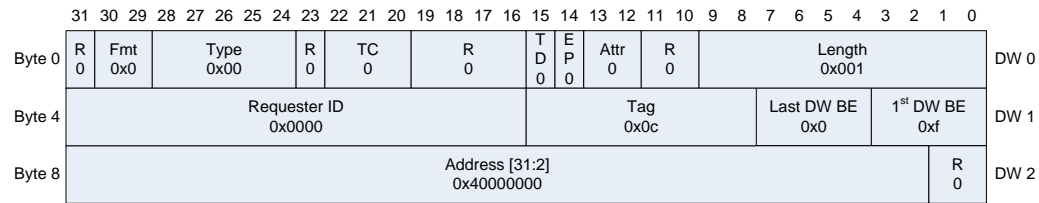


Figure 2.22: Memory Read Request Packet

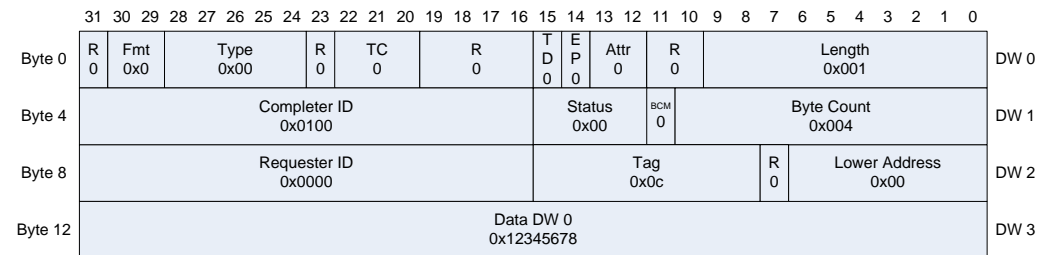


Figure 2.23: Memory Read Completion Packet

2.6.5.2 Data Link Layer Packet

Memory, IO, messages, configuration and completion packets are formed in the Transaction Layer. However there are also packets formed in the Data Link Layer which deal with the upkeep of the link. The standard format of a Data Link Layer sourced packet is shown in Figure 2.24. The Data Link Layer packets comprise of four different types: positive acknowledgement (Ack), negative acknowledgement (Nak), flow control (FC) and power management (PM). The sequence number adds 2 bytes of overhead where 12 bits comprises the actual sequence number and four bits are reserved. The DLLP Type field indicates whether it is an Ack, Nak, Power Management or Flow Control packet. These are shown in Table 2.7.

Ack or Nak packets are initiated to let the sending device know that a Memory Write or Memory Read request was successful or not. If it was not successful, the packet gets resent.

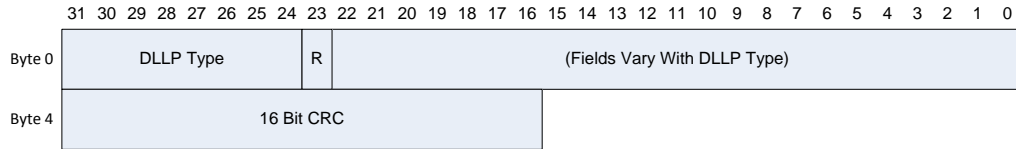


Figure 2.24: The standard Data Link Layer Packet structure

Encodings	DLLP Type
0000 0000	Ack
0001 0000	Nak
0010 0000	PM_Enter_L1
0010 0001	PM_Enter_L23
0010 0011	PM_Active.State.Request_L1
0010 0100	PM_Request_Ack
0011 0000	Vendor Specific - Not used in normal operation
0100 0v ₂ v ₁ v ₀	InitFC1-P (v[2:0] specifies Virtual Channel)
0101 0v ₂ v ₁ v ₀	InitFC1-NP
0110 0v ₂ v ₁ v ₀	InitFC1-Cpl
1100 0v ₂ v ₁ v ₀	InitFC2-P
1101 0v ₂ v ₁ v ₀	InitFC2-NP
1110 0v ₂ v ₁ v ₀	InitFC2-Cpl
1000 0v ₂ v ₁ v ₀	UpdateFC-P
1001 0v ₂ v ₁ v ₀	UpdateFC-NP
1010 0v ₂ v ₁ v ₀	UpdateFC-Cpl
All other encodings	Reserved

Table 2.7: Description of DLLP Type codes

2.7 FPGA

Many digital designs are moving from using dedicated digital logic to Field Programmable Gate Array (FPGA) based systems. FPGAs are a flexible alternative to an application-specific integrated circuit (ASIC). A typical FPGA contains a number of *logic blocks* and reconfigurable interconnects wiring the logic blocks together. When combined, these logic blocks can perform complex tasks or be as simple as a basic logic gate. FPGAs typically contain other elements such as memory. Almost all of the peripheral devices designed by Magritek make use of an FPGA to control the system. The PXIe Peripheral Module was also designed using an FPGA as a central element.

FPGA devices provide a lot of flexibility to the designer and allow for rapid prototyping of digital designs. FPGAs are traditionally configured using a Hardware Description Language (HDL) which describes the circuits operation and organisation. HDLs can furthermore be used to verify operation via simulation. The two main forms of HDL are Very-High-Speed Integrated Circuit HDL (VHDL) and Verilog. VHDL, based off ADA, is verbose and employs strong typing, whereas Verilog is similar to C in syntax but employs weak typing [21].

From the HDL files the compiler will map the operations on to the available resources of the selected device in a form known as a netlist. The netlist is then used by the compiler to produce a configuration file for the FPGA. For instance, say the following description were written in VHDL.

```
flip_flop : process(clk, rst) begin
  if (rst = '1') then
    Q <= '0';
  elsif rising_edge(clk) then
    Q <= D;
  end if;
end process;
```

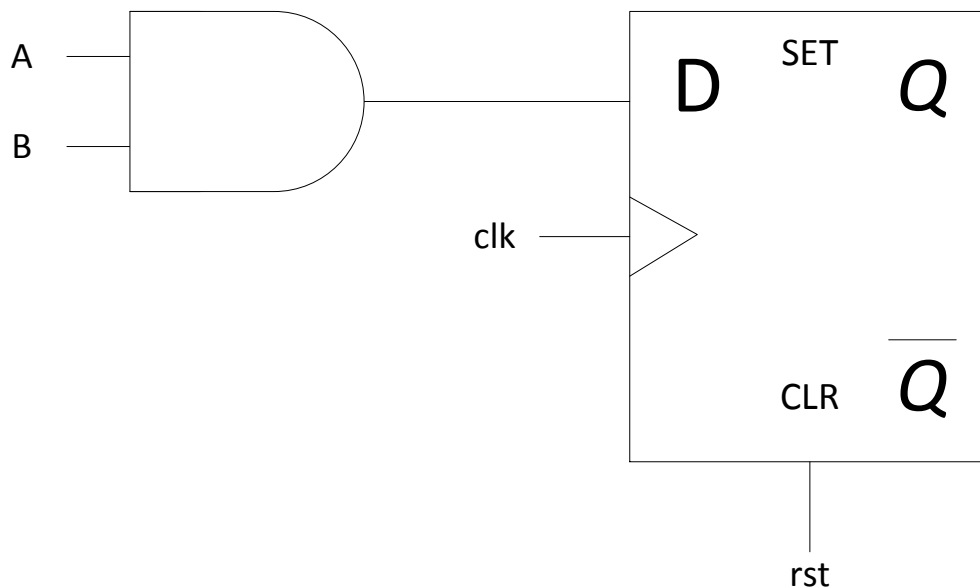


Figure 2.25: VHDL Synthesis to AND gate and D Flip Flop

The compiler will see this description and recognise it as a D flip flop and implement it as so. Likewise, if the following description were written.

```
test_circuit : process(clk, rst) begin
  if (rst = '1') then
    Q <= '0';
  elsif rising_edge(clk) then
    Q <= A and B;
  end if;
end process;
```

The compiler will notice that the description consists of a synchronous process (D flip flop) and combinational logic to set the input value (A AND B). This will then be synthesised as shown in Figure 2.25.

Xilinx's development tools recognise arithmetic operations and utilise the on board multipliers in synthesis. For example:

```
multiply : process(A, B) begin  
  Q <= A * B;  
end process;
```

The synthesiser will process `'*'` symbol and interpret this as a multiplication operation. It will then route the appropriate signals (A and B) to the inputs of a multiplier block on the FPGA.

These design examples are easy to synthesise but do not exhibit the true potential of FPGAs. Where FPGAs really become advantageous is in complex designs. Such designs would be difficult to construct from discrete components due to the high IC count. Such designs traditionally utilised software on CPUs or DSPs due to the complexity. However FPGAs provide the ability to produce designs that would be more efficiently performed in hardware, without the difficulty of connecting a large number of ICs together.

Most FPGA vendors also allow typical features to be implemented using a drag-and-drop method. So rather than writing each individual digital block in HDL, if one wanted a VGA controller for example, this could just be dragged into the design with the inputs and outputs declared. These are known as IP Blocks. Very complex features may be added this way. Having access to such tools greatly increases the speed of design. Blocks can either be added as soft IP or hard IP. Soft IP is where there is no dedicated hardware for the function required but the general logic on board is configured to provide the function. Hard IP makes use of hardware on the FPGA which is dedicated for that particular application which can provide performance benefits, but can also be wasteful if that function is not required.

Designs are often simulated first before being programmed to the physical FPGA. Simulation is provided using a *testbench* which the FPGA design communicates with. The testbench provides any required inputs and reads output signals in the design. The simulation software performs the description provided by the HDL. This lets the designer verify the HDL

and check the internal signals before configuring the FPGA.

The two major FPGA vendors are Xilinx and Altera. Both produce similar products but each have their own bundled design suites. Xilinx also includes its own ISim simulation software whilst Altera bundles 3rd party simulation software with their product. The choice of FPGA is normally down to preference, with Xilinx chosen for this design as Magritek have an existing relationship with the company and their tools and software are widely used within the business.

Using an HDL design to configure an FPGA requires a number of steps. The process for Xilinx devices includes:

- Writing an HDL design (normally in VHDL or Verilog).
- Synthesising the design which comprises converting the written project to an actual design.
- (Optional) Simulating the design in software to test the device works as expected.
- Mapping the hardware design to the actual hardware on the FPGA.
- Producing a file ready to program to the FPGA where the I/O pins of the FPGA are also specified.
- The design is programmed to the FPGA, typically using the Joint Test Action Group (JTAG) protocol.

Xilinx compiles projects in a number of steps. It first synthesises the design which consists of mapping all the descriptions written in software to hardware components which the FPGA utilises. Once this is complete, it finds the optimum way to place the design on the hardware given. For Xilinx devices this is usually done in their provided ISE design suite.

2.7.1 VHDL

As previously mentioned, there are a number of HDLs that can be used to describe how the FPGA can be configured. The two main languages used are VHDL and Verilog. While both languages are almost equally popular for FPGA development, there is a trend for Verilog to be more favoured in industry and the USA, whereas VHDL is preferred more commonly in academia and outside of the USA.

The verboseness of VHDL adds to the rigour for its use in the design of digital circuits. Project components must be described completely with the signal type and width of each input and output defined. For example, signal assignments must be made with the exact correct signal length else it will cause an error. In this way, there are more safeguards built into the design than in Verilog.

2.8 High speed signal board design

The high speed signals used for this PXIe required particular care to be taken in the routing of the signals. For such signals, meeting a characteristic impedance is usually required. Also, differential signals require the pairs that make up each link to be length matched. These techniques are described in detail below.

2.8.1 Characteristic Impedance

With high frequency signals, reflections on the line become important. Thus the size of these reflections need to be minimised. This is normally accomplished by making the tracks the signals travel along meet a characteristic impedance. If the characteristics of the wire change, reflections will occur along the line which effects the signal integrity and can result in power losses. These power losses can result in signals being detected incorrectly.

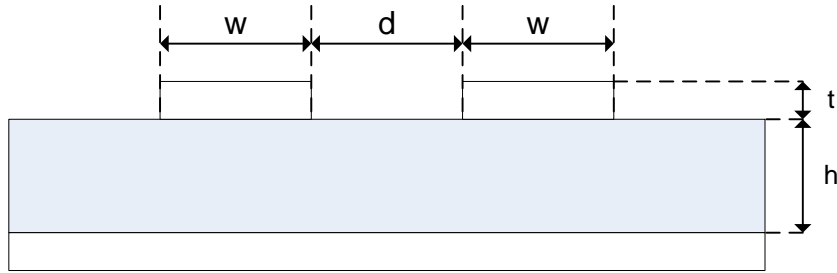


Figure 2.26: Parameters involved in microstrip calculation

The characteristic impedance is the ratio of the amplitudes of a single pair of voltage and current waves propagating along the line in the absence of reflections. It is calculated based on aspects of the track routing such as track width, copper thickness and track separation. If the tracks are designed to all meet this impedance, this keeps the losses to a minimum.

In order to meet the characteristic impedance for a PCB, it can be modelled as either a microstrip case or a stripline case. The microstrip case is used when the track is on an external layer of the PCB which is shown in Figure 3.31. The stripline case is used when the track is routed as an internal layer of a PCB which is shown in Figure 2.27. Each case has its own equation for calculating the impedance of the line. The single ended impedance calculation for the microstrip case is given in Equation 2.1. For differential signals, the impedance calculation for a microstrip is shown in Equation 2.2. For the stripline case, the single ended impedance calculation is shown in Equation 2.3. The differential impedance of the stripline case is shown in Equation 2.4.

$$Z_0 = \frac{87}{\sqrt{\epsilon_r + 1.41}} \ln\left(\frac{5.98h}{0.8w + t}\right) \quad (2.1)$$

$$Z_d = \frac{174}{\sqrt{\epsilon_r + 1.41}} \ln\left(\frac{5.98h}{0.8w + t}\right) \left(1 - 0.48 \exp\left(-0.96 \frac{d}{h}\right)\right) \quad (2.2)$$

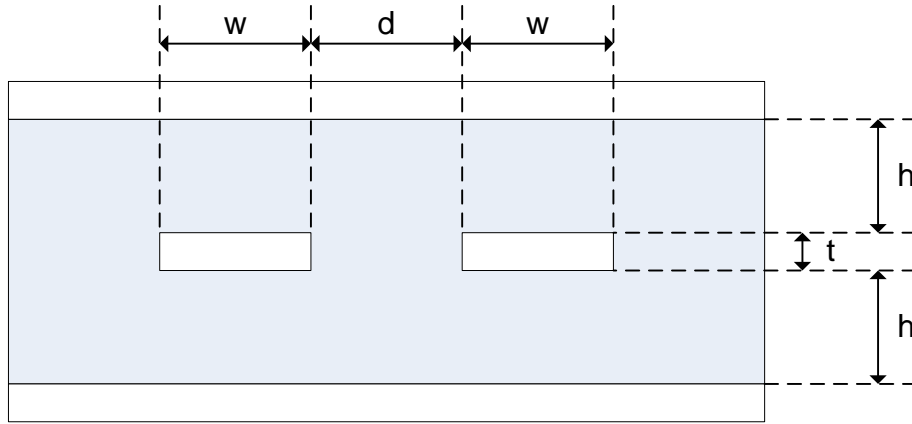


Figure 2.27: Parameters involved in stripline calculation

$$Z_0 = \frac{60}{\sqrt{\epsilon_r}} \ln\left(\frac{1.9(2h + t)}{0.8w + t}\right) \quad (2.3)$$

$$Z_d = \frac{120}{\sqrt{\epsilon_r}} \ln\left(\frac{1.9(2h + t)}{0.8w + t}\right) \left(1 - 0.347 \exp\left(-2.9 \frac{d}{2h + t}\right)\right) \quad (2.4)$$

2.8.2 Length Matching

The lines that make up a differential signal are opposite in polarity at any given time. For differential signalling to function correctly, pairs that make up a link need to be routed to the same length. This is done so that the two components of the signal reach the destination at the same time. If not, the differential signal could be received incorrectly.

2.9 Linux Systems

One of the ambitions for the design of PXI Modules was to provide an open system where the design work could be widely used over a number of products. To provide this, the modules were designed for use in the

Linux environment. Linux Device Drivers and Applications are usually provided in open source which made any modifications a lot easier.

A Linux system is built around a single monolithic kernel which configures a base system. On top of this, the user environment, window manager, graphical user interface (GUI) and applications are then added to provide greater functionality to the user.

As Linux is an Open Source environment, device drivers are commonly provided with source code. This makes developing and modifying designs a lot easier. Linux can build device drivers as a part of the kernel or separately as loadable modules. Commonly required drivers such as those for disk drives, sound etc are provided with the kernel. As well as these, loadable modules are added to allow additional peripherals to communicate. Device drivers are typically written as a C source file. These files then get compiled into a .ko kernel module where the linux command *insmod* is then called to insert the module into the kernel.

Chapter 3

Design and Implementation

This chapter will discuss the PXIe design involved in the project. This includes the design of the PXIe Peripheral Module, the FPGA design used for the PXIe module and the design of the backplanes used for testing PCIe and PXIe designs.

3.1 Description of project

Magritek's present modular NMR systems make use of a custom backplane which uses a parallel transmission system. This provides around 8 MB/s bandwidth which gives adequate performance for many NMR systems. However, the bandwidth requirements are insufficient for more demanding applications such as multi channel MRI. This was a major motivation for the move to a PXIe system.

The project consisted of providing a PXIe Peripheral Module and System Timing Module for Magritek's use. An FPGA design was carried out to allow PCIe data transfers. Also, test environments and documentation were provided so that PXIe systems could be further developed in the future. The design work for providing a PXIe would require:

- Development of a PXIe Peripheral Module and System Timing Mod-

ule in collaboration with Magritek. The schematics, connectors and board details were designed and provided for Magritek's use. This involved research into the signals required and their particular attributes. This work may then be used in the development of further Endpoint modules.

- Configuration of the FPGA and software to perform configuration and memory read and writes between devices. This consisted of evaluating Xilinx PCIe solutions and adapting them to allow Memory Read and Writes.
- Design of backplanes used to test PCIe and PXIe devices. These would be passive backplanes that would provide basic communication between a host and Endpoint. The communication between PCIe and PXIe devices could be tested.

In the next section the design of the PXIe Peripheral Module is detailed. The board, signal and connector requirements were researched to allow the board designer to produce a PXIe Peripheral Module.

3.2 PXI Express Peripheral Module

The design of the PXIe Peripheral Module required:

- Research into PXIe signals their corresponding voltage level, length matching and impedance requirements.
- Board design descriptions and sourcing of connectors and components required for PXIe transmission.
- Creation of schematic documents, schematic libraries and footprints for PXIe system.

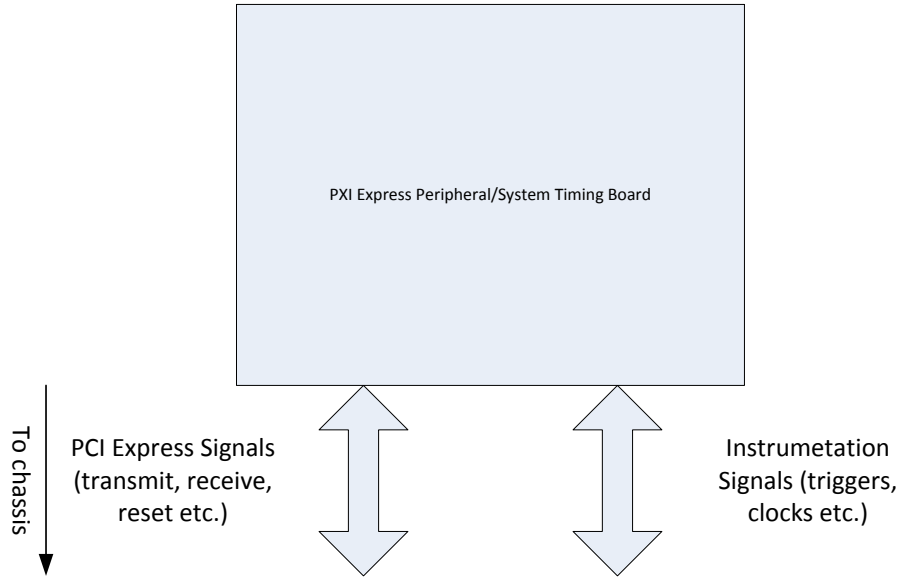


Figure 3.1: How the PCI Express serial transmission signals were separated from the instrumentation signals in PXI Express

To simplify the design, an abstraction between the PCIe functionality and the instrumentation signals was made. A diagram describing how the two sets of signals were abstracted is shown in Figure 3.1. This abstraction meant that the existing tools available for PCIe designs could be used with the additional PXIe functionality added to this.

As described earlier, FPGAs provide a lot of flexibility and fast prototyping. For the PXIe Peripheral Module, the Xilinx Spartan-6 XC6SLX45T was used. A big advantage of this FPGA is that a hard IP block for PCIe could be used which automatically configures the basic PCIe system [22].

The signals sourced from the backplane connectors were routed to the pins of the FPGA. The Spartan-6 XC6SLX45T provided 296 I/O pins [23]. The pins on this device were separated into four different voltage banks. This meant that at most, four different I/O voltage levels could be used.

Thus the voltages for each of the PXIe signals had to be researched and compiled so they would be connected to the correct bank. If there were any differing voltage levels, these would have to be converted to a supported voltage first.

Xilinx produces a Spartan-6 Connectivity Kit Development Board [24] which it also provides schematics for [25]. This development board is also based on the XC6SLX45T Spartan-6 FPGA. The board includes a x1 PCIe connector which can be connected to a PCIe slot. This provided a useful reference design when routing the PCIe signals for the designed PXIe Peripheral Module. The SP605 board was also used for testing of the FPGA design.

3.2.1 Mechanical requirements for board design

A PXIe Peripheral Module meets the 3U eurocard form factor as described in [11]. The dimensions given were 100 mm by 160 mm with a board thickness of 1.6 ± 0.2 mm which the designed board was required to meet. The high complexity of the design meant a multi-layer board was designed. Thus the board thickness requirements had to be factored into the board stack-up. The board requirements are shown in Figure 3.2. The board had to be laid out to meet this size with the connectors in the positions shown.

Connections between the PXIe chassis and plug-in boards are made using specific connectors. As the form factor was carried over from cPCIe, these were defined in this specification [18]. Connectors featured on boards included:

- Advanced Differential Fabric (ADF) Connector (ADF-F-3-10-2-F-25)
 - A 90 pin connector for the use of transmitting differential signals. Each differential pair on the connector is shielded with a ground signal. These come as right angle female connectors for use on boards and vertical male connectors for use on backplanes.

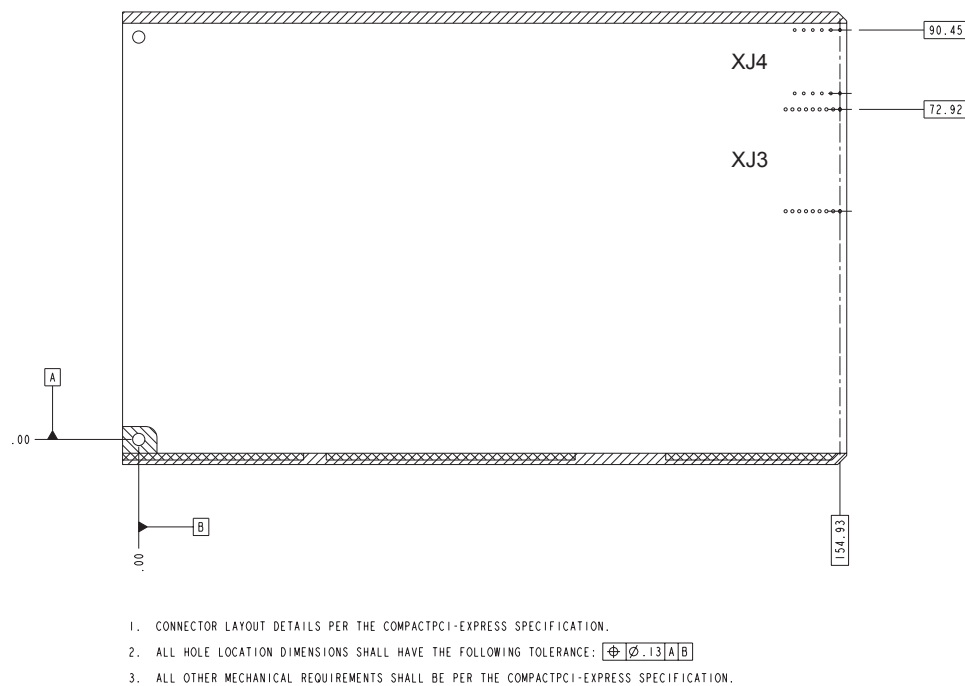


Figure 3.2: Board layout of PXI Express Peripheral Module [11]

- **Enriched Hard-Metric (eHM) Connector (eHM-F2)** - A 60 pin connector which carries power and non-differential (usually side-band) signals. These come as right angle female connector for use on boards and vertical male connectors for use on backplanes.
- **Universal Power Connector (UPM) (UPM-F-7)** - A 7 pin connector used for Host Modules which require more power. These come as right angle male connectors for use on boards and vertical female connectors for use on backplanes.

A PXIe Peripheral Module includes two connectors, a right angle female XJ4 EHM connector and a right angle female XJ3 ADF connector [11]. The connectors needed to be sourced from a reputable vendor. Connectors from ERNI electronics were used as these were verified to meet the PXIe standard. These connectors are shown in Figure 3.3.

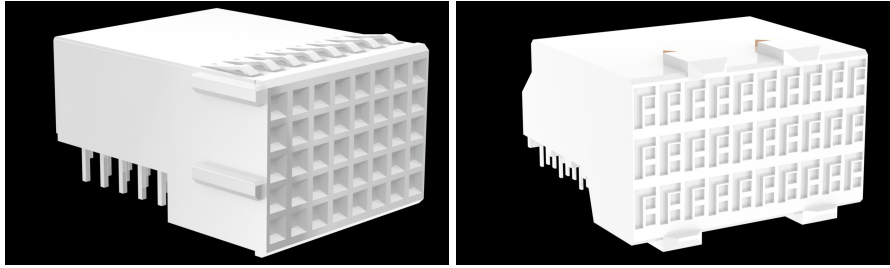


Figure 3.3: Connectors used for PXI Express Peripheral Module

3.2.2 PXI Express Power

The PXIe standard provided 3.3V, 5V, 12V and an auxiliary 5V (5Vaux) power line on the backplane. This information was passed onto the board designer so that if different voltage levels were required for the design, these were to be stepped up or stepped down from these levels.

3.2.3 PCI Express Signals

The core components of the serial transmission scheme employed by PCIe and PXIe devices are the differential data signals (transmit and receive) and the differential clock signal. The clock is used by the devices to produce its serial transmission scheme. The device was designed to meet revision 1.1 of the PCIe specification [8]. This provided a maximum bandwidth of 250MB/s where bits are clocked out at 2.5 Gb/s which is generated by multiplying the incoming 100 MHz by way of a PLL. These high speed differential signals require care to be taken when routing. Thus information on the characteristic impedance and length matching was investigated. PCIe also makes use of a number of sideband signals. The various signals used for PCIe transfers are detailed below.

3.2.3.1 PCI Express Transmit and Receive signals

The PCIe Transmit and Receive signals are differential serial connections which are connected across the backplane with $100\ \Omega$ balanced transmission lines. Each transmission pair was AC coupled to remove the DC component. This was done using external 100 nF decoupling capacitors at the output of the transmit pair on the FPGA. Being differential signals these were also required to be length matched. A $100\ \Omega$ differential termination was required which was provided by the FPGA so no external resistors were required.

3.2.3.2 Clock Signal

With PXIe systems, a 100 MHz clock is derived from the System Controller Card to each of the peripheral modules. This is a feature carried over from cPCIe where on standard PCIe designs, the clock is usually provided from the backplane or motherboard. The clock signal also required decoupling capacitors. Recommended decoupling capacitors of 100 nF were suggested, but errors in the board design meant 10 nF capacitors were used to decouple the signal. This still provided the decoupling required however. Termination of $100\ \Omega$ was required and provided internally by the FPGA.

3.2.3.3 Sideband Signals

Along with the differential transmit and receive pairs and the clock signal, the PCIe standard also uses a number of sideband signals. PXIe has sideband signals that vary subtly from the PCIe standard. The PXIe sideband signals include:

- SMBus - This includes SMBDAT and SMBCLK signals. SMBus is a two-wire interface for power and system management tasks. They were not required signals but were decided to be connected for possible future use.

- PERST# - The PCIe Reset signal. When the signal is high, all PCIe functions are held in reset. The signal is de-asserted 100 ms after the clock signal and voltage levels have reached stability. This is activated upon power up of the device until stability is reached and can also be triggered as a means of getting the device to reset when the clock signal or voltage levels fall outside of safe limits.
- PRSNT# - This detects the board presence. It is connected to ground on board so that by way of a pull-up resistor, can detect the presence of board being inserted.
- Hot-Plug - This includes the signals: MPWRGD, ALERT#, ATNLED, ATNSW#. Hot-Plug, as the name suggests, allows "hot" removal and insertion of cards to the system (removal or insertion whilst the system is still powered on). The board being designed was still in the prototyping stage and such functionality would not be required. Magritek's NMR units do not require cards to be swapped out often and the devices are usually powered off before replacing boards. Thus the signals MPWRGD, ALERT#, ATNLED, ATNSW# were not routed on the board.
- WAKE# - This can be used to wake up the system when in a low-power state, making use of the 5Vaux power. Again it was decided this functionality was not required in the prototyping. Thus the WAKE# signal was not routed on board.

3.2.3.4 PCIe signals used for Peripheral Module

The PCIe signals used included the transmit (1PET0) and receive (1PER0) differential pairs, the differential clock (1REFCLK), PERST#, SMBCLK and SMBDAT. The details of these signals were researched and compiled in Table 3.1. These signal requirements were followed when the board was routed.

Signal Name	Description	Type	Voltage	Frequency	Impedance	Termination
1PER0	Receive signal	Differential pair	0.175- 1.2 V	2.5 GHz	50 Ω Single Ended 100 Ω Differential	100 Ω (provided on FPGA)
1PET0	Transmit signal	Differential pair	0.8- 1.2 V	2.5 GHz	50 Ω Single Ended 100 Ω Differential	100 Ω (provided on FPGA) 100 nF decoupling capacitor
1REFCLK	Clock signal	Differential pair	2.1V max	100 MHz	50 Ω Single Ended 100 Ω Differential	100 Ω (provided on FPGA) 100 nF decoupling capacitor
PERST#	Reset signal	Single Ended	3.3 V	NA	NA	None
SMBCLK SMBDAT	SMBus signal and clock pair	Single Ended	3.3 V	NA	NA	None

Table 3.1: Description of PCI Express signals required for PXI Express Peripheral Module

The naming convention for the differential transmit (1PET0) and differential receive (1PER0) signals is different in PCIe to that used for PXIe (or that inherited from CompactPCIe). In PCIe, these are named all relative to the host system. On a PCIe slot or device, the 1PET0 signal is the signal which the host system transmits and the peripheral device receives. Likewise the 1PER0 signal is the signal which the host system receives and the peripheral device transmits. This is reversed on cPCIe and PXIe systems where the signals are all labelled relative to the local unit. Thus on peripheral modules, 1PET0 and 1PER0 signals are its transmit and receive signals respectively.

3.2.4 PXI Express Signals

PXIe provides the instrumentation signals required for the device. From the original PXI specification these include a 10 MHz clock (PXI_CLK10), eight trigger bus lines (PXI_TRIG[0:7]), a daisy chained trigger line (PXI_LBL6 and PXI_LBR6) and star trigger lines from the system timing slot to each of the other slots on board (PXI_STAR). These signals are all single ended signals.

With PXIe, the signals provided by PXI were retained with additional differential signals added. The additional signals consisted of a 100 MHz differential clock signal (PXIe_CLK100), a differential synchronisation signal (PXIe_SYNC100) and three length matched differential star signal lines from the system timing slot to the other slots on board (PXIe_DSTAR_A/B/C).

The star lines allow signals to be transmitted between the system timing slot and the other peripheral devices. These include the PXIe_DSTARA and PXIe_DSTARB slots which allow the system timing slot to provide clock signals and trigger signals respectively. PXIe_DSTARC is used for signals in the other direction where peripheral modules can transmit clock or trigger signals back to the system timing slot. All these signals provide greater performance than those of the original PXI specification as they

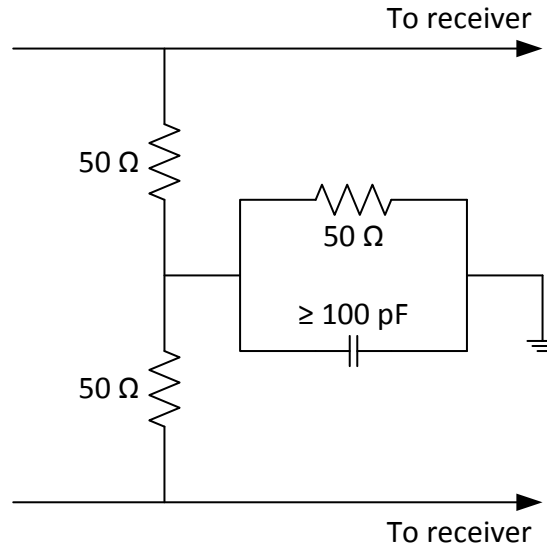


Figure 3.4: Termination circuit for incoming differential PXI Express clock signals

moved from single ended to differential signalling.

PXIe also allows slot identification done by using Geographical Addressing (GA) pins. The slot number is encoded using these five pins. For instance, physical slot one has GA[4:1] connected to ground and GA[0] unconnected. Likewise, physical slot three would have GA[4:2] connected to ground and GA[1:0] unconnected. These get read by connecting to the FPGA through pull up resistors. Unconnected GA pins get read as high and connected pins get read as low.

All the instrumentation signals provided were made use of and routed to the FPGA. For successful routing the details of each signal were researched and compiled. These are shown in Table 3.2. The 50 Ω to 1.3V termination circuit is shown in Figure 3.4.

As well as these constraints, the PXIe specification states that all the terminations shall not be made more than 1ns of electrical length beyond the backplane connector [11]. Taking the speed of electrical signals as the

Signal Name	Description	Type	Voltage	Frequency	Characteristic Impedance	Termination
PXLCLK10	10 MHz clock	Single Ended	3.3 V TTL	10 MHz	65 Ω	None
PXLSTAR	Star trigger	Single Ended	User Defined	NA	NA	None
PXL1BL6 PXL1BR6	Daisy-chained local bus	Single Ended	User Defined	NA	NA	None
PXLTRIG[0:7]	Trigger bus	Single Ended	3.3 V LVTTTL	NA	NA	Optional pull up with 20 K Ω
PXLCLK100	100 MHz differential clock	Differential pair	3.3 V LVPECL	100 MHz	100 Ω Differential 50 Ω Single Ended	50 Ω to 1.3 V
PXLSYNCL100	Synchronisation signal	Differential pair	3.3 V LVPECL	NA	100 Ω Differential 50 Ω Single Ended	50 Ω to 1.3 V
PXLDESTARA	Differential clock	Differential pair	3.3 V LVPECL	User Defined	100 Ω Differential 50 Ω Single Ended	50 Ω to 1.3 V
PXLDESTARB	Differential trigger	Differential pair	3.3 V LVDS	NA	100 Ω Differential 50 Ω Single Ended	100 Ω
PXLDESTARC	Differential clock or trigger	Differential pair	3.3 V LVDS	User Defined	100 Ω Differential 50 Ω Single Ended	None

Table 3.2: Description of instrumentation signals for Peripheral Modules

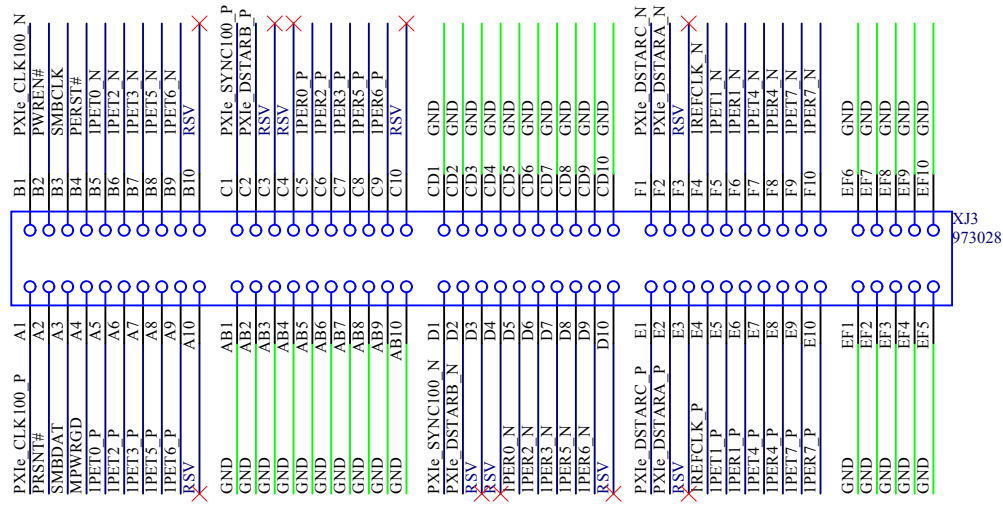


Figure 3.5: XJ3 Schematic for PXI Express Peripheral Module

speed of light this gives:

$$\text{Length} = \text{Speed of signal} \times \text{Time of signal} = 0.3 \text{ m}$$

Thus the signal must be terminated within 30 cm of the connector.

3.2.5 Creation of Libraries, Schematics and Footprints for Device

To implement the PXIe connectors on the board being made, schematics libraries and footprints for the XP3 and XP4 connectors were designed. The pin outs for these connectors were found in the PXIe specification [11].

The default templates were available on AltiumLive, an online resource where templates, component footprints etc are available to Altium users [26]. These templates gave a base design for a PXIe module including schematic files for the XP3 and XP4 connectors. The XJ3 and XJ4 schematics for the PXIe Peripheral module are shown in Figures 3.5 and 3.6.

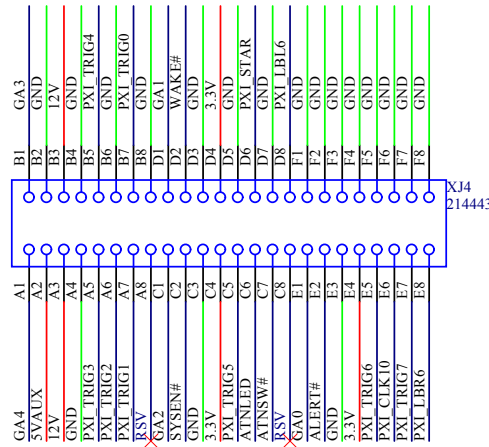


Figure 3.6: XJ4 Schematic for PXI Express Peripheral Module

The template also provided a blank PCB meeting the correct board size with footprints for XJ3 and XJ4 connectors positioned correctly.

Using these default templates, the signals from the PXIe connectors were routed to the correct pins of the FPGA. The PCIe transceiver and clock signals had to be connected to the assigned ports whilst the sideband signals and PXIe signals were routed to the correct bank on the FPGA.

3.2.6 Schematics

The described design was incorporated into a larger design for the PXIe Peripheral Module which included the NMR transceiver functionality. A block diagram of the the full system connected to the FPGA unit is shown in Figure 3.7. Detailed schematics produced for the PXIe system are shown in Figures 3.8 to 3.10. Figure 3.8 shows how some signals sourced from the backplane were terminated or AC coupled. Figure 3.9 shows the differential signals required for PCIe functionality were connected to particular transceiver ports of the FPGA. Figure 3.10 shows the additional PXIe signals connection to the FPGA on the 3.3 V bank.

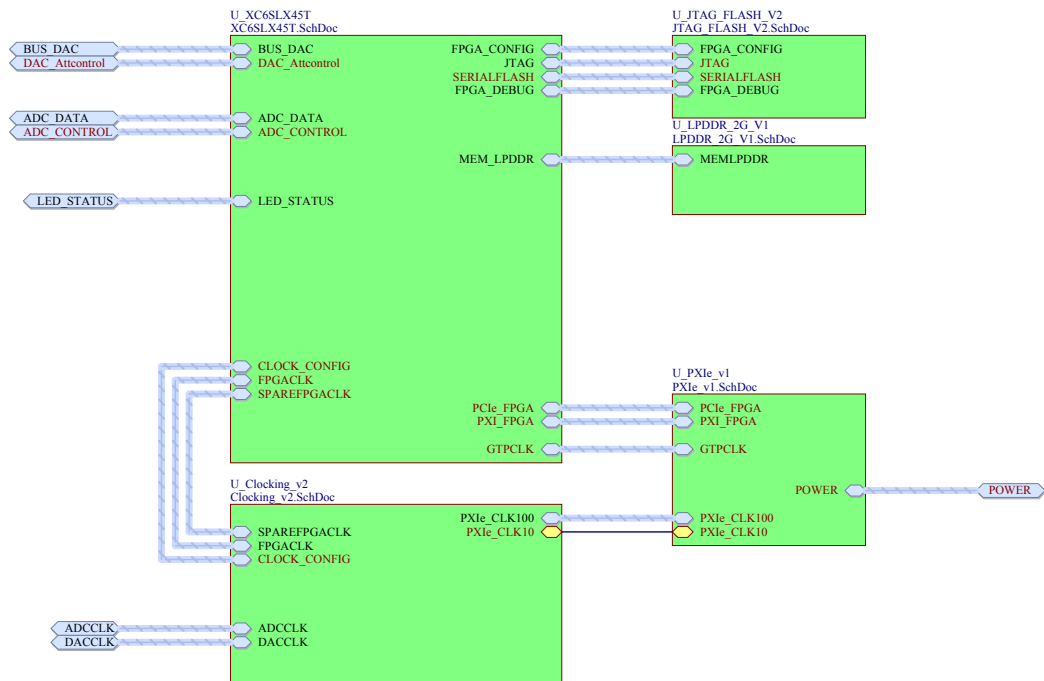


Figure 3.7: Block diagram of schematics for PXI Express Peripheral Module design

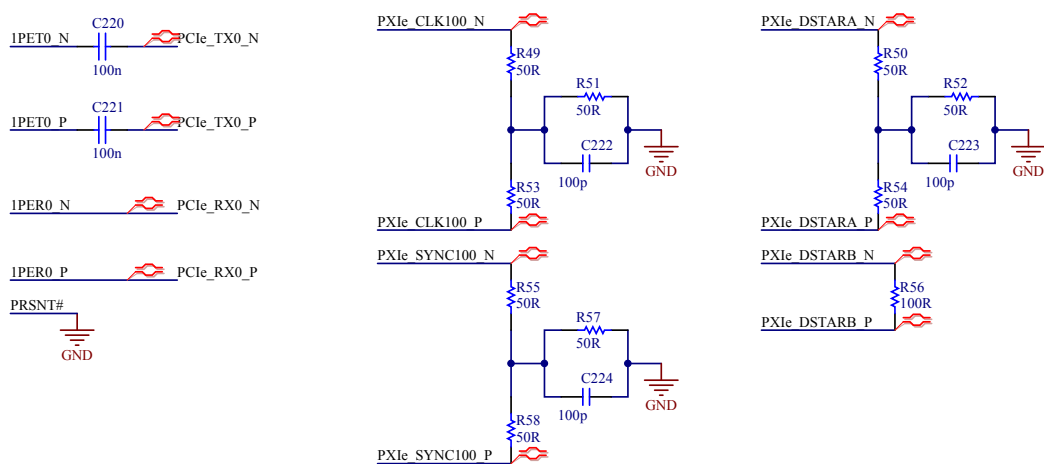


Figure 3.8: Processing of PXI Express signals used on PXI Express module

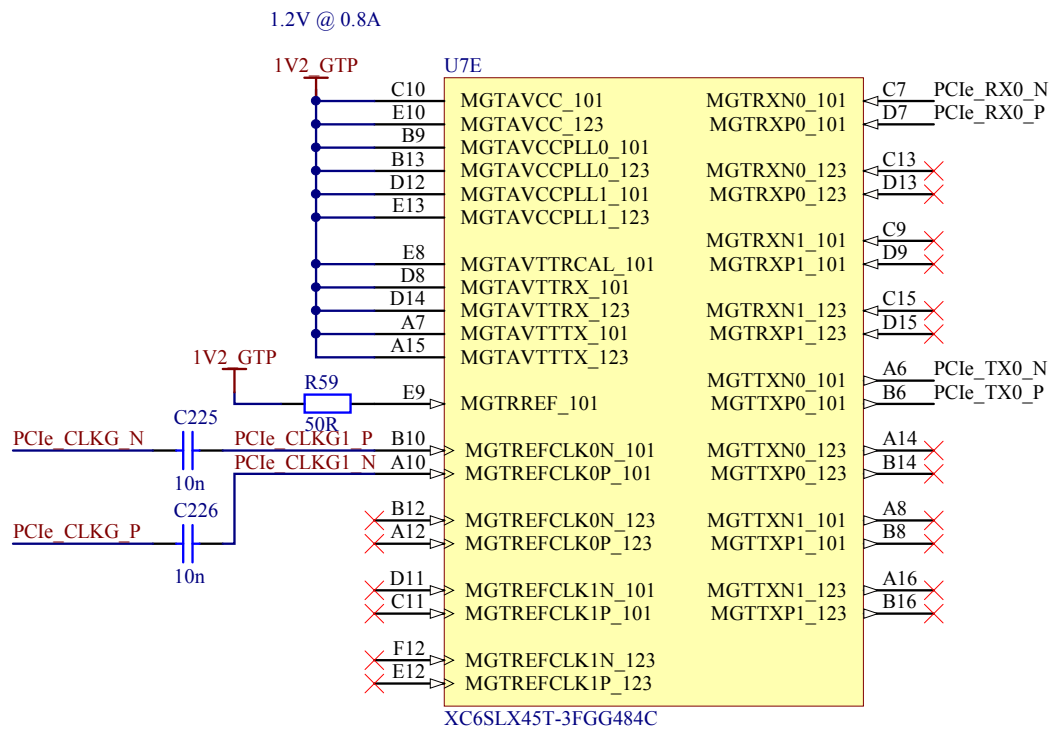


Figure 3.9: Connection of PCI Express signals to FPGA

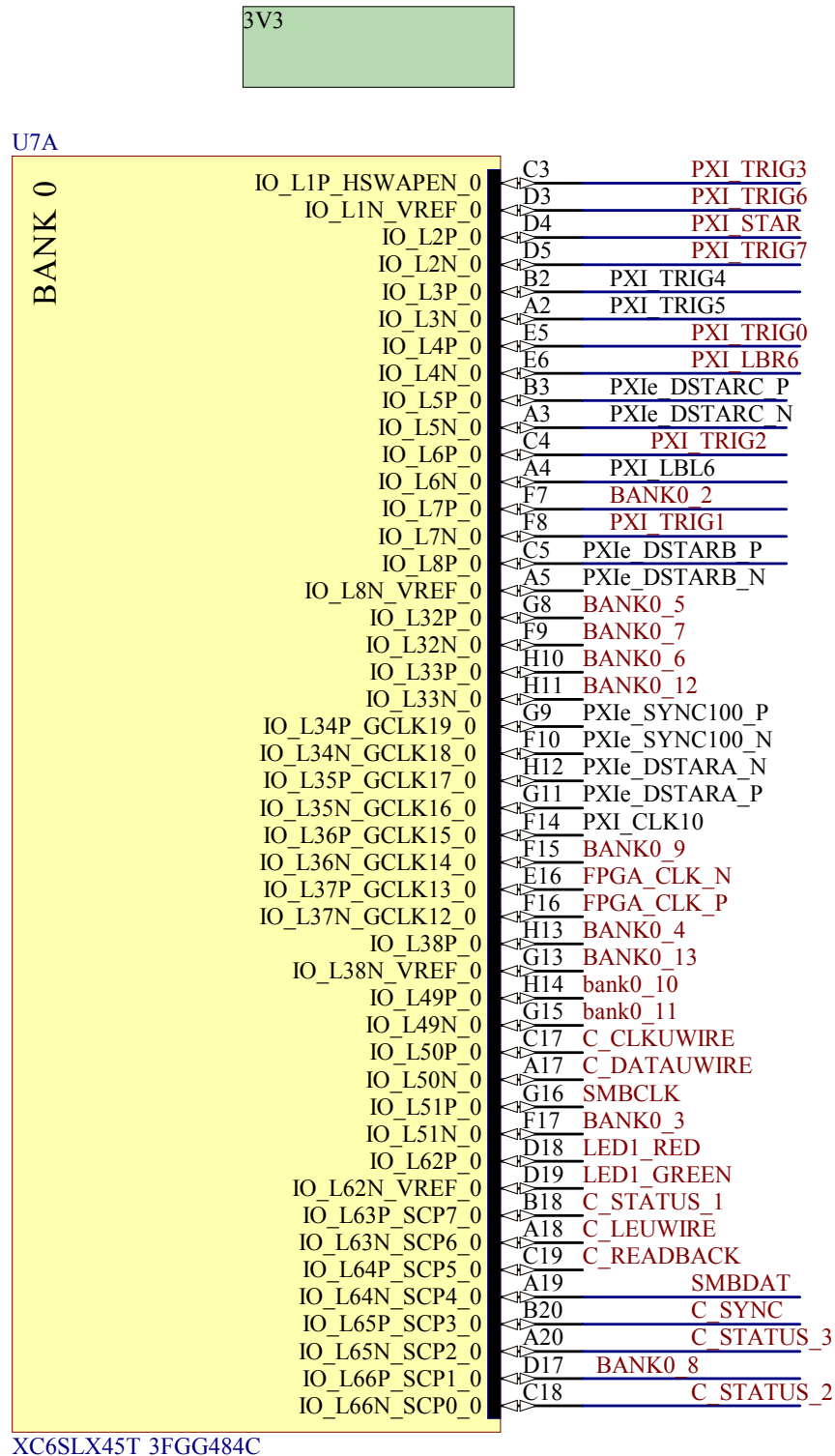


Figure 3.10: Connection of other PXI Express signals to FPGA

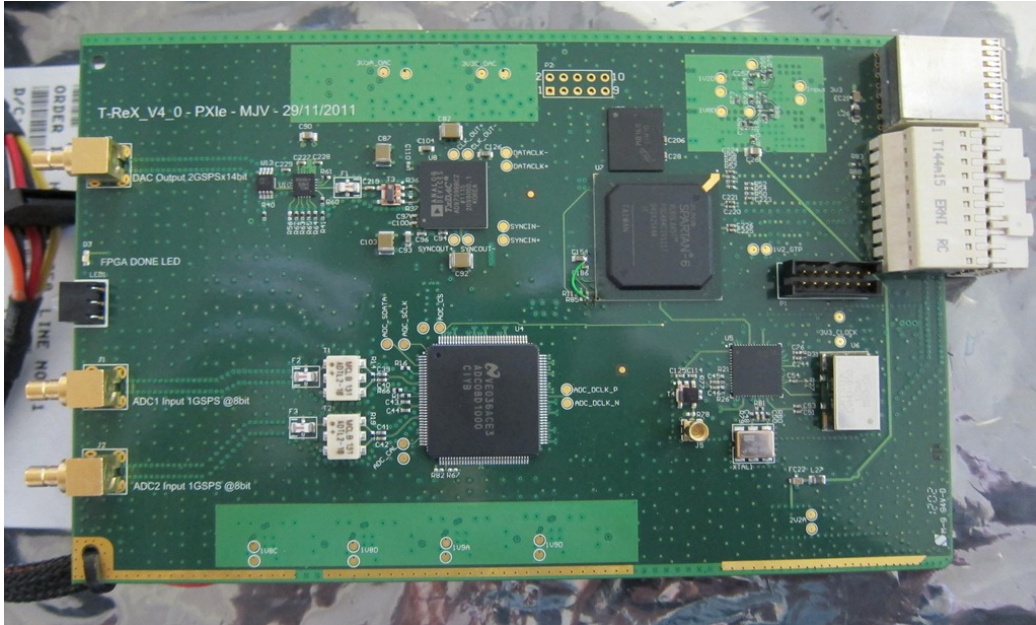


Figure 3.11: Designed PXI Express Peripheral Module - The T-Rex 4

3.2.7 Completed board

These completed schematics along with information on length matching, characteristic impedances and terminations was passed on to the board designer. These were integrated with the rest of the circuit design completed by others at Magritek. This board was sent off to be constructed and populated and the resulting board is shown in Figure 3.11. As can be seen, the board includes the XJ3 and XJ4 connectors and Spartan-6 FPGA.

The designed board came to 10 layers in total because of the large amount of circuitry. The total board size was 1.889 mm so just over the bounds of the recommended board size of 1.6 ± 0.2 mm. The layer stack is shown in Figure 3.12. The differential signals were routed on the inner layers hence the stripline model was used to meet impedance requirements. 0.08 mm track widths with a separation of 0.125 mm were used. Using the microstrip equations Equations 2.3 and 2.4, this gave a single ended impedance of 55Ω and a differential impedance of 101Ω thus meeting the

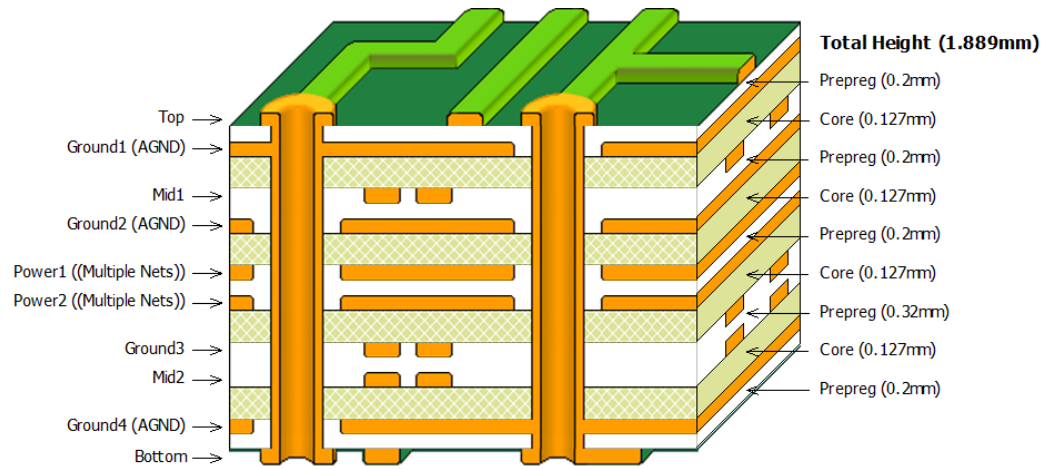


Figure 3.12: Layer stack of designed PXI Express Peripheral Module

requirements.

3.3 Design of System Timing Module

The design of the System Timing Module was similar to that of the Peripheral Module. The PCIe signals involved in the system were identical therefore this stayed the same. However, there were more instrumentation signals as the System Timing Module was where all of the star clock and trigger signals were sourced from. There was also the addition of an extra High-Speed Advanced Differential Fabric connector. This connector is known as a TJ2 connector. For System Timing Modules in chassis with more slots, there is also a TJ1 connector for the addition clock and trigger signals. However, it was not necessary to drive this many slots so only one extra connection was used.

As the direction of the star clock and trigger signals of the System Timing Module was opposite to that of the Peripheral Modules, the termination requirements were different. This is shown in Table 3.3.

At the time of writing, the System Timing Module was still in the design stage where others at Magritek are to complete the rest of the design.

Signal Name	Description	Type	Voltage	Frequency	Characteristic Impedance	Termination
PXLCLK10	10 MHz clock	Single Ended	3.3 V TTL	10 MHz	65 Ω	None
PXLSTAR	Star trigger	Single Ended	User Defined	NA	NA	None
PXL1BL6	Daisy-chained	Single Ended	User Defined	NA	NA	None
PXL1BR6	local bus	Single Ended	3.3 V LVTTTL	NA	NA	Optional pull up with 20 K Ω
PXLTRIG[0:7]	Trigger bus	Single Ended	3.3 V LVTTTL	NA	NA	Optional pull up with 20 K Ω
PXLCLK100	100 MHz differential clock	Differential pair	3.3 V LVPECL	100 MHz	100 Ω Differential 50 Ω Single Ended	50 Ω to 1.3 V
PXLSYNCL100	Synchronisation signal	Differential pair	3.3 V LVPECL	NA	100 Ω Differential 50 Ω Single Ended	50 Ω to 1.3 V
PXLDESTARA	Differential clock	Differential pair	3.3 V LVPECL	User fined	100 Ω Differential 50 Ω Single Ended	None
PXLDESTARB	Differential trigger	Differential pair	3.3 V LVDS	NA	100 Ω Differential 50 Ω Single Ended	None
PXLDESTARC	Differential clock or trigger	Differential pair	3.3 V LVDS	User fined	100 Ω Differential 50 Ω Single Ended	100 Ω

Table 3.3: Description of instrumentation signals for System Boards

The PXIe templates have been provided for this board for future work to commence.

3.4 Implementation of PXI Express system

With the physical design of the PXIe Peripheral Module, the mechanical and electrical requirements were met. The next step was to configure the device to communicate with a host system. Providing PCIe communication was focussed on where the requirements for the device are outlined below:

- Implement the PCI Express Configuration Space to allow the host system to initialise and configure the device.
- Allow memory read and write transactions with the host.
- Make good use of the bandwidth provided by PCIe by providing speeds near the maximum bandwidth of 250 MB/s.

For such a design to be implemented and connected to a host device, three components are required. This includes

- An FPGA design where it is configured using a Hardware Descriptive Language (HDL).
- A device driver is implemented to allow communication between the peripheral device and the host system.
- A user application run on the host system to allow the user to communicate with the peripheral device.

A diagram showing the relationship between these elements is shown in Figure 3.13. As can be seen, the device driver acts as the middle layer to allow communication between the Endpoint and the application on the host system.

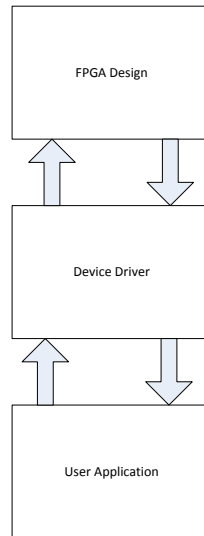


Figure 3.13: Basic design of FPGA based PCI Express Endpoint application

The PXIe Peripheral Module was being made to work in a PXIe chassis (NI PXIe-1062Q) to communicate with the PXIe controller board (NI PXIe-8101). The PXIe-8101 controller card came equipped with a x86 Celeron processor. It came pre-installed with Windows Vista; however it was configured to boot Fedora 10 Linux from a flash drive. Providing an open system which would allow for an ARM based PXIe System Module controller in the future, Linux was the chosen environment for development. This meant that an open source device driver and application was required to communicate with the PXIe Peripheral Module.

3.5 FPGA Design

The FPGA was configured to provide PXIe functionality. As with the design of the physical PXIe board, the device configuration was separated into two discrete parts.

- Implementation of PCIe transmission system
- Configuration for PXIe instrumentation signals such as clock and triggers

The focus of this project was to implement the PCIe transmission system on the device. The instrumentation signals were more implementation specific. Thus the FPGA designs researched and provided were primarily focussed on giving PCIe functionality.

With the FPGA being the central device, this gave a lot of flexibility to the design of the PXIe system, as well as the general function of the device. The Spartan-6 Integrated Endpoint Block for PCIe could be added to the design to implement the base PCIe system [22]. This was verified to meet version 1.1 of the specification and greatly simplified the design process.

The integrated Endpoint block allowed the device to be recognised and configured by the host system but provided no real functionality to the user. A user FPGA design was required to allow memory transfers to be sent along the link. Outlined below is how the integrated Endpoint block was used to implement a base system to send data along the link.

3.5.1 Spartan-6 FPGA Integrated Endpoint Block for PCI Express

The Xilinx Spartan-6 FPGA Integrated Endpoint Block for PCI Express is a hard IP block which can be added to Xilinx project. The configuration is all done inside this block and it provides a multitude of input and output pins that allow the user to tailor the design to their particular needs. This routes the internals of the FPGA to implement the details of the PCIe standard. In fact, it abstracts the Physical Layer and Data Link Layer away from the designer so they can look at things from the Transaction Layer. This greatly simplifies the design as the designer can move straight onto dealing with Transaction Layer Packets (TLPs) rather than dealing with the gritty details of the PCIe protocol such as 8b/10b encoding, data scrambling etc. It

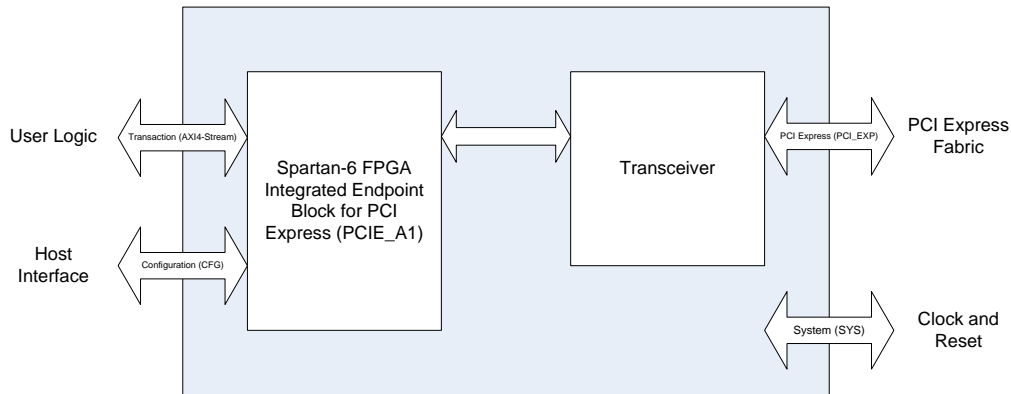


Figure 3.14: Diagram of Spartan-6 Integrated Endpoint Block for PCI Express [22]

also implements the PCI Configuration space so these registers can all be configured using the IP Core wizard.

The integrated Endpoint block meets revision 1.1 of the PCIe specification and implements a x1 link which provides a maximum bandwidth of 250MB/s. Whilst PCIe has now moved onto version 3.0 of the specification, the bandwidth provided by the Spartan-6 chip is more than adequate and offers major advantages over the transmission scheme used in Magritek's present system. Reproduced from the User Guide [22], a block diagram of the integrated Endpoint block is shown in Figure 3.14. The signals used to communicate with the integrated Endpoint block are separated into four types.

- The Transaction signals which use the standard AXI4-Stream interface to communicate with the user design. These are used for the transfer of data in and out of the integrated Endpoint block.
- The Configuration (CFG) signals which the host interface communicates with. These are internal signals which are active when Configuration Read and Writes are called.

- The PCIe (PCI_EXP) transceiver signals which includes differential transmit and receive signals.
- System signals which include the clock and reset signals.

The transaction and configuration signals are all internal signals which can be used to communicate with a user design. The transceiver signals are external signals which the integrated Endpoint block uses for transmit and receive functions. These consist of the signals defined earlier in the chapter in Table 3.1. Thus in the user constraints file (UCF) for the design, these signals must all be mapped to the correct locations which are shown in Figure 3.9. The transceiver and clock signals are provided with defined I/O pins which cannot be changed. The sideband signals however can be connected to any I/O bank provided with the correct voltage. For direct connections which do not use a level shifter, this is a 3.3 V bank. These simply need to be defined inside the UCF.

The use of the AXI-4 stream interface provides a single standard interface to make IP integration easier [27]. This allows complicated designs to be implemented a lot easier as they make use of the same input/output signals. There are a multitude of Xilinx designs that make use of the AXI4-Stream interface [28]. This interface meant that future, more complicated designs, could be produced with reduced development time.

The integrated Endpoint block provides the user with some configurable options. This allows particular registers inside the PCI Configuration Space to be set. The PCI Configuration Space implemented by the Spartan-6 Integrated PCI Express block is shown in Table 3.4. Configuration space options are set in a wizard for the IP Core which is shown in Figure 3.15. The size and type of the PCI configuration space Base Address Registers (BARs) can be set. It allows 32-bit memory spaces, 64-bit memory spaces and I/O spaces to be implemented (though use of I/O spaces has been deprecated). The vendor ID and sub-vendor IDs can be configured here too. Also the maximum payload size for TLPs can be set. The

Spartan-6 Integrated Block for PCI Express

xilinx.com:ip:s6_pcie:2.4

Base Address Registers

Base Address Registers (BARs) serve two purposes. Initially, they serve as a mechanism for the device to request blocks of address space in the system memory map. After the BIOS or OS determines what addresses to assign to the device, the Base Address Registers are programmed with addresses and the device uses this information to perform address decoding.

BAR 0 Options

☒ Bar0 Type: Memory ☐ 64 bit ☐ Prefetchable

Size: 2 Kilobytes

Value: FFFFFFF800 (Hex)

BAR 1 Options

☐ Bar1 Type: N/A ☐ 64 bit ☐ Prefetchable

Size: 1 Bytes

Value: 00000000 (Hex)

BAR 2 Options

☐ Bar2 Type: N/A ☐ 64 bit ☐ Prefetchable

Size: 128 Bytes

Value: 00000000 (Hex)

BAR 3 Options

☐ Bar3 Type: N/A ☐ 64 bit ☐ Prefetchable

Size: 1 Bytes

Value: 00000000 (Hex)

BAR 4 Options

☐ Bar4 Type: N/A ☐ 64 bit ☐ Prefetchable

Size: 1 Bytes

Value: 00000000 (Hex)

BAR 5 Options

☐ Bar5 Type: N/A ☐ Prefetchable

Size: 1 Kilobytes

Value: 00000000 (Hex)

Expansion ROM Base Address Register

☐ Expansion Rom Size: 2 Kilobytes

Value: 00000000 (Hex)

Datashheet < Back Page 2 of 9 Next > Generate Cancel Help

Figure 3.15: Spartan-6 Integrated Endpoint Block for PCI Express Wizard

input frequency provided to the FPGA can be either 100 MHz or 125 MHz so this has to be set.

The integrated Endpoint block provides automated responses for Configuration Space requests. Also, it will detect and report on errors found on packets received in the core. So if the device was used in a host system, it would perform initialisation and configuration. If any of these packets were received malformed or unrecognised, it would report on errors.

A major advantage of the integrated Endpoint block is that the user

Bits [31-24]		Bits [15-8]		Bits [7-0]		Reg
Device ID			Vendor ID			0x00
Status			Command			0x04
Class Code				Revision ID		0x08
BIST	Header Type	Latency Timer	Cache Line Size			0x0C
Base Address Register 0 (BAR0)						0x10
Base Address Register 1 (BAR1)						0x14
Base Address Register 2 (BAR2)						0x18
Base Address Register 3 (BAR3)						0x1C
Base Address Register 4 (BAR4)						0x20
Base Address Register 5 (BAR5)						0x24
Cardbus CIS Pointer						0x28
Subsystem ID			Subsystem Vendor ID			0x2C
Expansion ROM Base Address						0x30
Reserved				Revision ID		0x34
Reserved						0x38
Max latency	Min Grant	Interrupt	Cache Line Size			0x3C
PM Capability		NxtCap	PM Cap			0x40
Data	BSE	PMCSR				0x44
MSI Control		NxtCap	MSI Cap			0x48
Message Address (Lower)						0x4C
Message Address (Upper)						0x50
Reserved		Message Data				0x54
PE Capability		NxtCap	MSI Cap			0x58
PCI Express Device Capabilities						0x5C
Device Status			Device Control			0x60
PCI Express Link Capabilities						0x64
Link Status			Link Control			0x68
Reserved Legacy Configuration Space (Returns 0x00000000)						0x6C-0xFF
Next Cap	Capability Version	PCI Express Extended Capability - DSN				0x100
PCI Express Device Serial Number (1st)						0x104
PCI Express Device Serial Number (2nd)						0x108
Reserved Extended Configuration Space (Returns Completion with 0x00000000)						0x10C-0xFFFF

Table 3.4: PCI Configuration Space Header of the Spartan-6 FPGA Integrated Endpoint Block for PCI Express

design need only deal with the Transaction Layer. Technical aspects at the Data Link Layer such as flow control are abstracted away and automatically handled. Likewise the Configuration read and write requests are handled internally in the integrated Endpoint block.

3.5.2 Format of PCI Express Design

FPGA designs are usually broken up into a number of elements where each element takes care of a particular task. For instance, a simple communication device such as a UART could consist of a Receiver and a Transmitter embedded in a top wrapper unit. This is considered good design practice rather than placing every process inside one large HDL file. A common starting place for FPGA designs is simply a block diagram. Each block in the diagram can then be used as its own element in the FPGA project.

Thus a basic PCIe design was drafted as:

- Integrated Endpoint Block for PCIe to provide basic PCIe functionality
- A receive module to deal with incoming packets available from the integrated Endpoint block
- A transmit module to construct TLPs and send them to the integrated Endpoint block to be transmitted
- A memory element which will place data received in an appropriate location and be available for the transmitter to access to send data out
- A wrapper element to connect the receiver, transmitter and memory element together
- A top wrapper to connect the integrated Endpoint block to the rest of the design.

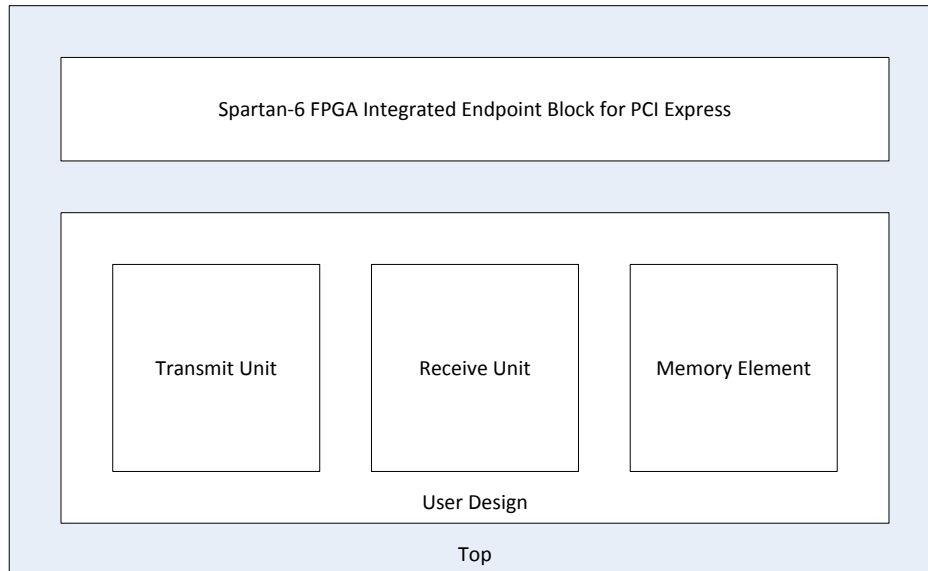


Figure 3.16: Basic format of user design for PCI Express

A block diagram of such a design is shown in Figure 3.16. This was used as the core design which any user design would meet. Designs may include additional elements inside the user design to add functionality. However, any design must include at least these elements.

Three types of data transactions are allowed in PCIe: 32 bit memory, 64 bit memory, and I/O transactions. However, I/O elements have been deprecated in PCIe designs. For simplicity and as being tested in a 32 bit system, in this design 32 bit memory transactions were used.

3.5.3 PCI Express Data Transfer

There are two main forms of communication that can be made between the host system and peripheral elements.

- Programmable Input/Output (PIO) transactions which are instigated

by the CPU where data is sent through the chipset of the host computer.

- Direct Memory Access (DMA) transactions which allows the peripheral device to become the Bus Master (BM) and send data directly to the host memory.

Programmed Input/Output consists of data transfers instigated by the host device where PCIe memory read and memory write requests are sent out and completions received from peripheral devices. This gives all control to the user, which suffers two serious downsides.

- High CPU utilisation as each data transfer needs to be instigated by the host.
- Lack of efficiency in PCIe link as packets can only send 1 DW of data at a time. With each packet requiring at least three DWs of header information, this adds at least 75% overhead.

Direct Memory Access overcomes the two major disadvantages of the PIO system. Here, the bulk of the memory transfer is performed by the peripheral device meaning this task is off loaded from the CPU. This frees up the CPU to perform other tasks whilst the memory transfer is taking place. Also, because data does not need to be presented in 1DW chunks, larger payloads can be appended to PCIe packets, greatly increasing transfer efficiency.

PIO transactions require a receiver unit to detect memory read and memory write transactions and a transmitter unit to send out completion packets. A DMA system also requires the transmitter to construct its own memory read and memory write packets, a receiver to take the incoming completion packets, and a unit to signal interrupts after transactions complete. PIO and DMA transactions and their implementation on an FPGA are described in greater detail below.

3.5.4 Programmed Input/Output

Programmed Input/Output (PIO) allows a system host CPU to access Memory Mapper Input/Output (MMIO) and Configuration Mapped Input/Output (CMIO) in the PCIe fabric. The Endpoint then accepts Memory and I/O Write transactions and responds to Memory I/O Read transactions with Completion and Data transactions.

In a basic PIO system, the CPU must initiate all of the transfers. The transmit unit of the PIO system generates Completion packets. However, it cannot initiate Memory Read and Memory Write transactions. Likewise, the receive unit of the PIO system accepts Memory Read and Memory Write packets. However, it does not accept Completion packets as these would only be sent when responding to a Memory Read or Memory Write request.

A downstream operation takes place when data moves from the Root Complex to the Endpoint. An upstream operation occurs when data moves from the Endpoint to the Root Complex. Downstream operations occur when the CPU initiates a data transfer by sending a store register to a MMIO mapped to the Endpoint. This address gets resolved as being one belonging to a PCIe Endpoint and so the Root Complex generates a Memory Write TLP with the appropriate MMIO location address, byte enables and register (or data) contents. Data moves upstream when the CPU issues a load register from a MMIO. Again, the address gets resolved as belonging to an Endpoint so the Root Complex generates a Memory Read TLP with the appropriate MMIO location address and byte enables. The Endpoint receives this Memory Read TLP, communicates with the memory controller to find the data required and generates a Completion with Data TLP. This Completion is then received by the Root Complex and its payload is loaded into the targeted register. Due to the nature of these operations, each transfer is limited to 1DW of data at a time. If more than 1DW of data is sent, the TLP is ignored with no completion generated.

PIO transactions have major limitations as explained earlier. The data

speeds are effectively 25% or less of the maximum PCIe bandwidth as each DW of data needs to be sent with its own header as transactions of over 1DW or data are not allowed. The other issue with this design is the CPU utilisation whilst the transfers are occurring. For transfers in the Megabytes of data, this would render the CPU unable to perform other tasks for a noticeable time.

3.5.4.1 Example Xilinx PIO Design

When a PCI Express Endpoint Block for PCIe core is added, Xilinx automatically creates an example Programmed Input/Output (PIO) project [22]. A block diagram of the system reproduced from the Spartan-6 FPGA Integrated Endpoint Block for PCI Express User Guide [22] is shown in Figure 3.17. This allowed a basic system for packets to be sent across the PCIe link. This demo project comes complete with source code and does not require any input from the designer to get going. The PIO design allows up to 8KB of memory organised as 4 512DW (2KB) Block RAM banks. Each of the 4 block RAMs are implemented as a 2KB dual-port block RAM. These regions are represented by the BARs from the Endpoint's configuration space. When transactions are received by the core, the core decodes the address and determines which of the regions is being targeted. The PIO design by default allows an IO space, 32 bit memory space, 64 bit memory space and an expansion ROM. For the testing of this project, a 2KB 32 Bit memory space was used. In order to simulate the PIO design, the project came with the Root Port Model simulation program. This simulated the process that would be taken by a host system which the Endpoint would communicate with. This is discussed in further detail in the next chapter. Complimenting the FPGA design, Xilinx also provided an application note XAPP1022, *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* [29]. In XAPP1022, a device driver and user application was provided for Windows and Linux systems, with the Linux design also coming with

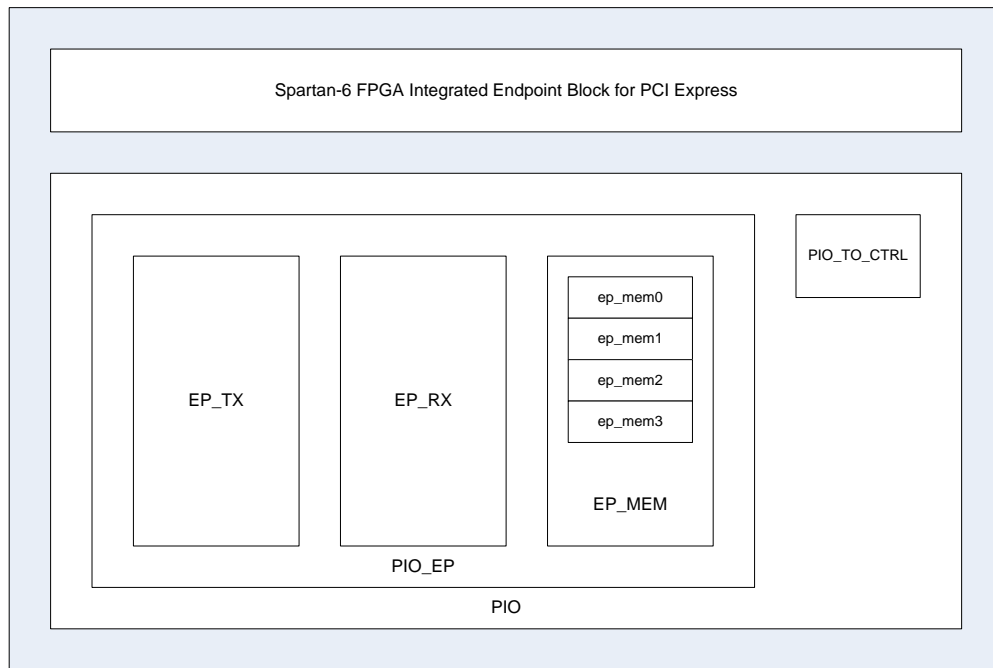


Figure 3.17: Diagram of Programmable Input/Output System [22]

source code.

3.5.5 Direct Memory Access Transmissions

PCIe is a packet based architecture with the ability to send data at high speeds. Thus the ability to perform fast read and write transactions to memory is useful for PCIe systems. To perform this efficiently, if a device is wanting to read or write to the system memory, the device must receive direct access to the memory and thus become the Bus Master. This is what comprises a Direct Memory Access (DMA) transfer.

A DMA transfer in PCIe takes place when the Endpoint generates a Memory Read or Memory Write transaction, rather than simply responding with a completion packet. The DMA transfer can be triggered by a PIO write command from the host or from another external event. A DMA

Memory Write transaction consists of the Endpoint generating a Memory Write packet and sending it across the link. The packet gets received by the Root Complex and passes on the received data to memory. The Endpoint will hold a lock on the memory until the transmission completes. Once the Memory Write has completed, the Endpoint signals an interrupt signal. This gets passed onto the CPU which informs it that the memory has been updated. A DMA Memory Read transaction consists of the Endpoint generating a Memory Read packet and sending it across the link. The Root Complex receives this packet and retrieves the requested data from memory. This then gets sent back to the Endpoint. When all the requested data has been received, the Endpoint will then send an interrupt along the link to signal that the transmission is completed.

3.5.5.1 Xilinx Bus Master Performance Demonstration Design for the Xilinx Endpoint PCI Express Solutions

Xilinx provides the application note XAPP1052 *Bus Master Performance Demonstration Reference Design* [30]. XAPP1022 provides an example Bus Master DMA system. This gives the HDL to implement on the Spartan-6 FPGA used. Also provided is source code for the device driver and application for both Linux and Windows environments.

The DMA unit had more advanced transmit and receive units to handle these DMA transfers. The memory element also differed in that it comprised of a number of registers which could be read and set to control the operation of the device. Within the transmit unit was an additional interrupt module, used to tell the Endpoint Block to generate an interrupt request. In the application note, a diagram of the DMA system was given which is shown in Figure 3.18. As can be seen, the system is made up of a number of modules. The PCIe Endpoint Block links in with the interface block which the other modules are nested inside. The top wrapper includes the integrated Endpoint block generated from IP Wizard and the interface. Control and status registers contain operational information for

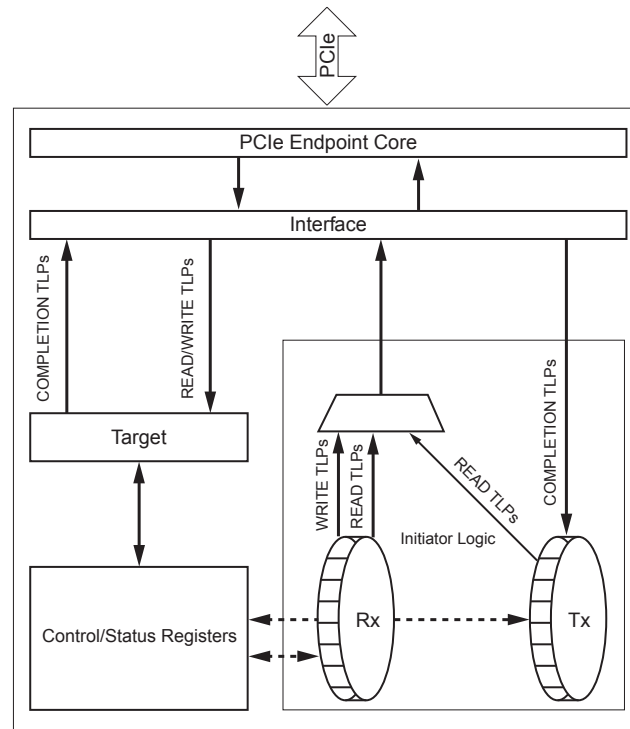


Figure 3.18: Diagram of Xilinx DMA design [30]

the DMA controller. The example design is used to measure performance of data transfers.

The engine works with PIO transactions similar to the PIO Example design. However it makes use of an older version of the integrated Endpoint block which used the TRN interface rather than the newer AXI-Stream interface. The initiator block generates Memory Read or Memory Write TLPs depending on if upstream or downstream transfer is selected. The design only supports generating one type of data flow at a time. The Bus Master Enable bit (Bit 2 of PCI Command Register) must be set to initiate TLP traffic upstream. The initiator logic generates Memory Write TLPs when transferring data from the Endpoint to the system memory. The write DMA control and status registers specify the address, size, payload content, and number of TLPs to be sent.

3.5.5.2 Other PCI Express DMA solutions

A number of vendors provide FPGA IP DMA solutions for PCI Express devices [31, 32]. These provide high performance systems which can be placed on FPGAs to take care of DMA transfers. The SP605 board was provided with a node locked licence for the Northwest Logic DMA Engine. The issue with these systems is that they are licensed systems where the source code cannot be shared freely. They are also primarily provided only in Verilog which did not meet Magritek's requirements.

3.5.6 Problems and improvements to be made on sample Xilinx designs

One option for providing PCI Express functionality to the designed PXIe Boards would have been to make use of one of the provided Xilinx designs. The PIO design provided by Xilinx allowed basic PCIe transfers to be made; however it did not make efficient use of bandwidth. The XAPP1052 reference design provided fast transfer speeds through DMA; however it made use of the older version of the integrated Endpoint block which used the TRN interface. The FPGA design for the designed PXIe Peripheral Module was instead provided in VHDL and made use of the latest version of the integrated Endpoint block. Using the AXI-Stream interface provided future proofing of the design as newer devices only support the AXI-4 interface.

Another issue with the reference DMA design is its use of Verilog HDL. Although this is a well supported language, it does not provide the same strong typing that VHDL does, as discussed earlier. Also, VHDL is widely used for FPGA designs at Magritek hence was the sensible choice for the HDL to be written in.

There were also issues with the design of the finite state machines used in both the PIO project and the XAPP1052 DMA engine. This is explained in further detail below.

Finite State Machines As explained earlier, data was sent and received to the integrated Endpoint block in 32 bit or 1 DW chunks. Thus the data output would vary depending on which of these DWs was being processed. In order to implement such a system on the FGPA, a finite-state machine (FSM) was used. The machine can only be in one particular state at a time where a task gets completed depending on what DW of the packet was being processing.

There are two major types of state machines named Mealy and Moore. The output of a Moore state machine is determined solely by its current states. Alternatively, the output of a Mealy state machine is determined also by the input values. Given the number of options available for PCI Express packets, the best model for the state machine implementation was Mealy, else a large amount of states would be required for all the particular conditions.

A state machine design is typically separated into combinatorial and sequential sections. Such a design is shown in Figure 3.19. This sort of design is useful when spikes in the output are tolerable where the inputs may change. However, if this is not appropriate, registered outputs can be used. Such a design is shown in Figure 3.20.

In the example Xilinx designs, the state machine did not follow one of these standard formats. Instead, the combinatorial and sequential sections were combined into one large process.

```
if rising_edge(clk) then
    ...
    when (state) is
        WRITE =>
            if (condition = '1')
                output <= '1';
                state <= WRITE;
            end if;
    ...
end if;
```

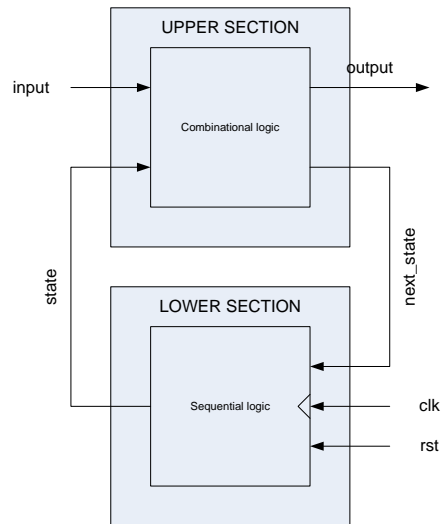


Figure 3.19: A Mealy Finite State Machine

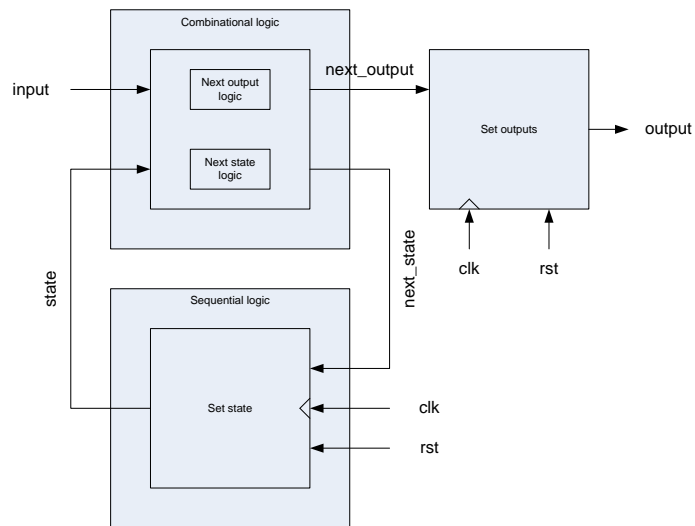


Figure 3.20: A Mealy Finite State Machine with registered outputs

```
end if;
```

All transitions were designed like this. This is considered bad practice as it is making decisions based on the state and inputs (combinational) and setting the state (sequential) inside the same process [33]. This is why the `set_next_state` and `next_state_logic` shown in Figures 3.19 and 3.20 are separated into different processes.

3.5.7 **FPGA design for use on PXI Express Peripheral Module**

The PXIe FPGA design was based on the XAPP1052 reference design. This allowed an open source DMA system to be designed as well as providing an open source device driver and application for Linux which could be further developed. The XAPP1052 reference design was used to test throughput speeds thus using this as the base design provided the ability to measure and analyse the performance. The HDL for the PXIe Peripheral Module however was provided in VHDL and made use of the integrated Endpoint block which used the AXI4-Stream interface. Unused signals were removed from the design and more rigorously designed state machines were used. Following the structure of the XAPP1052 design, FPGA design used on the PXIe Peripheral Module is shown in Figure 3.21.

The memory element of this design consisted of a number of defined registers inherited from the XAPP1052 design. Also, the transmit unit includes an interrupt controller to signal the end of memory read and memory write transactions to the CPU. Also included was the read metering unit. A configuration element was also included to read parameters such as the max payload size, the MSI address and the maximum link width. This would allow the system to configure itself to the requirements of the transmission. Making use of the same general design of the XAPP1052 application meant that the device driver and user application provided could be used without modification.

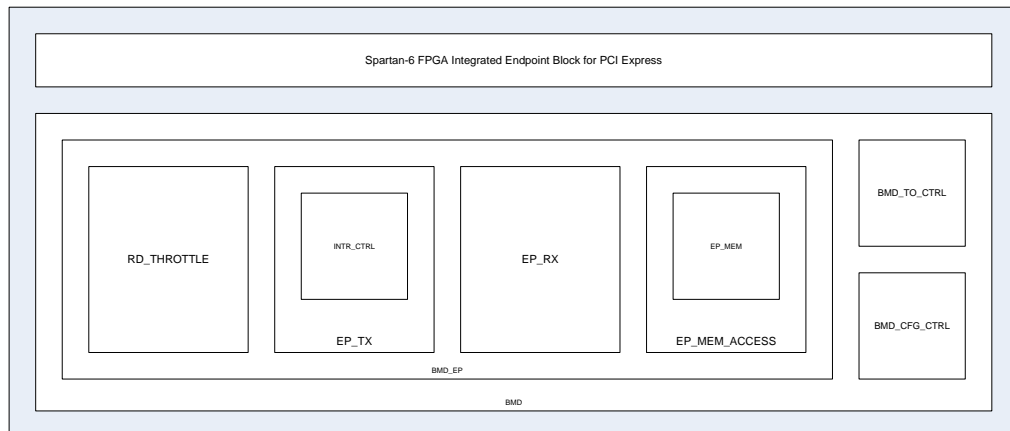


Figure 3.21: Diagram of DMA Design

XAPP1052 was a sample design to test throughput for Xilinx based DMA systems. The design could test the performance of DMA Memory Read and Memory Write transactions. The performance measurement was made by starting a counter when the data transfer was initiated. This would then be stopped on the completion of the transfer. By reading back the number of clock cycles, the performance of the system could be calculated. This measurement gave the actual data transfer rate rather than just the raw bit speeds.

As described in [21], there is no major technical advantage of FPGA designs configured in VHDL over Verilog. However for a number of reasons it was decided VHDL was the best language of choice for this project. These reasons included:

- Strong typing and verboseness of the language meant the design could be more rigorously defined.
- VHDL does not allow sequential assignments to be made thus any sequential assignments made in the XAPP1052 Verilog code was removed and the VHDL description altered to suit.
- FPGA development at Magritek is primarily done in VHDL thus it

improved integration and ease of modification.

- Process of porting the Verilog source code to VHDL gave a more comprehensive understanding of the system.

Providing the system in VHDL required porting the 10 different modules provided in the XAPP1052 application to VHDL. The structure of processes differed greatly between the two languages so these had to be rewritten. Standard assignments were similar in VHDL and Verilog where the `:=` character was used to set vectors. These assignments were synchronous where the order of the assignments was unimportant. However, Verilog also made use of sequential assignments which required the HDL to be restructured in parts. In VHDL, assignments are always blocking in that they wait until the next clock cycle before assignments are made. This is typical of hardware designs which are concurrent in nature. For example, see the following code snippet.

```
if rising_edge(clk) then
    ...
    if (condition = '1')
        a <= '1';
        b <= a;
        c <= b;
    end if;
    ...
end if;
```

Here, 'a', 'b' and 'c' are all '0' before the assignment. Following the assignment 'a' would equal '1' whilst 'b' and 'c' would both equal '0'. This is because of the blocking mechanism of VHDL. It would not matter in what order 'a', 'b' and 'c' were assigned in this statement. An equivalent assignment could be made in Verilog. However Verilog can also perform sequential or non-blocking assignments using the `=` character. For example, see the following code snippet.

```
always @ (posedge clk) begin
    ...
    if (condition = '1') begin
        a = '1';
        b = a;
        c = b;
    end;
    ...
end;
```

Again, 'a', 'b' and 'c' are both equal to '0' before this assignment. After these assignments took place, 'a', 'b' and 'c' would all equal '1' as each assignment would occur sequentially. Such a feature is common in software (such as C which Verilog inherits its syntax from) where sequential instructions are performed. Thus for such processes to be performed in VHDL (without the ability to perform sequential assignments) the assignment would be made in the previous state. Making such changes allowed the order of the assignments more obvious to the system designer.

As previously mentioned, the design of the state machines in the XAPP1052 required improvement. These were altered in the FPGA design for the PXIe Peripheral Module. How and where the outputs are set in a HDL state machine are somewhat down to personal preference. In some designs, the outputs get set inside the same process as the next state process. This sort of design is appropriate for a state machine with few output signals. However, this design involved numerous output signals. Also, the particular outputs set vary greatly depending on the current state. For this reason, registered outputs in their own process were used. When an incoming clock signal was received, it would set data depending on what the current state was. The state machine used in the design for the PXIe Peripheral Module was based on the three always block design given in [34]. The diagram of how the state machines were implemented using three processes is shown in Figure 3.22. An example of how this was writ-

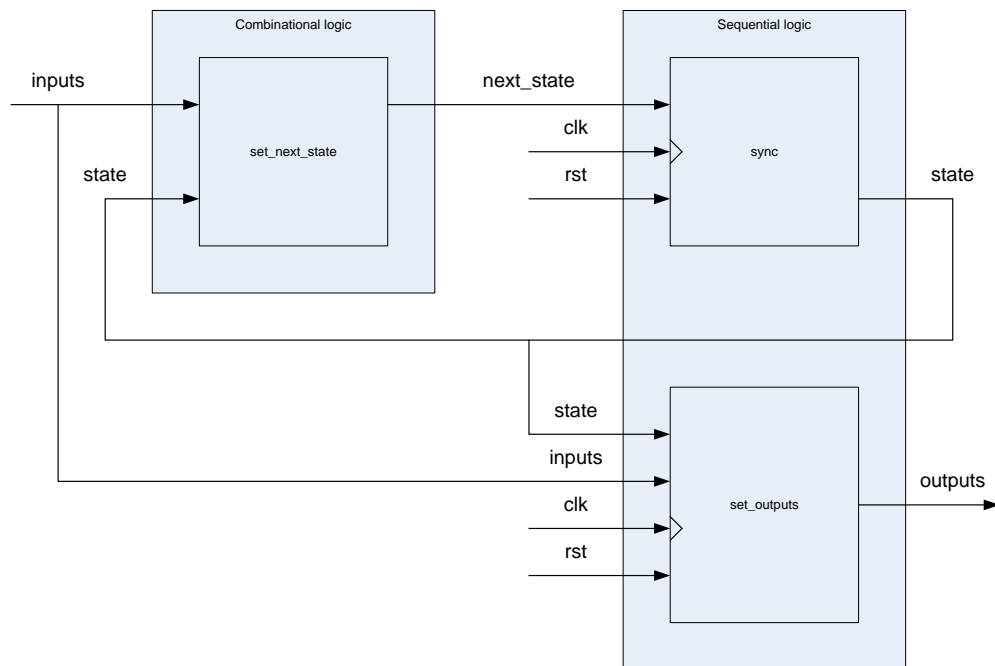


Figure 3.22: Design of State Machines in PXI Express FPGA project

ten in VHDL is shown below.

```

sync : process(clk, rst)
begin
    if (rst = '1') then
        state <= RESET;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process;

```

```

set_next_state : process(...)
begin
    case (state) is

```

```

        when RESET =>
            if (condition_1 = '1')
                next_state <= STATE_1;
            ...
            end if;
        ...
    end case;
end process;

set_outputs : process(clk, rst)
begin
    if (rst = '1') then
        output_1 <= '1';
        ...
    elsif rising_edge(clk) then
        case (state) is
            when RESET =>
                ...
            ...
        end case;
    end if;
end process;

```

The design of the state machine meant that outputs were set on a rising clock edge which would also cause a state transition. In this way, outputs were also set at the end of a state rather than at the start of a state.

The XAPP1052 used the older TRN interface and also was made for a whole range of FPGAs which included signals which the Spartan-6 required. To move to the newer version of the integrated Endpoint block, this required the previously used TRN signals to be replaced by their AXI4 equivalents. If there was no equivalent signal, the description was altered to work using the the AXI4-Stream interface. The move to the AXI4-

Stream standard meant that, if required, the design could be more easily ported to a newer FPGA which did not support the legacy TRN interface.

3.5.7.1 Memory Unit

The memory unit used in the design kept the same structure as that used in the sample Xilinx DMA design. In order to access this memory space, the integrated Endpoint block wizard was configured to set BAR0 as a 1KB memory element. The memory element consisted of a number of registers used to control the operation of the device. These would set parameters such as, what data would be sent and received and the length of each transaction. The registers and their location relative to the start of the 1KB BAR are shown in Table 3.5.

Address	Register Name	Register Contents
0x00	DCSR Device Control Status Register	Bits 31-24: FPGA Family Bits 19-16: Core Data Path Width Bits 15-8: Version Number Bit 0: Initiator Register
0x04	DDMACR Device DMA Control Register	Bit 31: Read DMA Operation Data Error Bit 24: Read DMA Done Bit 23: Read DMA Done Interrupt Support Bit 22: Read DMA No Snoop Bit 21: Read DMA Relaxed Ordering Bit 16: Read DMA Start Bit 8: Write DMA Done Bit 7: Write DMA Interrupt Diabale Bit 6: Write DMA No Snoop Bit 5: Write DMA Relaxed Ordering Bit 0: Write DMA Start
0x08	WDMATLPA Write DMA TLP Address	Bits 31-2: Write DMA Lower TLP Address
0x0C	WDMATLPS Write DMA TLP Size	Bits 31-24: Write DMA Upper TLP Address Bit 19: 64 bit Write TLP Enable

		Bits 18-16: Write DMA TLP TC Bits 12-0: Write DMA TLP Size
0x10	WDMATLPC Write DMA TLP Count	Bits 15-0: Write DMA TLP Count
0x14	WDMATLPP Write DMA Data Pattern	Bits 31-0: Write DMA TLP Data Pattern
0x18	RDMATLPP Read DMA Expected Data Pattern	Bits 31-0: Read DMA Expected Data Pattern
0x1C	RSMATLPA Read DMA TLP Address	Bits 31-0: Read DMA Low TLP Address
0x20	RDMATLPS Read DMA TLP Size	Bits 12-0: Read DMA TLP Size Bits 31-24: Read DMA Upper TLP Address Bit 19: 64 bit Read TLP Enable Bits 18-16: Read DMA TLP TC
0x24	RDMATLPC Read DMA TLP Count	Bits 15-0: Read DMA TLP Count
0x28	WDMAPERF Write DMA Performance	Bits 31-0: Write DMA Performance Counter
0x2C	RDMAPERF Read DMA Performance	Bits 31-0: Read DMA Performance Counter
0x30	RDMASTAT Read DMA Status	Bits 15-8: Completions w/ UR Tag Bits 7-0: Completions w/ UR Received
0x34	NRDCOMP Number of Read Comple- tions with Data	Bits 31-0: Number of Completions with Data
0x38	RCOMPSIZW Read Completion Data Size	Bits 31-0: Total Completion Data
0x3C	DLWSTAT Device Link Width Status	Bits 13-8: Negotiated Max. Link Width Bits 5-0: Capability Max. Link Width
0x40	DLTRSTAT Device Link Transaction Size Status	Bits 18-16: Max. Read Request Size Bits 10-8: Programmed Max. Payload Size Bits 2-0: Capability Max. Payload Size
0x44	DMISCONT	Bit 8: Receive Non-Posted OK

	Device Control	Miscellaneous	Bit 1: Read Metering Enable
			Bit 0: Completion Streaming Enable
0x48	DMSICONT	Device MSI Control	Bit 27: cfg_interrupt_msienable Bits 26-24: cfg_interrupt_mmenable Bits 23-16: cfg_interrupt_do Bit 8: LEGACYCLR Bits 7-0: INTDI

Table 3.5: Registers in DMA Reference Design

All the unmentioned bits are simply reserved and left at '0'. The DMA system can be controlled simply by setting these registers. These registers get written to and read by performing PIO transactions instigated by the host system. Each PIO request can only hold one DW of data, which is the size of one register so these get written to and read 1DW at a time.

In order to perform a Memory Write DMA transfer with the Endpoint, the following process was followed.

- Reset system (PIO Write DCR1) - 0x00000001
- De-assert Initiator Reset (PIO Write DCR1) - 0x00000000
- Write DMA H/W Address (PIO Write WDMATLPA) - H/W Address
- Write DMA TLP Size (PIO Write WDMATLPS) - Write TLP Size
- Write DMA TLP Count (PIO Write WDMATLPC) - Write TLP Count
- TLP Payload Pattern (PIO Write WDMATLPP) - Data Pattern
- Write DMA Start (PIO Write DCR2) - 0x00000001
- Wait for Interrupt TLP
- Write DMA Performance (PIO Read WDMAPERF)

The reset of the system clears any possible transaction that may have been taking place and resets the data registers to 0. The system is then de-asserted to allow the data registers to be set again. The registers on the Endpoint are then set which includes the address for the Endpoint to communicate with, the size of transfer, the number of transfers and the data pattern. The bit to start the transfer is then set which starts the Endpoint transfer. Once the total data is sent out, the transaction completes and the Endpoint signals an Interrupt TLP to the host to indicate this. The measured performance of the Endpoint then gets read back via a PIO transfer.

In order to provide this functionality, the DMA unit was to provide the following transactions:

- Receiving of PIO Memory Read packets and sending completions in response
- Receiving of PIO Memory Write packets and updating data
- Sending Memory Read packets and receiving the completions
- Sending Memory Write packets

Thus the Transmit and Receive Units had to be written to deal with these transaction types. These are detailed below.

3.5.7.2 Receive Unit

The receive unit included a finite state machine which would process all the possible packets it could receive. The packets included:

- Memory Read packets
- Memory Write packets
- Completion packets

Memory Read and Memory Write packets were known to be PIO transactions from the host unit which would carry 1DW payloads. The state machines for these transactions followed how they were given in the example Xilinx design. The receiving of Completion packets were responses to DMA Memory Read transfers. The state machine for this system is given in Figure 3.23. As well as the transition details given in the state machine diagram, transitions also required the `m_axis_rx_tvalid` and `m_axis_rx_tready` signals to be high. The `m_axis_rx_tvalid` signal is provided by the integrated Endpoint block to tell the user application when the data is valid. The `m_axis_rx_tready` is provided by the user design to let the integrated Endpoint block know when it is ready to receive data.

Memory Read packets The reception of Memory Read packets follows the process shown in Figure 3.23. On probing the first DW of data, bits 30 to 24 of the data signal show the packet type is detected as a 32 Bit Memory Read Transaction ("0000000"). It also checks if the packet has a payload of 1DW (which PIO transactions should meet). It then sets the data included in this first DW of the header. After one clock cycle, it moves on to the next DW of the header. It then sets the other bits such as the request id and tag. Then it moves to the final DW of the packet which provides the address that the host is attempting to read. The receive unit then sets `req_compl_o` high, signalling to the transmit unit to send a completion packet in response. It then moves into the wait state until the completion is completed (`compl_done_i = '1'`) before moving back to the reset state.

Memory Write packets The reception of Memory Write packets follows the process shown in Figure 3.23. The first DW of data is read with bits 30 to 24 indicating a 32 Bit Memory Write Transaction ("1000000"). It sets all requisite data and moves onto the next DW. Here the BE is read and moves onto the next DW. The address is then read which gives the register being written to. The data is then set to the `wr_data_o` vector and `wr_en_o`

set high to enable the register to be written. It then moves in to the wait state until the write is completed (`wr_busy_i = '0'`).

Completion packets Completion packets follow a similar process to that of the Memory Read and Memory Write packets. It checks the TLP type, and begins the processing of the Completion Packet. It then extracts information to check what the completion was in response too. When processing the payload of the packet, as this is a simple Endpoint test application, it simply checks the received data up against the expected data. If it is correct it will carry on checking the data. If not, it will signal an error signal to indicate that it has not received the data successfully. Once each DW of data is checked, it is simply discarded.

3.5.7.3 Transmit Unit

The transmit unit also included a finite state machine which would process all the possible transactions it could fulfil. This included:

- 32 bit and 64 bit memory write packets
- 32 bit and 64 bit memory read packets
- Completion packets

The completion packets are generated in response to PIO transactions from the host. These would always offer 1DW of data. The 32 and 64 bit memory read and write packets were the DMA transfers initiated by the Endpoint. The finite state machine which processed all these transfers is shown in Figure 3.24. As well as the transitions given in the state machine, each transition was also dependent on `s_axis_tx_tready` being high. This is provided by the integrated Endpoint block to the user application to inform when it is ready to receive more data to transmit. This was to prevent data being loaded in the integrated Endpoint block when it still had not processed the previous lot of data.

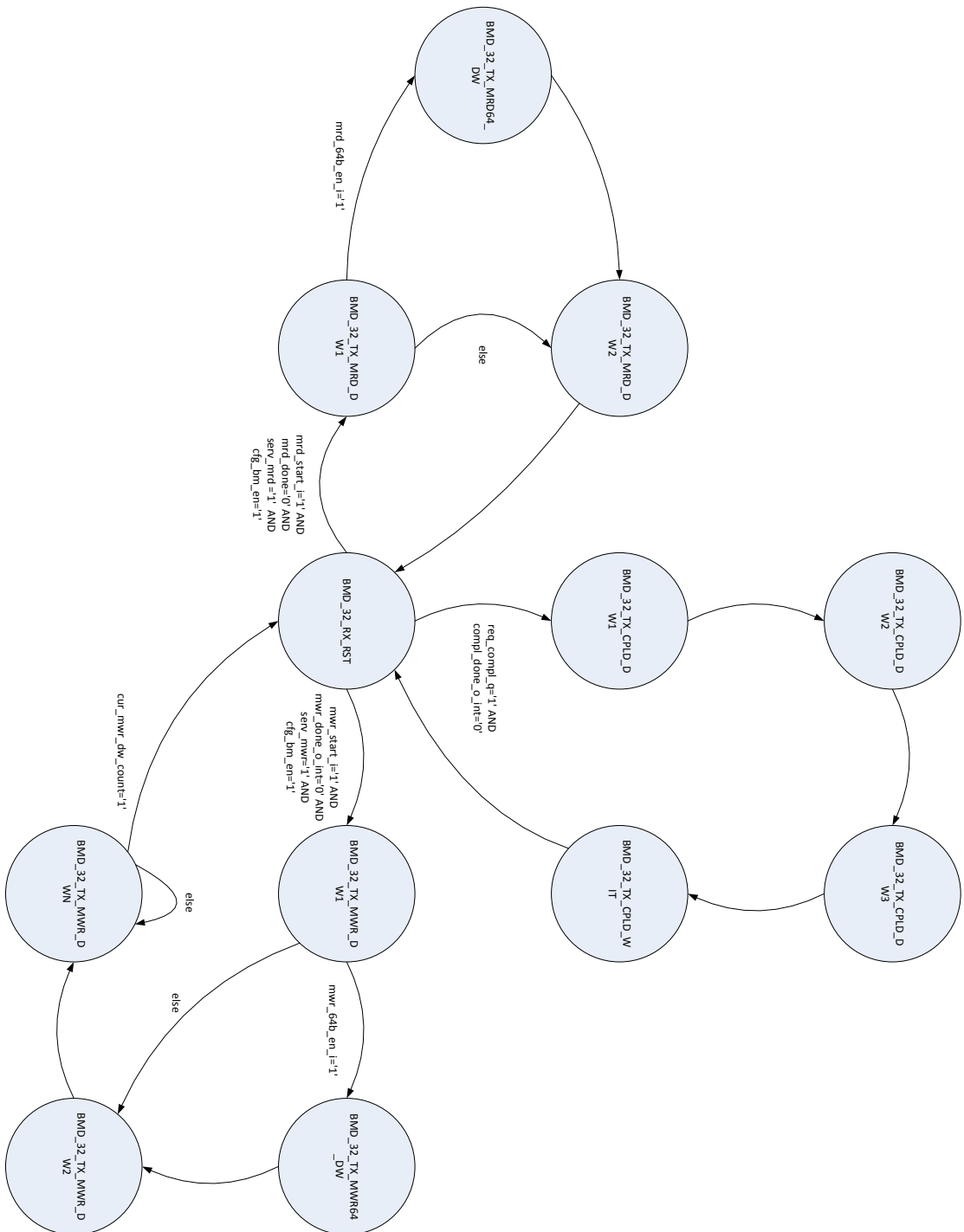


Figure 3.24: State Machine of Transmit Unit

Memory Read Packets When the conditions are met for Memory Read transaction, it will follow this routine in the state machine. These conditions are that a read request has been signalled (`mrd_start.i = '1'`), the current read request has not finished (`mrd_done = '0'`), ongoing write requests are not blocking read requests (`mrd_serv = '1'`) and bus mastering is enabled (`cfg_bm_en = '1'`). It then places the appropriate data in DW0 of the header and moves onto the next state. Here it continues placing data in the header. If a 64 Bit address is being used, there is an extra optional state in the state machine to deal with this case. This would provide the higher bytes required to communicate with the system. It then moves on to DW2 where it places the lower bytes of the address. Once this is complete it moves to the reset state where it waits for the completion data to be received.

Memory Write Packets When conditions are met for a Memory Write transaction, it follows this routine in the state machine. These conditions are that a write request has been signalled (`mwr_start.i = '1'`), the current write request is not finished (`mwr_done.o_int = '0'`), a memory read request is not blocking write requests (`mwr_serv = '1'`) and bus mastering is enabled (`cfg_bm_en = '1'`). The appropriate data is then placed in the first DW of the header and it moves on to the next DW of the header. It then places the appropriate data for the second DW of the header. If a 64 bit memory write request is being made, it moves onto this optional state where it would provide the higher bytes required. It then moves on to DW2 where it places the lower address. This is the end of the header where it then moves onto placing data in the packet. This is provided from the data that is set in the data register of the design. This will continue until the maximum allowed payload size has been reached or the memory write request has completed. If there is more data to send to complete the transfer, it will create another memory write packet. Otherwise, it will move into the reset state.

Completion Packets When conditions are met to respond to a request with a Completion Packet, this routine in the state machine is followed. The conditions being that a completion request has been signaled (`req_compl_q = '1'`) and the completion has not finished (`compl_done_o_int = '1'`). It fills in the required header information as required in each DW, including the packet type and address. At DW3, it places the data requested from the targeted register. Finally, it moves to the wait state where it will then move to the reset state if the `s_axis_tx_tready` signal is high, indicating it is ready to receive more packets to transmit.

Interrupt Unit As well as sending out packets of data, the Transmit unit was also responsible for issuing interrupts to the CPU to indicate the DMA transfer had completed. Traditionally, the CPU receives interrupts by having one of its interrupt pins written to. For PCI transactions, these pins were labelled INT1 through INT4. However, with newer generations of CPUs this is used less and less and instead Message Signalled Interrupts (MSIs) are used. An MSI consists of setting a small amount of data to a special address in memory space. The chipset then delivers the corresponding interrupt to the CPU. It has a few advantages over legacy style interrupts. Using this special piece of memory space to signal interrupts allows the CPU to have less pins which makes for a simpler, cheaper and more reliable connector. MSI also increases the number of possible interrupts. Conventional PCI limited the number of interrupts to 4 per card. However, as all cards sat on the same bus, this meant that typically they only had access to one interrupt line. MSI on the other hand allows dozens of interrupts per card. This was not a particular advantage for this application but could be useful later, for example if there were multiple data transmissions occurring simultaneously.

PCI Express allows interrupts to be triggered by both of these mechanisms. For legacy style interrupts, a memory write transaction is performed by writing to a particular memory location which the root complex

maps as being one of the INT1-4 locations. The style of interrupt used by the Endpoint is decided by the integrated Endpoint block. During the configuration phase, the Endpoint communicates with the Root Complex which provides information as to what interrupt mechanism is supported. There are two sorts of interrupts that are triggered, interrupts for Memory Write transactions and interrupts for Memory Read transactions. When a Memory Write transaction completes, the transmit unit triggers an interrupt to signal to the CPU that the data has been updated. When the Endpoint receives the last completion packet in response to a Memory Read transaction, the transmit unit will also generate an interrupt to signal that it has finished reading data.

3.5.7.4 Implementation on FPGA

The FPGA design for the PXIe Peripheral Module had to be programmed to the FPGA itself. The Xilinx ISE is used to generate a bit file from the HDL which can then be used to program the FPGA. The device gets programmed through communication using the Joint Test Action Group (JTAG) protocol. This is a widely used protocol for debugging and programming purposes.

When FPGA devices are programmed, the configuration is only valid whilst power is provided. Once power is removed, the FPGA loses its configuration. Thus if the configuration needs to stay through power cycles, the configuration data needs to be saved on board. This is done by saving the configuration to some flash memory on board. Thus when power gets restored to the device, it will load the configuration from the flash device.

The FPGA design was configured for the designed PXIe Peripheral Module as well as the SP605 development board. Each of these devices used the same Spartan-6 FPGA thus the FPGA design could be tested first on the SP605 board.

When a host system starts, it performs configures all the connected peripheral devices. Thus in order for the board to be detected by the host

system, the FPGA needed to be configured on startup. However, when power is not supplied, FPGAs lose their configuration. Two methods were used to have the device configured to be detected by the host:

- The configuration could be loaded onto some on board flash memory so that once power is received, the FPGA would configure itself fast enough to be detected by the host.
- The host is powered up with the peripheral device connected where the FPGA is then configured directly using JTAG. Once this was achieved, a soft reset is performed so the host restarts and is then able to detect the device on start up. This process is a lot faster however, the configuration is not saved to the device and loses it when the system is powered down. However it was useful in testing the operation of the device where any possible problems with using the flash memory method could be circumvented.

On the SP605 board, a 64 Mb flash memory device was used, the X4W25Q64VSFIG. This is communicated with the FPGA using the Serial Peripheral Interface Bus (SPI) interface. This is a four wire communication protocol which allows for x1 to x4 communication speeds. x4 was used to get the FPGA to configure as quickly as possible.

In order to allow these devices to function correctly, particular settings in the Xilinx compilation process had to be configured. This was done by adding some additional parameters to the compilation, which could be done in the ISE design suite as shown in Figure 3.25. Firstly, the project was configured to allow for an external clock signal as it was sourcing the clock signal off the backplane. To use the .spi flash device, it had to be set to allow x4 configuration mode.

Communication with the device was provided with JTAG where the bundled program IMPACT was used to configure the FPGA. A screenshot of the program is shown in Figure 3.26. When connected over JTAG, this probes the device and detects the attached FPGA and flash device on

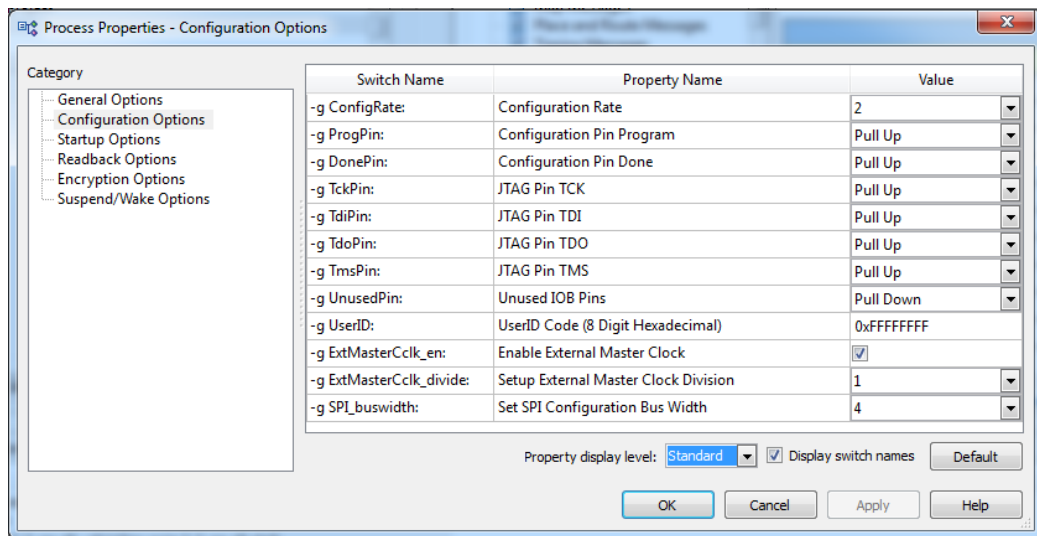


Figure 3.25: Configuration Options in Xilinx ISE Project Navigator

board. When a boundary scan is performed, this finds the FPGA and any attached flash memory. As can be seen, the FPGA is detected along with a connected SPI/BPI flash device. The FPGA can be directly programmed with the .bit file, else a .mcs file is used. A .mcs file is generated by first setting the targeted device (this being a .spi 64Mbit device). The .bit file can then be converted to the required .mcs file.

The performance of the DMA system and its comparison with the other FPGA designs is detailed in the next chapter.

3.6 Backplanes for testing devices

As a means of testing the card's physical design and the FPGA work for configuring the system, backplanes for testing the devices were created. As Xilinx FPGAs were being used for the designed PXIe Peripheral modules, FPGA development was tested first on Xilinx development board including the Virtex-6 Connectivity Kit [35] which contained the ML605 board and the Spartan-6 Connectivity Kit [24] which included the SP605

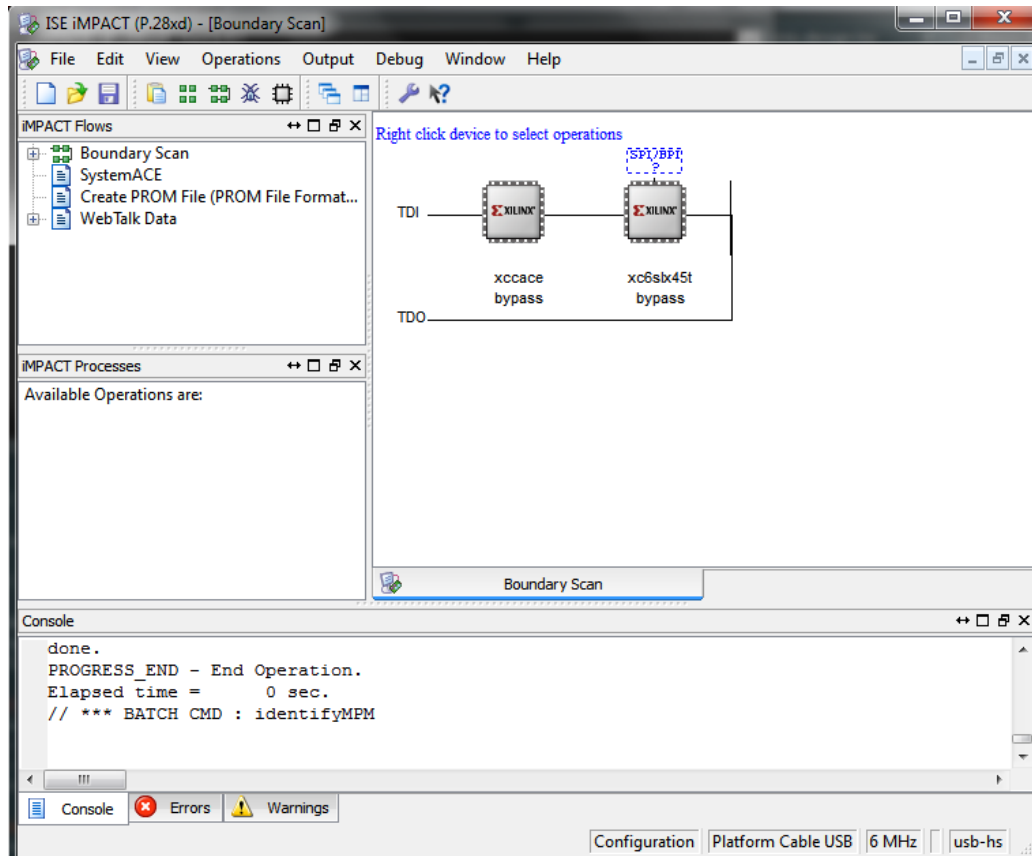


Figure 3.26: Screenshot of the program IMPACT when connected to the SP605 board over JTAG

board. The Virtex-6 is a high end FPGA offered by Xilinx which gives a x8 lane PCI Express device over the simple x1 lane which the Spartan-6 provides. FPGAs allow soft processors to be configured using their logic. The Virtex-6 would be configured to run a MicroBlaze soft processor core on which Linux could be installed. Thus the Virtex-6 would be configured as the host device to communicate with the SP605 board. Thus backplanes were to be provided which would enable communication between these Xilinx development boards as well as PXIe Controller Boards and Peripheral Modules.

These backplanes were designed to test both the physical design and FPGA design of the designed PXIe modules. As well as this, the preliminary design here could be adopted for construction of a full PXIe chassis. The impedance matching, connector spacing and circuit design could all be directly ported for use on a PXIe chassis.

The three backplanes would operate as so:

- An x8 PCI Express slot routed to a x1 PCI Express slot (so only one lane would be connected from the x8 slot). In this setup the Virtex-6 Connectivity Kit (ml605) would plug into the x8 slot and behave as the host of the system and the Spartan-6 Connectivity Kit (SP605) would be the Peripheral Module. These boards require an input clock signal which will be provided on the backplane.
- An x8 PCI Express slot routed to a PXIe Peripheral slot. The PXIe functionality and wider NMR use will not be tested on the PXIe Peripheral Module, merely the PCI Express and physical design of the board. One lane would be routed between the x8 PCI Express slot and the PXIe Peripheral Slot. As with the other backplane, a clock is required to be provided on the backplane to the two slots.
- A PXIe System Slot (the host) connected to a x1 PCI Express slot. Initially, the PXIe-8101 National Instruments System Module would act as the host and communicate with the SP605 board. For future use,

a designed PXIe System Module would be tested here. PXIe System Modules provide the PCI Express clock source to the peripheral modules. Thus the clock signal is derived from the System Slot and routed across the backplane to the x1 PCIe slot.

- A PXIe System Slot to PXIe Peripheral Slot. As before, the clock signal is derived from the PXIe System Module and routed to the PXIe Peripheral Slot.

As well as the clock signals routed to each slot, the differential transmit and receive signals were required to be routed between the connectors. The overview of the PCIe to PCIe backplane whole backplane is shown in Figure 3.27 where the other backplanes all followed a similar format. The routing of signals and power are described in further detail below.

3.6.1 Power

PCI Express boards can source their power from the slot they are connected to. However, as the Spartan-6 and Virtex-6 boards were for general development, they would not necessarily be plugged into a PCI Express slot so had alternative power connectors. Thus when the boards were connected to a PCI Express slot, they were powered by way of a standard Molex 4-pin connector which had been traditionally used for PATA and low-end SCSI disk drives. This connector plug can be seen in Figure 4.3. The power plug to connect to this is usually sourced off an ATX motherboard power supply. Because of this power requirement, it was decided to use an ATX motherboard power supply to draw power for these backplanes. This meant that the molex connectors would be provided, as well as all the required voltages (3.3V, 5V, 5Vaux and 12V). The PXIe devices and on board circuitry would all be provided power on the backplane.

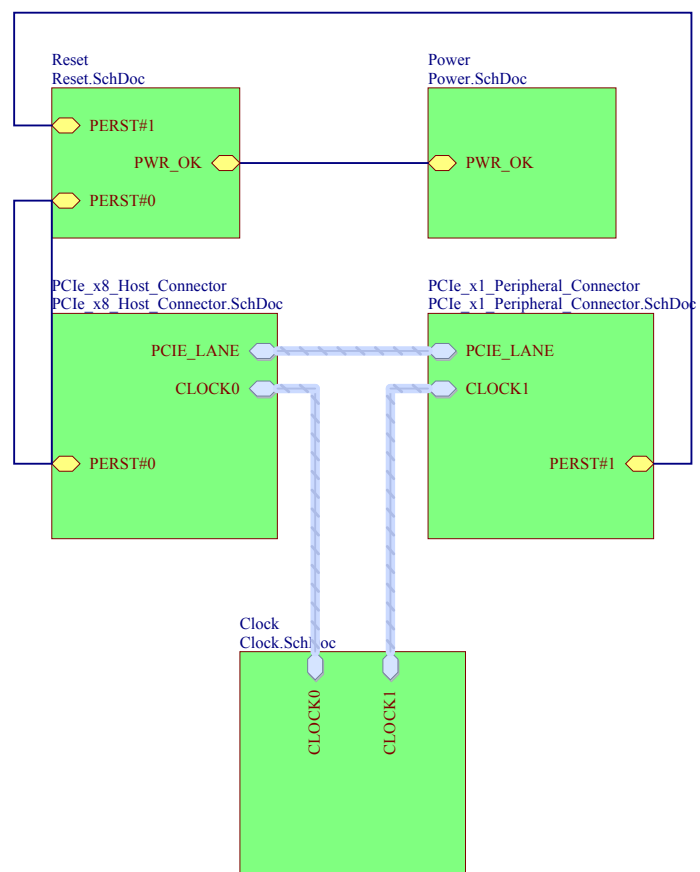


Figure 3.27: Top level view of PCIe to PCIe backplane

3.6.2 Reset Circuit

Another important signal in the PCI Express specification is the PCI Express Reset (PERST#) signal. PERST# is driven low to cards until 100 ms after voltage levels reach stability. If at a later time the ATX power supply falls out of stability, PERST# should also be driven low. To provide this system, the use of the ATX power supply was especially useful. ATX power supplies have a 'Power Good' signal which is driven high when the power supply voltage rails reach stability. Thus the 'Power Good' signal can be used as the PERST# signal to be delivered to the connectors on the backplanes.

The implementation is slightly more complicated though. The PCIe specification requires that the power levels be stable for at least 100 ms before the PERST# signal is de-asserted. In order to do this a simple timer circuit was implemented on the backplane. The timing diagram is shown in Figure 3.28. The schematic circuit for this is shown in Figure 3.29. When the 'Power Good' signal goes high, this gets inverted to trigger a 555 timer configured as a monostable to start timing. The output of the 555 is used as the trigger for a D flip flop. When the transition is made from high to low from the 555 timer, the output of the D flip flop goes high so drives the PERST# signal high. The 'Power Good' signal is also used as the D input for the flip flop. This means that when power levels had stabilised and the 555 timer had completed, only then would the output of the D flip flop go low, hence de-asserting PERST#.

3.6.3 Signal integrity

The boards were fairly basic however attention to signal integrity was required. A PCIe link runs at 2.5Gb/s and these high speeds require care to be taken when routing the signals. Routing recommendations for each PCI Express link was given in design guidelines in [36] [37]. This gave recommended track width, differential signal separation and general tips

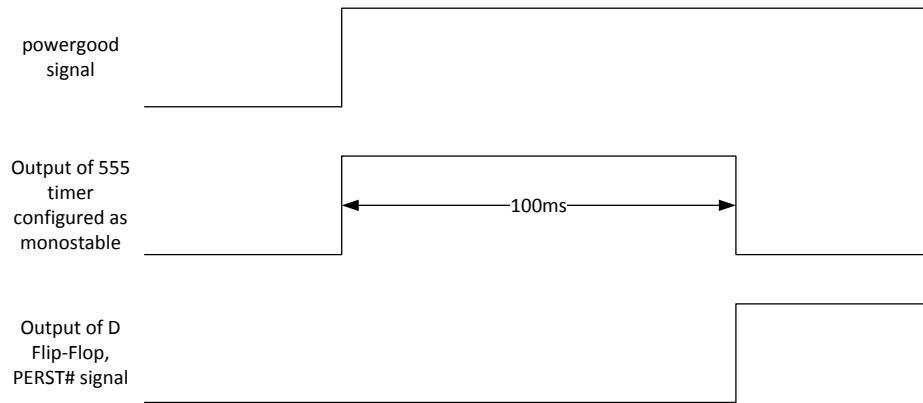


Figure 3.28: Timing diagram of PERST# circuit

on avoiding noise. However, these recommendations were for 6 to 8 layer boards thus the calculations to find track widths required for characteristic impedances were much different.

3.6.3.1 Impedance Matching

To meet specification, the tracks needed to meet a single-ended impedance of $50 \pm 10 \Omega$ and differential impedance $100 \pm 10 \Omega$

Originally, due to the relatively simple nature of the boards and low component count, it was planned for these backplanes to be made from two layer PCBs. These were to be 1.6 mm FR4 boards with a copper pour of 1oz per square foot or 1.4 mils. This gave copper thickness of 0.036 mm. The board stack up for such a board is shown in Figure 3.30.

In order to calculate the impedance requirements, the details of the board had to be collected. The impedance calculation was done according to the microstrip calculation due to the tracks sitting on the outside of the board. The equation for calculating the single ended impedance is given in Equation 2.1.

The parameters of the board were trace thickness (t) = 0.036 mm, dielec-



Figure 3.29: Schematic of PERST# circuit

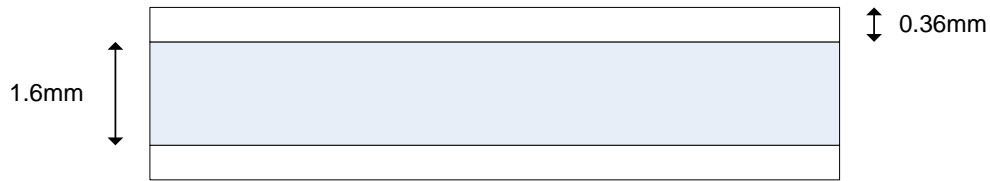


Figure 3.30: Structure of standard 2 Layer PCB

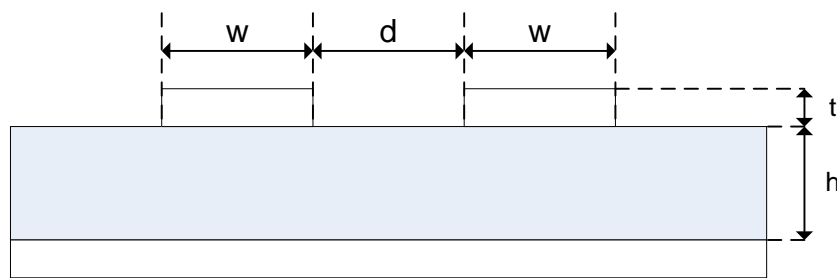


Figure 3.31: Parameters involved in microstrip calculation

tric thickness (h) = 1.6 mm and dielectric constant of $\epsilon = 4.2$. Substituting multiple values of track width into Equation 2.1 gives a required width of around 2.5 mm to meet the impedance requirements. Given the pin separation of many of the connectors is less than 2 mm, this would not allow for routing to the pins. Thus another method had to be tried.

As the impedance was proportional to the dielectric thickness, if the dielectric thickness could be reduced, this would mean the required track width would not be so great. However, a 0.3-0.4 mm board would not prove sturdy. To get this separation down without reducing the overall thickness to a very small width, it was decided to move to a four layer design. The default board stack for this was an inner core of 1 mm with outer layers 0.3 mm either side of this. Thus if routing on the top or bottom layer, this would give a microstrip case with a dielectric height of 0.3 mm. This board stack is shown in Figure 3.32. As well as meeting the single ended impedance, a differential impedance of 100 Ω had to be met. The



Figure 3.32: Structure of standard 4 Layer PCB

equation for this calculation is shown in Equation 2.2.

$$Z_d = \frac{174}{\sqrt{\epsilon_r + 1.41}} \ln\left(\frac{5.98h}{0.8w + t}\right) (1 - 0.48 \exp(-0.96 \frac{d}{h}))$$

Firstly, the track width was decided on by trying multiple values to work with the single ended impedance equation. It was desired to have the tracks as thin as possible so that routing to the closely separated pins would be easiest. A track width of 0.4 mm was chosen as this was the smallest available which met the requirements as this gave an impedance of 59.4 Ω . With the track width chosen, the track separation was decided by using the differential microstrip equation. A separation of 0.3 mm gave an impedance of 97.0 Ω so this was chosen.

3.6.3.2 Length Matching

As well as meeting the impedance requirements, other steps were taken to ensure the signal integrity. Because of the use of differential signals, the propagation delay of each line has to be the same else they will appear out of phase. Hence both tracks on a pair need to be the same length. Following the guidelines given in [37], the length difference between each member of a pair should be no longer than 5 mils (or 0.127 mm). As the

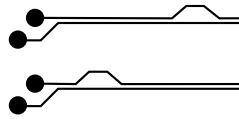


Figure 3.33: Use of serpentine to correct length mismatch. The incorrect method is shown above with the correct method shown below

board was routed according to metric measurements, this was taken as 0.1 mm, however it was attempted to match the signals closer than that. Also according to the guidelines, all mismatches, must be corrected near the mismatch. An example of this is shown in Figure 3.33. When routing from the pins of a connector, one pin was further away and thus the track length is longer. This must be corrected close to this mismatch rather than further along the track. Also shown in the diagram, is the style of length matching used. Serpentine were used with 45 degree bends as per the design recommendation. Each section of serpentine must be at least 3 times the track width, or 1.2 mm in this case.

To route the differential signals, some of the advanced tools provided by the Altium software package were used. A screenshot of these tools is shown in Figure 3.34. Board design in Altium makes use of user defined rules around track width, signal separation and so on. These were so when routing tracks across the board, the software would check for any breaking of these rules and default to the set track widths and separation given.

3.6.3.3 Other steps

As well as meeting these length and impedance requirements, other steps were taken to maintain signal integrity. The routing of the signals on the bottom layer which included the differential transmit, receive and clock signals can be seen in Figure 3.35. The longer the trace is, the greater the loss will be due to the greater impedance along the line. For this reason, the high frequency differential signals were kept as short as possible. To provide this for the clock signals, the on board clock was put as close as

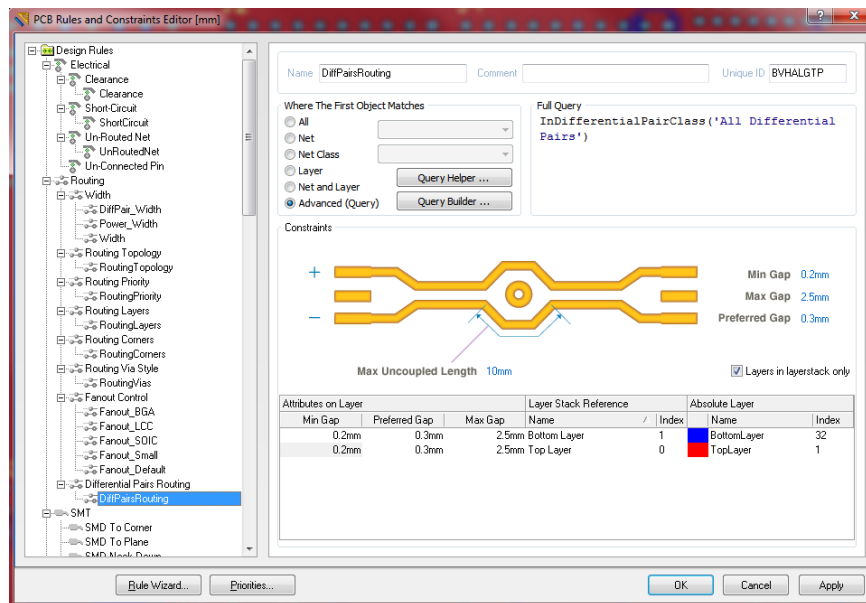


Figure 3.34: Screenshot of Altium PCB Rules and Constraints Editor

possible to the connectors. Also, the slots to hold the PCI Express and PXIe boards were kept as close as possible whilst still providing adequate space to allow the two boards to both be installed in the backplane. Although it was important to keep the track lengths to a minimum, it was also necessary to keep the transmit, receive and clock signals a moderate distance apart so as to not allow coupling between the lines. This is recommended from [37] to be at least four times the track width. Thus when routing signals out from the connectors, they were first routed outward somewhat to keep adequate distance between the various differential signals. As well as the above steps, the differential pairs were kept on one layer only and they were not routed through vias. Routing through vias would have changed the characteristic impedance and introduced reflections into the signal path. Thus, by keeping the signals only on one layer, this meant that no possible problems with losses through vias would be encountered.

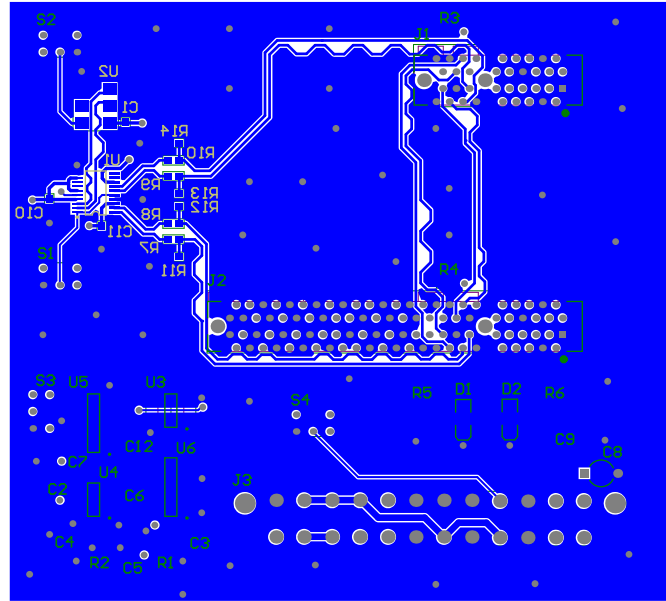


Figure 3.35: Bottom layer of PCB for backplane showing the routing of differential signals between connectors

3.6.4 Component Selection

With the schematics completed the boards for each backplane were laid out. First footprints for each connector had to be created. For a design of such high frequency operation, through hole components were avoided where possible due to the board noise they create. Some of the footprints were standard and existing Altium libraries existed hence were imported whilst others had to be created by referencing the components or connectors data sheets.

3.6.5 Completed PCBs

The PCBs were sent to be manufactured and components for the PCBs ordered. The boards are shown in Figures 3.36 and 3.37. At the time of writing, these boards were still being prototyped and had yet to be populated and tested.

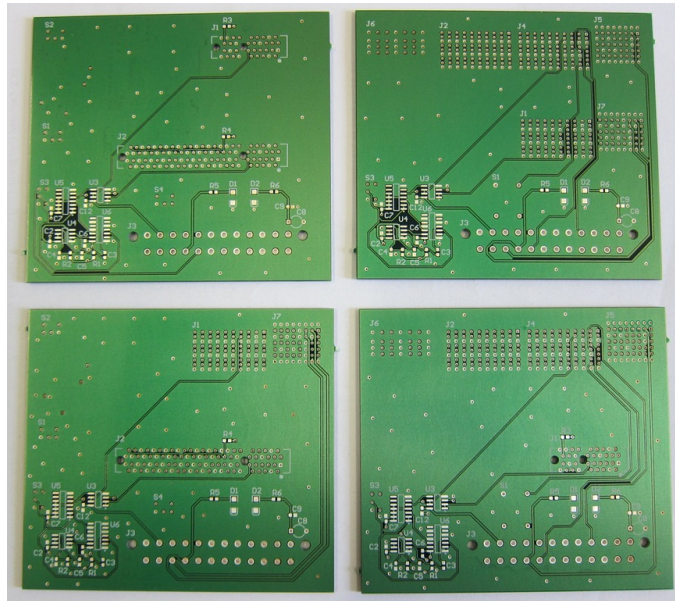


Figure 3.36: Top layer of backplane PCBs

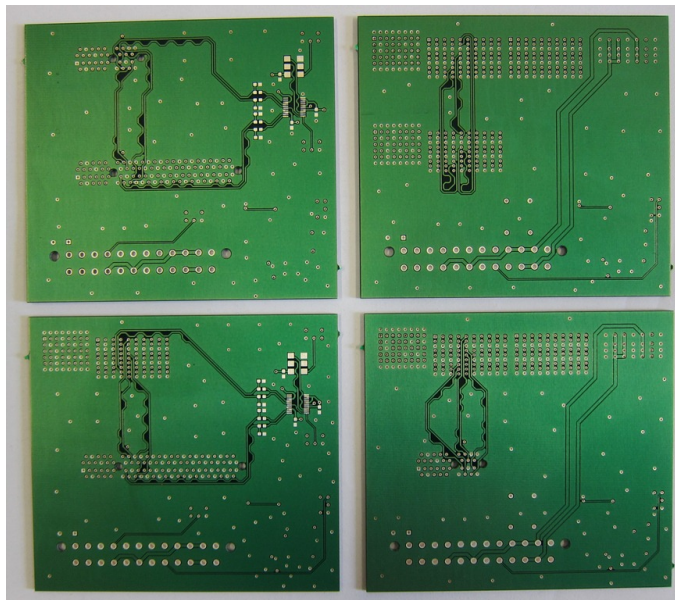


Figure 3.37: Bottom layer of backplane PCBs

Chapter 4

Testing and Design Evaluation

This chapter details the testing and evaluation of the designed system. This includes the physical and electrical testing of the designed PXIe Peripheral Module, the evaluation of the FPGA designs and the implementation of the FPGA on the designed PXIe Peripheral Module.

4.1 Testing of PXI Express Peripheral Module

The physical design of the PXIe Peripheral Module was tested to ensure it met the electrical and physical requirements of the PXIe protocol. This is described in detail below.

4.1.1 Electrical testing of PXI Express Peripheral Module

Some preliminary testing with the board was performed where power was applied externally. This was done by using a PC power supply to supply the correct voltages to all of the power rails. This was found to work as intended and would allow programming to take place over a JTAG connection. This setup is shown in Figure 4.1.

However, in the development of the board, the lack of I/O pins available on the FPGA meant that some signals considered unessential were

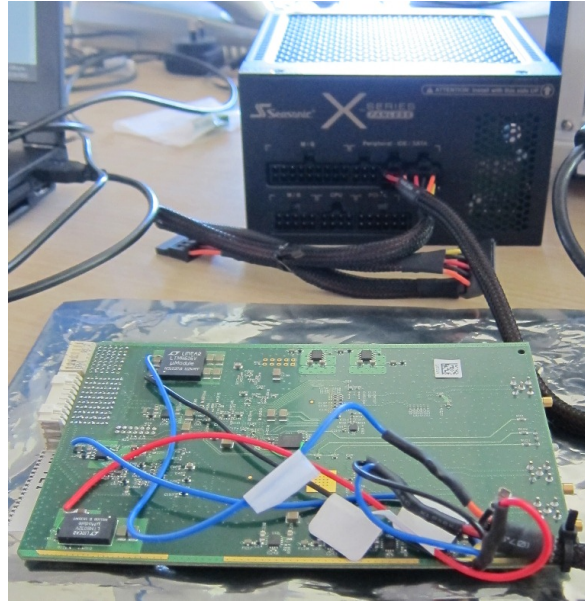


Figure 4.1: Designed PXI Express Peripheral Module powered by PC power supply

removed from the design. For this reason, the PERST# reset signal was left unconnected. The lack of this signal meant that the board did not strictly meet the PXIe specification. However, testing of the module (detailed later) confirmed that the device still functioned correctly. This signal will be added to further revisions of the device.

4.1.2 Mechanical testing of PXI Express Peripheral Module

When the PXIe Peripheral Module was received back, it was confirmed to meet the board dimensions given in the PXIe specification [11] as stated in the background chapter. The designed board connected to the PXIe-1062Q chassis is shown in Figure 4.2. This confirmed the board met the required mechanical requirements as it was able to be inserted in the chassis and powered up (indicated by the lit green LED). The cover and handle used to secure the board in the chassis was not designed, however this could



Figure 4.2: Designed PXI Express Peripheral Module installed in the NI PXIe-1062Q Chassis

be added on a later revision. There were also some subtle issues with the board manufacturing which would be fixed for the next revision. For instance, footprints used for the PXIe connectors were found to be slightly too small. These had to be altered to allow the connector to fit.

4.2 FPGA Designs

The FPGA designs described in the previous chapter were then tested, evaluated and compared.



Figure 4.3: Xilinx PCI Express Spartan-6 Connectivity Kit Board - The SP605

4.2.1 Test Configuration

The PCI Express functionality was first tested on the Xilinx SP605 board. This uses the same Spartan-6 FPGA as used on the designed PXIe Peripheral Module (XC6SLX45T-3FGG484C). It contains a 1x PCIe finger so can be installed in a PC. This board is shown in Figure 4.3. As PXIe is an extension of PCI Express, the PCI Express design was first tested on the SP605 board and then ported onto the designed PXIe device. This was useful for initial testing of the FPGA design as particulars of the PXIe design did not have to be considered.

The device was tested primarily in Linux and required an environment for these to take place. The particular operating system used was Fedora 10 [38]. Although this was a somewhat outdated operating system, it was secure and Xilinx applications recommended this environment [29, 30].

This meant that no difficulties such as driver incompatibilities would be experienced.

The FPGA design was tested on the SP605 board and on the designed PXIe Peripheral Module. The test configuration for these devices is detailed below.

4.2.1.1 Test Configuration for SP605 Board

Revision 1.1 of the PCI Express standard was being implemented which was introduced in 2004 [8]. This meant that a leading edge computer was not required to test the device. A computer with a motherboard with PCI Express Revision 1.1 slots was required. The host device used met the following specifications.

- Intel Pentium 4 630 3.00 GHz processor
- Intel Corporation D945GNT (J3E1) Motherboard
- 1 GB DDR2 RAM
- 75 GB Seagate HDD
- 1 x16 PCIe slot, 2 x1 PCIe slots

The test setup is shown in Figure 4.4. The chipset of the motherboard used was the Intel 945G [39]. This allowed a maximum payload size of 128 bytes to be sent which is the smallest allowed for a link. As the Endpoint accepted up to 512 bytes, this limited the capabilities somewhat. However, the largest packet size sent by the DMA user application was 128 bytes so this was appropriate.

Commonly, PCI Express devices source their power off the backplane. However the SP605 was powered by alternative means. When the board is disconnected from a host computer, power is provided through a 9 pin DC power unit. When connected to a host PC, it draws its power from a



Figure 4.4: Host PC Setup used for testing SP605 Xilinx Development Board

4 pin molex connector. The power provided from the PCI Express slot is simply left unconnected. The molex connector provides ground, 12V and 5V power rails. Any other voltages get up or down from here as required.

The Integrated Endpoint Block for PCI Express gets configured depending on the specifics of the board design. The integrated Endpoint block can take in two clock frequencies, 100 MHz or 125 MHz. On the SP605 board, the 100 MHz REFCLK signal is converted by way of a PLL to 125 MHz. Thus the integrated Endpoint block was set to accept a 125 MHz input frequency. In fact, there was an option to set all the parameters as required for the SP605 development board, as this was a Xilinx produced board. This is shown in Figure 4.5. This was set in the integrated Endpoint block wizard.

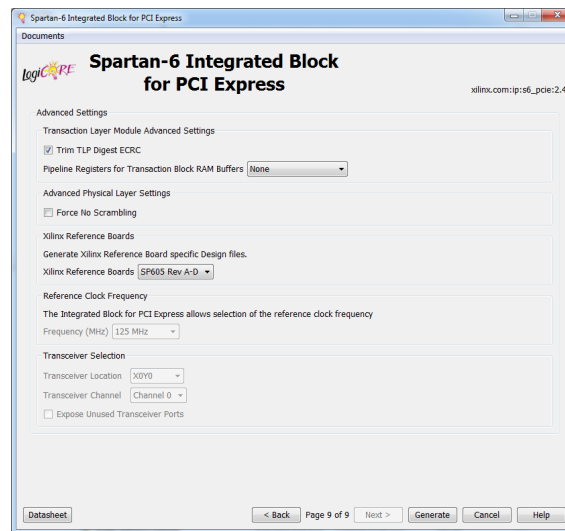


Figure 4.5: Setting PCI Express design to use the settings for the SP605 development board in Spartan-6 Integrated Endpoint Block for PCI Express wizard

4.2.1.2 Test Configuration for designed PXI Express Peripheral Module

The National Instruments PXIe-1062Q fitted with the PXIe-8101 embedded controller card was used for testing the designed PXIe Peripheral module. The embedded controller card provided specifications of:

- Intel Celeron 575 2.0 GHz processor
- Four x1 PCI Express links
- 1GB of DDR2 RAM
- 80GB onboard HDD
- Gigabit Ethernet, 2 USB ports and a DVI port for video out

As the embedded controller card was fitted in the PXIe-1062Q chassis, this provided slots for three PXIe boards, where one could be a System Timing Module. The other PCI Express link provided with the embedded

controller card was used for a PCI-to-PCI Express bridge to provide bandwidth to the legacy PXI slots. Detailed specifications were not provided for the chipset used however the diagram is shown in Figure 4.6. The maximum allowed payload size was not relevant as the sample application used packet sizes of 128 bytes which is the minimum accepted packet size for PCI Express devices [8].

The PXIe System Module system came installed with Windows Vista and could also run an embedded version of LabVIEW for running their particular data acquisition and testing devices. For testing of the module however, Linux was required as this was the intended application host environment for the system. Rather than changing the configuration and partitioning of the hard drive, Fedora 10 Linux was installed on a flash drive and the PXIe system was tested in this environment.

4.2.2 Software for testing

As well as providing the hardware to test the devices in, software was required for communication. As explained earlier, the device drivers and applications were provided by Xilinx and were used as is. This included the device driver and application provided from Bus Master Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions [30] and Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores [29].

Detection of the devices was confirmed using the linux *lspci* command. This performs a read of all the PCI devices connected to the computer and reports back in the terminal. An example *lspci* command run in Fedora 10 is shown in Figure 4.7. The line "05:00.0 Memory controller: Xilinx Corporation Device 0007" shows the detection of the Xilinx FPGA design. The *lspci* associates the Vendor ID and Device ID of 10EE and 0007 respectively as a Xilinx device and reports as such.

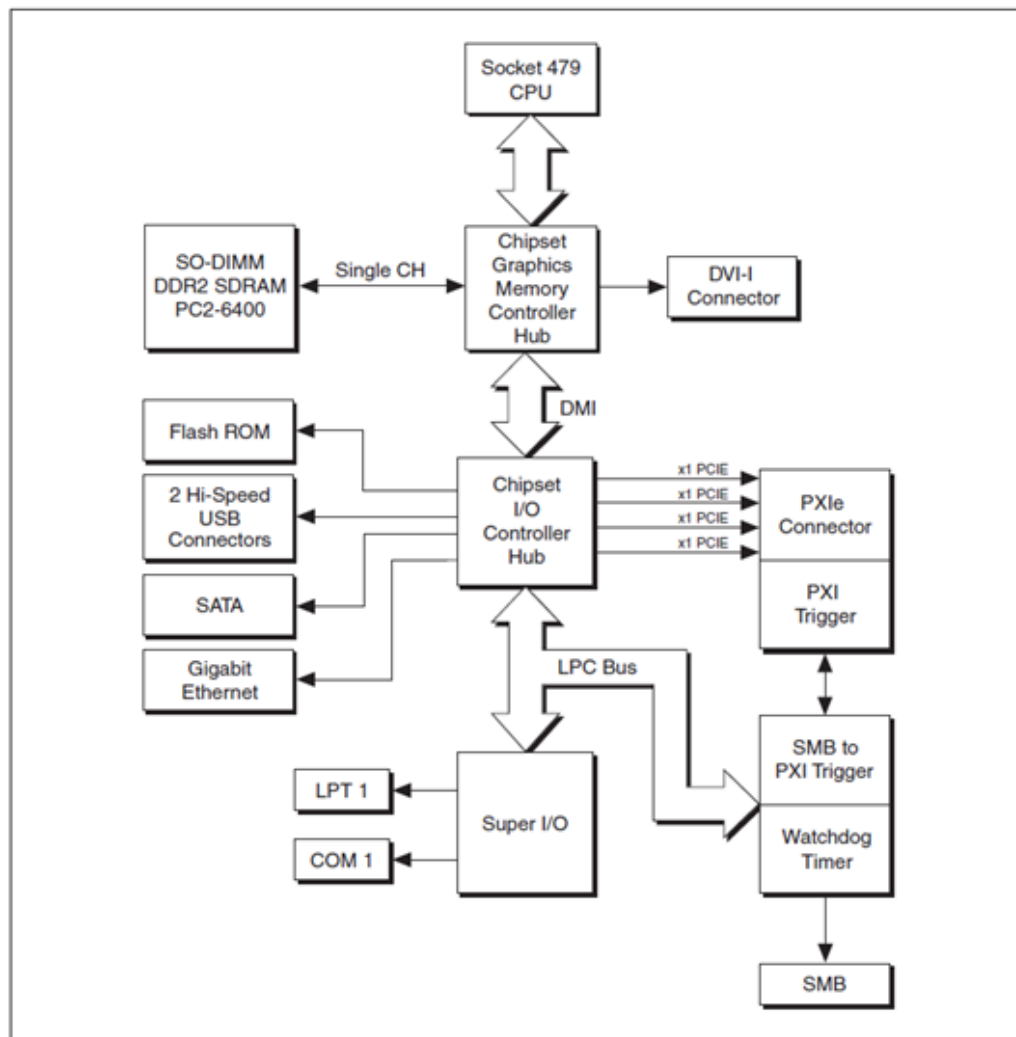
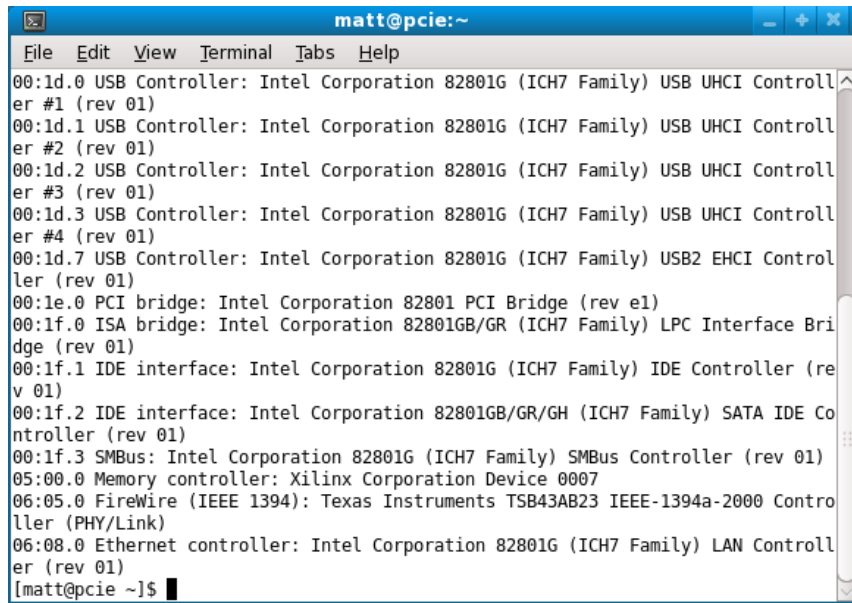


Figure 4.6: Block diagram of PXIe 8101 Embedded Controller card



```

matt@pcie:~
File Edit View Terminal Tabs Help
00:1d.0 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #3 (rev 01)
00:1d.3 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #4 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801G (ICH7 Family) USB2 EHCI Controller (rev 01)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev e1)
00:1f.0 ISA bridge: Intel Corporation 82801GB/GR (ICH7 Family) LPC Interface Bridge (rev 01)
00:1f.1 IDE interface: Intel Corporation 82801G (ICH7 Family) IDE Controller (rev 01)
00:1f.2 IDE interface: Intel Corporation 82801GB/GR/GH (ICH7 Family) SATA IDE Controller (rev 01)
00:1f.3 SMBus: Intel Corporation 82801G (ICH7 Family) SMBus Controller (rev 01)
05:00.0 Memory controller: Xilinx Corporation Device 0007
06:05.0 FireWire (IEEE 1394): Texas Instruments TSB43AB23 IEEE-1394a-2000 Controller (PHY/Link)
06:08.0 Ethernet controller: Intel Corporation 82801G (ICH7 Family) LAN Controller (rev 01)
[matt@pcie ~]$

```

Figure 4.7: Screenshot in Fedora 10 showing `lspci` command in terminal

The PIO FPGA system was tested, and found to be sending and receiving data successfully though the throughput speeds were not measured. However, as each DW of data was accompanied by 3DW of header, this 75% overhead meant that the system performance was at best, 62.5 MB/s. The performance of the XAPP1052 application and the FPGA design used for the PXIe Peripheral Module was measured. The sample user application used for both designs provided throughput speeds to be measured. A screenshot of the test application is shown in Figure 4.8. The settings were kept consistent for all of the testing to allow a fair comparison point.

- Test DW data of 0xFEEDBEEF (default data given by user application)
- 32 DWs data to be sent for each packet
- 32 packets transmitted giving a total data size of 4096 bytes
- Each test run 100 times

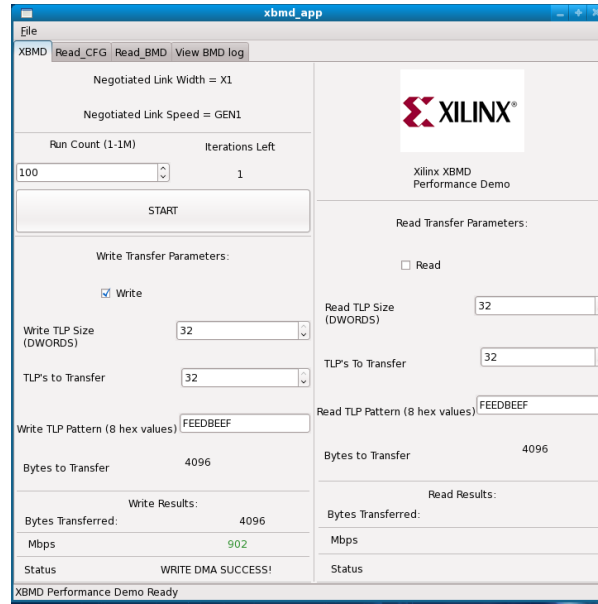


Figure 4.8: Screenshot of user application running

The measurement was made using a counter to operate whilst the data transfer was occurring. After the transfer had completed, the number of clock periods was read from the WDMAPERF register for Memory Write measurements and RDMAPERF register for Memory Read measurements. The internal clock speed of the system was 62.5 MHz. As the data was clocked out at 2.5 Gb/s, this meant that the system was clocked once every 40 data cycles. This is because 1 DW of data becomes 40 bits when encoded by the 8b/10b encoding. The user application provided the performance measurement of a test, however the equation used for calculating it from the number of clock cycles was incorrect. Instead the transmit speeds were calculated from the measured clock cycles during a transmission using the following equation.

$$\text{Speed} = \frac{\text{Total data} \times \text{Frequency of clock}}{\text{Number of clock cycles}}$$

Performance measurements were received by performing PIO transactions on the register space. These registers are 32 bits wide and the user

application read back the values in their hex representation. These were first converted to decimal which could then be used to calculate the overall speed of transmission.

4.2.3 PIO Design

Initial testing of the integrated Endpoint block was done by making use of the example PIO system. It gave an overview as to how the PCI Express transactions are accomplished and showed the major downsides of using only PIO transactions. In the XAPP1022 application note, Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores, device drivers and user applications were provided for the device.

In order to test the design, the Endpoint was configured such that the driver could communicate with the device. This could communicate with 32 bit memory elements thus a single Base Address Register (BAR) in place BAR0 was configured as a 2 KB 32 bit memory space. The Vendor ID and Device ID were set as 10EE and 0007 respectively as these are the default IDs for Xilinx devices. The class code was set with a base class of 05 (indicating a "memory controller") and a sub-class of 80 (indicating a "other memory controller"). The payload size was left at the default size of 512 bytes.

This basic design was tested in two ways. Firstly it was tested in simulation using the Root Port Model. After this was verified, it was compiled and programmed on the SP605 board.

4.2.3.1 Root Port Model

The Root Port Model was full simulation program used to verify the PIO project to test the function of the device. The Root Port Model functioned as the host computer so the interactions between the Endpoint and the computer could be analysed. The simulation was comprised of a number

of testbench units and other test scripts. The simulation can be run in the graphical ISim simulation program or can be run at command line [22]. When run, the root port model keeps a log of the data transmitted and received from the host side. This simulation consisted of:

- The system reset gets de-asserted allowing the integrated Endpoint block to start up and a lock to be achieved with the input clock. Once this is achieved the internal reset signal sourced from the integrated Endpoint block gets de-asserted.
- The integrated Endpoint block starts up allowing data to be sent and received.
- Configuration Read and Writes performed by the host on the integrated Endpoint block to enumerate and configure the device.
- Test data of "01020304" is written to and read back from the device.

A screenshot of the full simulation in ISim where all these processes take place is shown in Figure 4.9. Once the PLL of the integrated Endpoint block achieves a lock with the input clock and starts up, the internal reset signal gets de-asserted. Registers in the Configuration Space then get read and written to, starting with the Device and Vendor IDs being read. An example configuration request is shown in Figure 4.10. As can be seen, the state machine of the user application stays in the reset state for the entire duration. This is as the configuration requests are handled internally by the integrated Endpoint block. After configuration is performed, a sample data read and write is performed. This is done by first writing data of "01020304" to the Endpoint then reading it back. This tests that the unit's transmit, receive and memory elements are working as desired. Screenshots from ISim showing the initial memory write, the memory read and the completion packet sent out are shown in Figures 4.11 to 4.12. These simulations performed gave valuable information on the

enumeration, configuration and Memory Read and Memory Write processes.

4.2.3.2 Implementation on device

The PIO system was programmed to the SP605 board. This was done by loading the configuration into the SPI flash memory. The host system then detected the device on startup once power was applied and the FPGA configured itself. This was tested in the Fedora 10 Linux environment. The detection of the device was confirmed by issuing the `lspci` command, which provides a read out of the connected PCI devices.

As the system verified it detected the SP605, communication with the device was made. In XAPP1022, a device driver and sample application were provided for the generated PIO design. The `xpcie.c` file was compiled to the `xpcie.ko` kernel module. This was then inserted to the device using the `insmod` (Insert Module) command which returned "true" verifying that the driver was added successfully. The Memory Endpoint Test (MET) program was then run. This was a terminal based program which constructed random data, wrote data to the Endpoint and read it back to verify that the data was sent and received successively. The program verified that the transfers were occurring successfully. The user application abstracted away the 1DW memory limitation apparent in PIO transactions. The application allowed memory transactions of over 1DW where each block of data is automatically separated into 1DW chunks with 3DW of header added. Likewise when receiving, data gets automatically compiled together based on how large the data requested was. For example, a Memory Read transaction of 8 DW could be performed, however this would consist of 8 Memory Read packets being sent out, and 8 Completion packets being received.

The PIO design allowed a basic memory communication system to be implemented. However, it showed up the major limitation of using PIO transactions for large memory transfers. The transfer and system utilisation

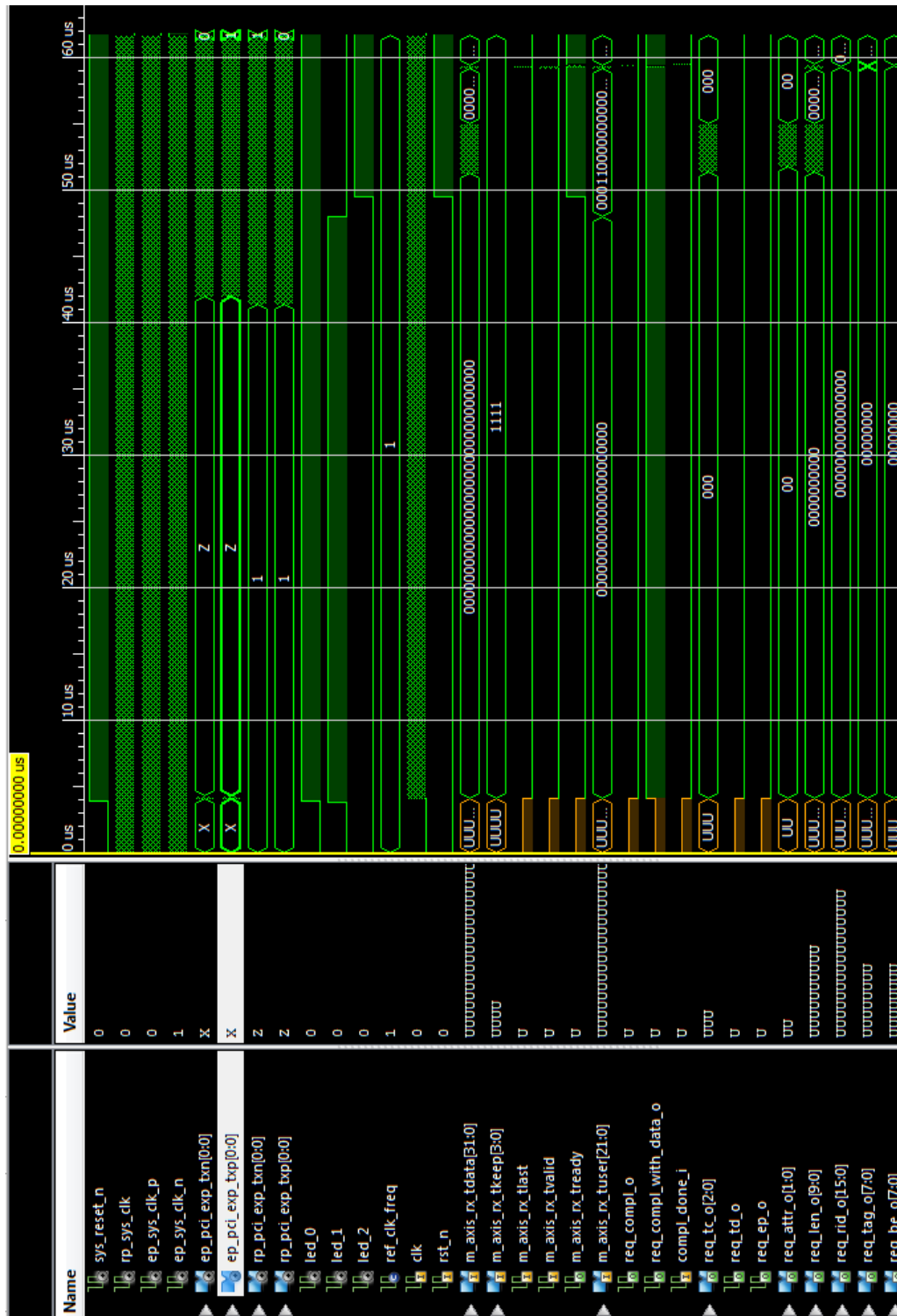


Figure 4.9: Root Port Model Simulation

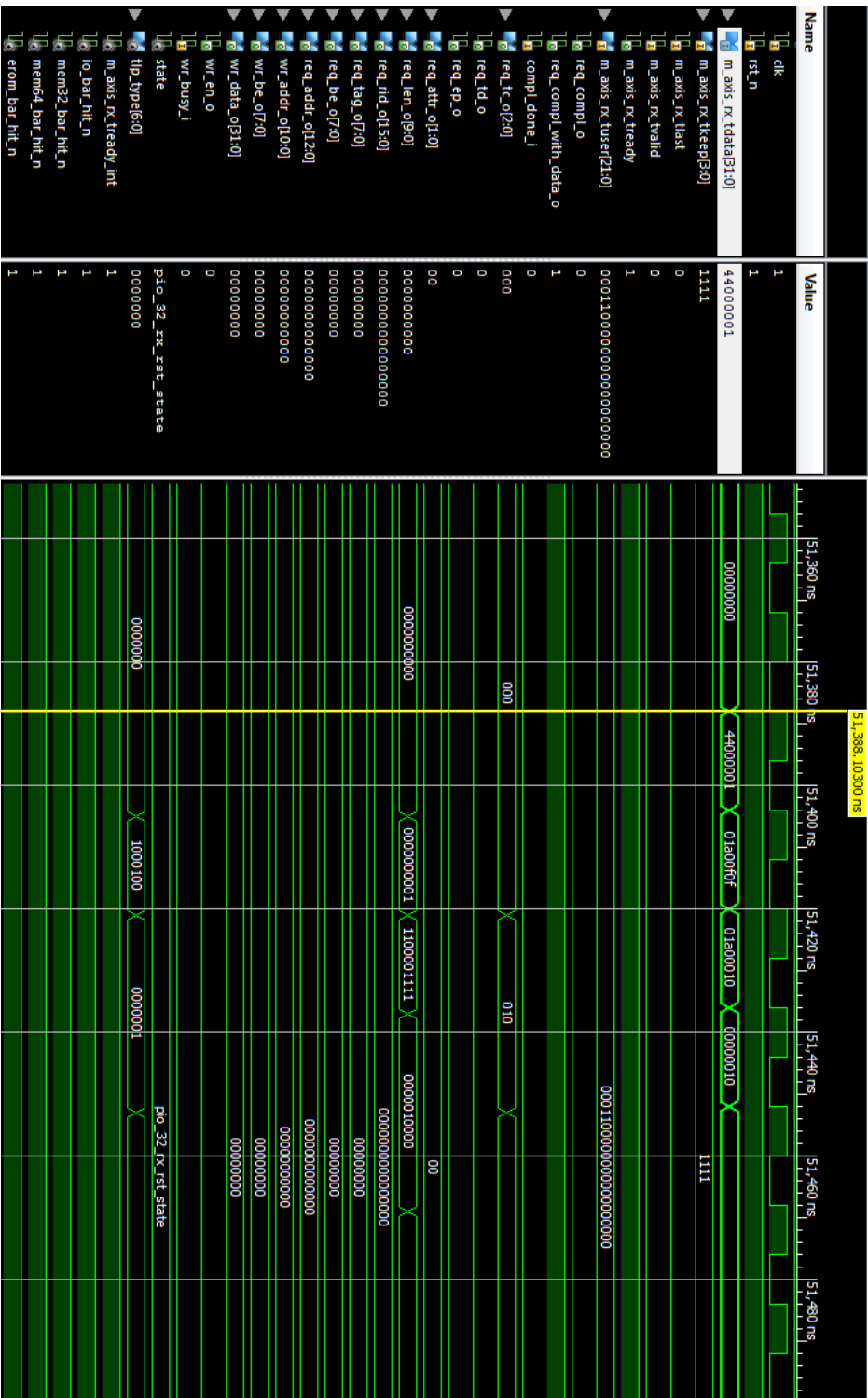


Figure 4.10: Example Configuration Write in ISim

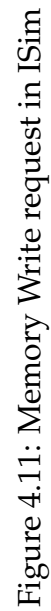


Figure 4.11: Memory Write request in ISim

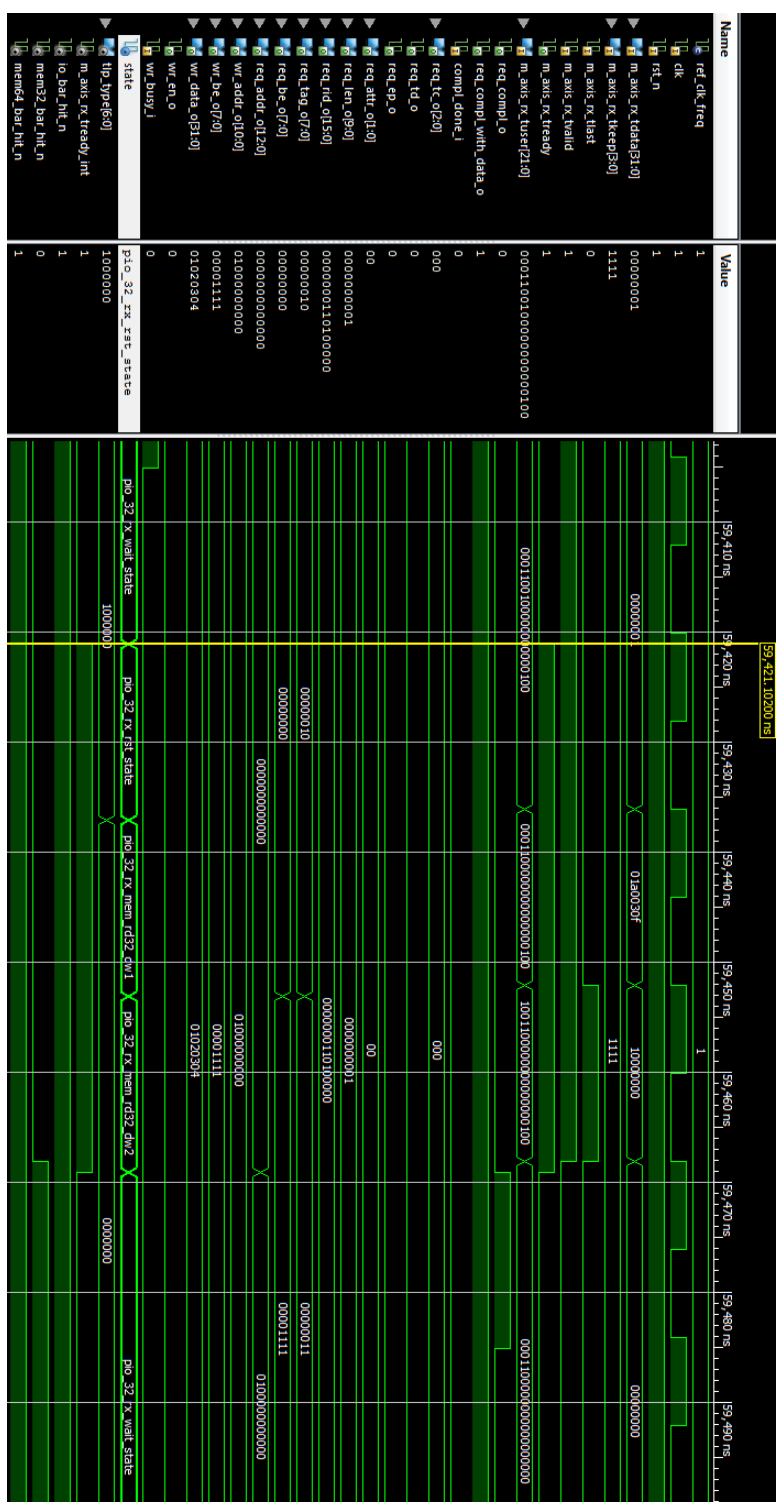


Figure 4.12: Memory Read request in Isim

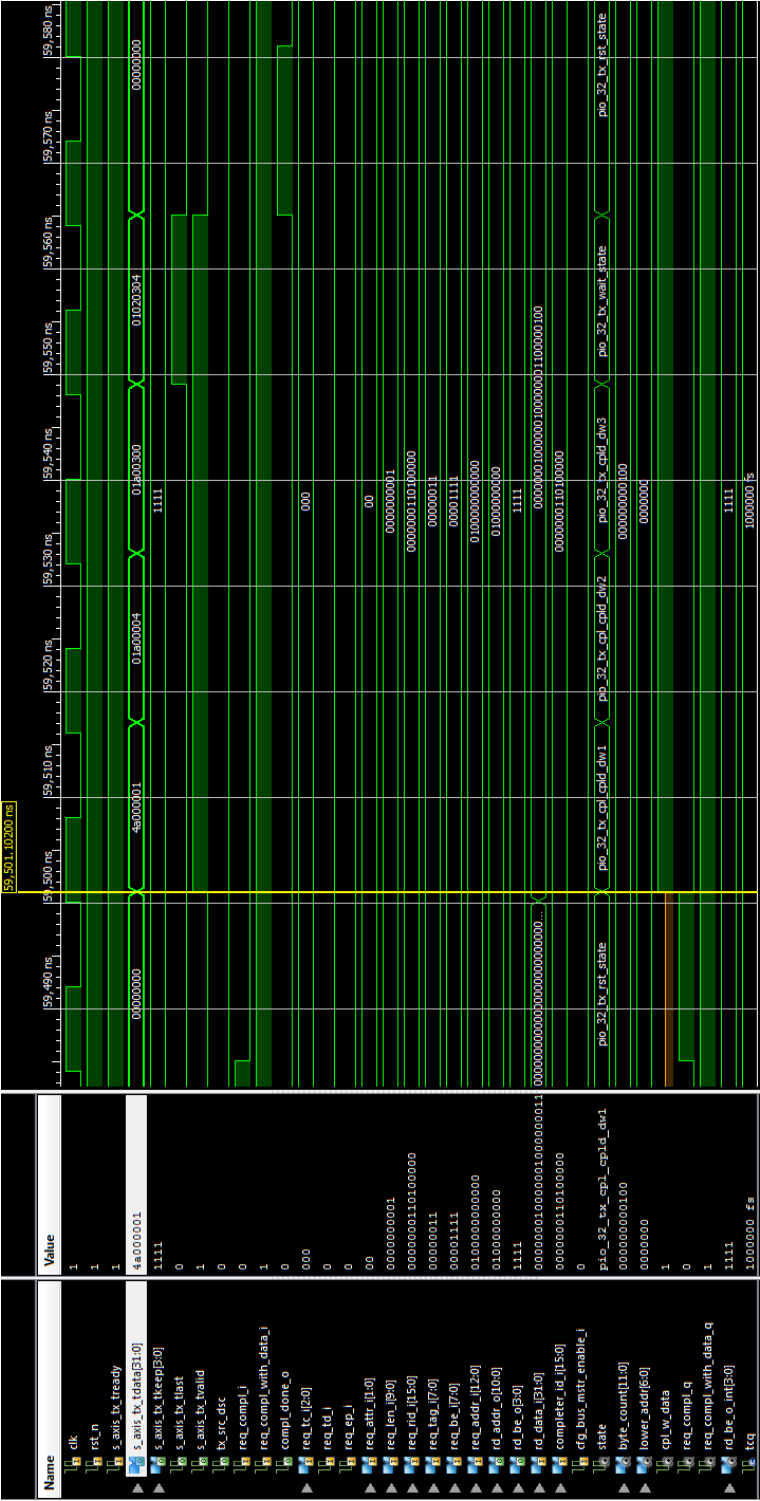


Figure 4.13: Completion packet in ISim

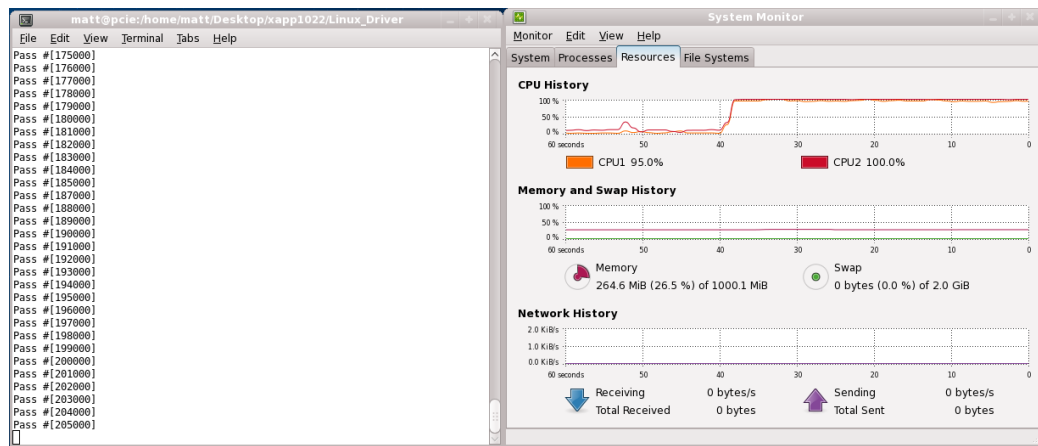


Figure 4.14: CPU Utilisation with PIO transactions occurring

tion are shown in Figure 4.14. As can be seen, when these Memory Read and Writes were occurring the CPU was fully occupied. Also, the effect of the great load was noticed as the CPU fan moved to full speed adding increased noise and heat. Such aspects were very undesirable which would have been noticeable for large data transfers.

The system utilisation of the example PIO design was also compared. Xilinx provides the design in VHDL and Verilog. The system utilisation was thus compared between the two designs. The system utilisation of the VHDL and Verilog systems is shown in Tables 4.1 and 4.2 respectively. Comparing the resources used in the two designs, they are rather similar which is not surprising as the HDL which describe the two designs are effectively equivalent. Subtle differences in the way processes between the two designs are synthesised can explain the differences in resources used.

4.2.4 XAPP1052

The provided Xilinx DMA system was first tested as a comparison point for the FPGA design on the PXIe Peripheral Module. The XAPP1052 application made use of the older version of the integrated Endpoint block

Resource	Number used	Percentage used
Slice registers	394	2%
Slice LUTs	358	5%
Occupied Slices	171	6%
MUXCYs	28	2%
LUT Flip Flop pairs	476	

Table 4.1: FPGA Utilisation for VHDL PIO System

Resource	Number used	Percentage used
Slice registers	406	1%
Slice LUTs	323	1%
Occupied Slices	156	2%
MUXCYs	28	1%
LUT Flip Flop pairs	456	

Table 4.2: FPGA Utilisation for Verilog PIO System

which used the TRN interface. No simulation was provided for this design as was provided with the Root Port Model. However this Root Port Model could be modified to test particular features of the design. The FPGA utilisation was also tested to use as a benchmark against the newly designed system. This is given in Table 4.3. As can be seen, the system utilisation is greater than that of the PIO design shown in Table 4.2. However the overall resources used is not particularly high.

The design was tested in the Fedora 10 Linux environment. Provided

Resource	Number used	Percentage used
Slice registers	917	1%
Slice LUTs	1095	4%
Occupied Slices	396	5%
MUXCYs	292	2%
LUT Flip Flop pairs	1355	

Table 4.3: FPGA Utilisation for Older DMA System

was a Linux device driver and user application. As with the PIO design, the driver source code (the `xbmd.c` file) was compiled to a kernel `.ko` module and inserted into the kernel using the `insmod` command. The user application was then compiled. This application was used to measure the memory read and memory write throughput speeds. These measurements were taken as a benchmark, where this could be compared with a FPGA design for the designed PXIe Peripheral Module. PIO reads and writes to the register space were used to control the system. An example register read is shown below.

```
*** XBMD Register Values ***
DCSR = 0x20021600
DMACR = 0x101
WDMATLPA = 0x22c00000
WDMATLPS = 0x20
WDMATLPC = 0x20
WDMATLPP = 0xfedbeef
RDMATLPP = 0xfedbeef
RDMATLPA = 0x23000000
RDMATLPS = 0x20
RDMATLPC = 0x20
WDMAPERF = 0x465
RDMAPERF = 0x0
RDMASTAT = 0x0
NRDCOMP = 0x0
RCOMPDSIZE = 0x0
DLWSTAT = 0x101
DLTRSSTAT = 0x20002
DMISCCONT = 0x8080001
DLNKC = 0x0
*** End XBMD Register Space ***
```

Tests were run for Memory Read and Memory Write transfers with the

Test	Clock periods	Speed
Memory Write	1125	227 MB/s
Memory Read	1575	162 MB/s

Table 4.4: Performance of TRN system

measurements shown in Table 4.4. Here the performance measurement was read back, converted to decimal and the transmission speed calculated using Equation 4.2.2.

4.2.5 FPGA Design for the PXI Express Peripheral Module

The new FPGA design was intended for use on the reported PXIe Peripheral module. This design was tested first on the SP605 board and then ported to the designed module.

4.2.5.1 Testing on the SP605 board

The project was compiled and programmed to the flash device of the SP605 board as before. The throughput speeds were tested here to compare against the XAPP1052 design using the TRN interface. A screenshot of the user application is given in Figure 4.8.

The FPGA utilisation and performance of the design is shown in Tables 4.5 and 4.6. As can be seen, the utilisation is somewhat higher than that of the XAPP1052 design. However, for the benefits gained by moving to the AXI4-Stream interface and providing the system in VHDL, this was considered acceptable. Also, the utilisation still allowed much more configuration to be added to the device. The measured performance speeds were identical to that of the TRN system. This suggests that the transmission ran uninterrupted in both systems to complete the transmission. Thus no benefit was seen from either system. Thus good utilisation of the PCI Express bandwidth was achieved.

Resource	Number used	Percentage used
Slice registers	1160	1%
Slice LUTs	1417	4%
Occupied Slices	498	7%
MUXCYs	320	2%
LUT Flip Flop pairs	1643	

Table 4.5: FPGA Utilisation for VHDL AXI-4 DMA Design

Test	Clock periods	Speed
Memory Write	1125	228 MB/s
Memory Read	1578	162 MB/s

Table 4.6: Performance of VHDL AXI-4 DMA Design

4.2.5.2 Implementation on designed PXI Express module

With the system verified on the SP605 board, the design was then ported to the designed module. The data transmission side of PXIe makes use of effectively the same signals as PCI Express so there were no differences in this respect. There were some subtle but important differences in the two devices which needed to be factored in. As explained earlier, the SP605 design made use of an on board PLL which steeped the input clock frequency up to 125 MHz. However, on the designed module, the clock from the backplane was simply routed straight into the FPGA as shown in Figure 3.9. Thus the integrated Endpoint block settings had to be configured to accept a 100 MHz input clock to accommodate for this. This is shown in Figure 4.15. Also, the PERST# reset signal ended up not being routed on the designed module. The FPGA design could have been altered to implement some sort of advisory circuit however this was decided as unnecessary as the PERST# signal would be connected on later revisions. As the PERST# signal was not connected, this was removed from the UCF file. Inside the top module this signal was instead tied to '0' as the Endpoint reset required an active high signal. The device can function without the

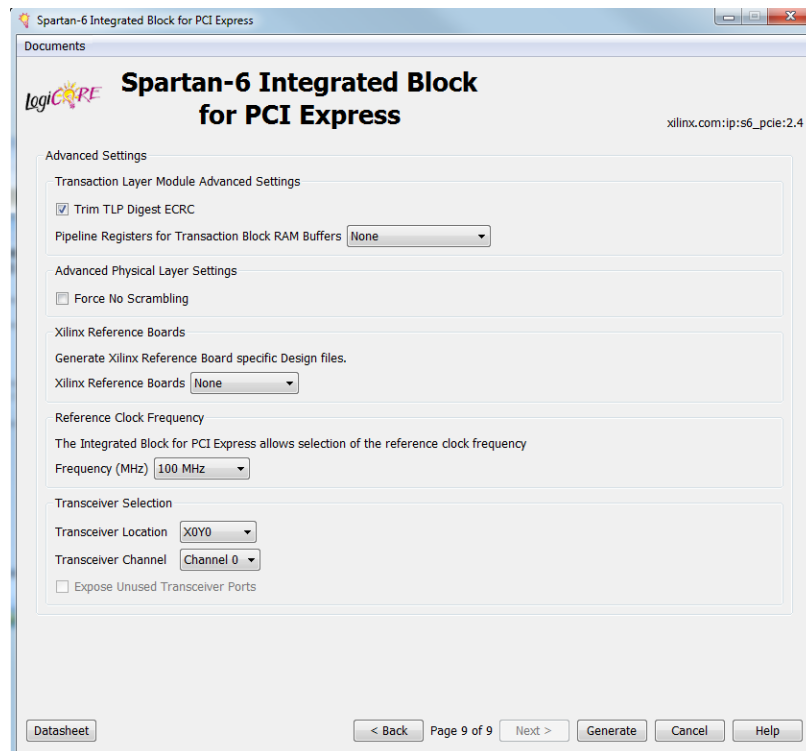


Figure 4.15: Choice of input clock frequency for Spartan-6 FPGA Integrated Endpoint Block for PCI Express

PERST# signal for a few reasons:

- The device would only be configured once the power and clock signals had already been brought up. Thus there would be no issue with the power levels or clock being out of range as the system would already have reached stability.
- If the clock did fall out of range, this would be detected in the integrated Endpoint block as the PLL would be unable to achieve a lock. This would mean the integrated Endpoint block would then trigger a self reset without the requirement of the PERST# signal.

If the power levels fall out of range on the PXIe chassis, PERST# signal should be asserted to the peripheral modules. However, this would not



Figure 4.16: PXI Express system installed with PXI Express Peripheral Module running example application

occur with the signal being unconnected to the FPGA. Due to the high quality of the chassis the card was connected to, this was unlikely to be a problem.

The device was tested by connecting it to the chassis, powering it up, programming the FPGA and then performing a soft reboot of the host system. This meant that power was sustained to the device thus the FPGA configuration would be kept and the device would be detectable on start up. The chassis booted up and detected the device successfully. This was confirmed by performing the `lspci` command in the terminal once the computer had started. The device was detected, so this was followed by installing the device driver and user application. The PXIe system, installed with the designed PXIe Peripheral Module, running the user application is shown in Figure 4.16.

4.2.6 Comparison of designs

In order to fairly test the designs tried, they were all compared when compiled for the SP605 board. Two major factors were evaluated, the throughput of the devices and the resources used on the FPGA. As well as providing a more adaptable system for further modification and integration, performance increases were also found with the new system.

As can be seen, additional resources of the FPGA were used with the VHDL design. This was possibly due to the fact that many more internal signals were required for the VHDL version. However, by improving the design and its efficiency, this could be reduced. The resources used were not overly high however and still allowed for more additions to the design.

The measured performance of the two designs were basically identical. Write speeds were transmitting at near the maximum theoretical bandwidth of the link. This meant that the Endpoint was able to constantly send out data with no interruptions.

Memory Read speeds were substantially lower than write speeds which is to be expected. Memory Read requests are non-posted transactions thus require a completion packet. This adds some latency into the transaction which effects the overall transmission speed. A memory read transaction follows the following process:

- Memory Read request initiated from the Endpoint. This starts the timer measuring the performance of the system and sends a packet with no data requesting data to be read back.
- The host receives the request, retrieves the data and sends back a completion packet with the requested data.
- The Endpoint receives the requested data, and will issue another read request (following the process again) until the full 4KB of data has been read.

- Once the full 4KB of data has been received, the timer measuring read performance stops and issues an interrupt to signal its completion.

This process adds much latency into the transaction. Each request requires a wait for the packet to reach the host and a response to be received back before any data is received back. This is the reason for the major difference in transfer speed between Memory Write requests and Memory Read requests.

Chapter 5

Conclusions and Future Work

Making use of the schematics and footprints provided through the AltiumLive [26] provided an easily implementable board design. This provided a board layout and schematics required to quickly implement a base PXIe system. Though the hole sizes for the footprints for the connectors subtly differed from the size required, this was still usable and would be altered for future designs. The FPGA based design was found to work well with it providing a central location for required signals to be connected to. The low cost Spartan-6 unit was able to provide high throughput speeds with its Integrated Endpoint Block for PCI Express.

The PXIe Peripheral Module FPGA design verified the ability to use existing PCIe solutions for PXIe devices. The FPGA design provided the ability to perform read and write transfers to the host system. The use of the open source DMA system allowed high speed Memory Read and Memory Write transfers to be achieved. This gave measured throughput speeds of 227 MB/s for writes and 162 MB/s for reads. As well as providing this performance, the source code was provided in VHDL making it more suited for further enhancements by others at Magritek. Porting of the system to the VHDL language allowed a greater understanding of the DMA system thus making alterations easier in the future.

The test backplanes for the system were constructed and parts ordered

but were not populated. They are to be used further along as a means of testing newer FPGA designs as well as PXIe Controller cards.

5.1 Further Work

To move to the full PXIe system, a host module and chassis would need to be designed to communicate with the designed PXIe module. The work on the test backplanes for PXIe modules could be used as a starting point for designing a full chassis system. PXIe chassis meet a standard design with the size, board separation and signals required well defined. Thus most of the design work is already completed, it would simply be a task of implementing the described format. As well as the chassis, a host system would also be required. This would consist of developing a computer system which would fit on a host module. The intended processor for such a system would be ARM based, possible the newly released all programmable System on a chip (SoC) Zync-7000 device provided by Xilinx [40]. As well as this RAM, external connectors such as ethernet and USB, some form of non-volatile memory (either flash or a hard disk) would be required on the host system.

The FPGA design and device driver would also be further developed for Magritek's use. The user application was fairly basic with the ability to simply run endpoint speed tests. With some additional work, this could be further developed to integrate well with the intended NMR/MRI use. On board the PXIe Peripheral Module was a block of DDR2 RAM which the device would use when performing transactions. The waveform to be sent out and received back would be stored in this DDR2 RAM. With this configured, the data could be sent over DMA. When loading the waveforms to be sent out, this could be loaded up via a DMA read from the peripheral unit. When sending data back to the host for further processing, this could be performed also by a DMA transfer in the other direction. As well as this, for use in an NMR design, the performance measurement function-

ality would not be required. Thus the registers and processes used for this part of the design could be removed which would improve the efficiency of the design.

The DMA design could be extended to allow Scatter/Gather DMA [41]. The present DMA system requires a block of contiguous or nonsegmented block of physical memory to transmit. However, on most systems it is difficult to get nonsegmented memory returned from the operation system. Thus Scatter/Gather stores the starting addresses of all the memory segments. After a move operation starts the DMA controller automatically adjusts the start address of the next segment after a previous segment of memory is completed. This allows for noncontiguous block of memory to be sent. Recently released open source solutions have been provided which could be evaluated for use on the Spartan-6 system [42, 43].

As it currently stands, DMA transfers are instigated by the host providing an appropriate PIO write command to trigger a DMA transfer. However, this trigger for DMA transfers could be provided from a number of sources. For instance, once a NMR/MRI test has been successfully completed, this could trigger the PCIe controller to send a memory write transfer to the host system.

In order to transfer data greater than 128 bytes of size, the logic to split the payload data would need to be developed. This could simply limit packets at 128 bytes or a more sophisticated system which accounted for the largest payload size allowed could be used. The system would comprise of a unit which would take the data to write and separate it into several chunks so this maximum payload size was not exceeded. This logic could be added to the transmit unit where it would simply check the length of the data and the current payload size and once the maximum was reached, it would create a new TLP.

Bibliography

- [1] J. Monahan. (2012, November) S100 Computers: A Web Site For S-100 Bus Computer Owners. [Online]. Available: <http://www.s100computers.com/index.html>
- [2] (2012) The birth of the IBM PC. IBM. [Online]. Available: http://www-03.ibm.com/ibm/history/exhibits/pc25/pc25_birth.html
- [3] *PCI Local Bus Specification*, PIC-SIG Std., Rev. 3.0, August 2002.
- [4] (2012) Kea dc to 400 mhz nmr spectrometer. Magritek. [Online]. Available: <http://www.magritek.com/products-kea-overview>
- [5] M. Rouse and V. Choodi. (2011, March) VMEbus (VersaModular Eurocard bus). WhatIs.com: The leading IT encyclopedia and learning center. [Online]. Available: <http://whatis.techtarget.com/definition/VMEbus-VersaModular-Eurocard-bus>
- [6] (2010, November) Short Tutorial on VXI. National Instruments. [Online]. Available: <http://www.ni.com/white-paper/2899/en>
- [7] (2010, November) PXI Express Specification Tutorial. National Instruments. [Online]. Available: <http://www.ni.com/white-paper/2876/en>
- [8] *PCI Express Base 1.1 Specification*, PICSIG Std., Rev. 1.1, March 2005.
- [9] *PCI Express Base 3.0 Specification*, PCISIG Std.

- [10] *LXI Device Specification 2011*, LXI Consortium, Inc. Std., Rev. 1.4, May 2011.
- [11] *PXI Express Hardware Specification*, Electronic, PXI Systems Alliance Std., Rev. 1.0, August 2005.
- [12] "AM389x Sitara ARM Processors," Texas Instruments, Data Sheet, July 2012.
- [13] (2012) NI PXIe-1062Q - 8-Slot 3U PXI Express Chassis With AC - Up to 3 GB/s. National Instruments. [Online]. Available: <http://sine.ni.com/nips/cds/view/p/lang/en/nid/202664>
- [14] "XIO1100 Data Manual," Texas Instruments, Data Sheet, August 2011.
- [15] (2012) PCI Express (PCIe): The High-Bandwidth Scalable Solution. Xilinx. [Online]. Available: <http://www.xilinx.com/technology/protocols/pciexpress.htm>
- [16] (2012) PCI Express, PCI, PCIe, PCI-XP, and PCI-SIG Solutions. Altera. [Online]. Available: http://www.altera.com/technology/high_speed/protocols/pci_exp/pro-pci_exp.html
- [17] A. Wilen, J. Schade, and R. Thornburg, *Introduction to PCI Express: A Hardware and Software Developer's Guide*, D. J. Clark, Ed. Intel Press, 2003.
- [18] *CompactPCI Specification*, PCI Industrial Computer Manufacturers Group Std., Rev. 3.0, August 1999.
- [19] *PXI Hardware Specification - PCI eXtensions for Instrumentation: An Implementation of CompactPCI*, Electronic, PXI Systems Alliance Std., Rev. 2.2, September 2004.

- [20] *CompactPCI Express: PICMG EXP.0 R1.0 Specification*, PCI Industrial Computer Manufacturers Group Std., Rev. 1.0, June 2005.
- [21] D. Smith, "VHDL and Verilog compared and contrasted-plus modeled example written in VHDL, Verilog and C," in *Design Automation Conference Proceedings 1996, 33rd*, jun, 1996, pp. 771 –776.
- [22] Xilinx, "Spartan-6 FPGA Integrated Endpoint Block for PCI Express - User Guide," Xilinx, Application Note, January 2012, uG672, V1.2.
- [23] "Spartan-6 Family Overview," Xilinx, Data Sheet, March 2010.
- [24] Spartan-6 FPGA Connectivity Kit. Xilinx. [Online]. Available: <http://www.xilinx.com/products/boards-and-kits/DK-S6-CONN-G.htm>
- [25] Xilinx, "SP605 Schematics," 2009.
- [26] (2012) Altium homepage. Altium. [Online]. Available: <http://live.altium.com/>
- [27] "AXI Reference Guide," Xilinx, User Guide, April 2012.
- [28] (2012) AXI4 IP. Xilinx. [Online]. Available: http://www.xilinx.com/ipcenter/axi4_ip.htm
- [29] J. Ayer, Jr., "Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores," Xilinx, Application Note, November 2009.
- [30] J. Wiltgen and J. Ayer, "Bus Master Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions," Xilinx, Application Note, September 2011.
- [31] (2012) PCI Express Solution. Northwest Logic. [Online]. Available: <http://nwlogic.com/products/pci-express-solution/>

- [32] (2012) PCIe with DMA (EZDMA). PLDA. [Online]. Available: <http://www.plda.com/prodetail.php?pid=156>
- [33] V. A. Pedroni, *Circuit Design and Simulation with VHDL*. The MIT Press, 2010.
- [34] C. E. Cummings, "Coding and Scripting Techniques for FSM Designs with Synthesis-Optimized, Glitch-Free Outputs," in *SNUG Boston*. Sunburst Design, Inc., 2000.
- [35] (2012) Virtex-6 FPGA Connectivity Kit. Xilinx. [Online]. Available: <http://www.xilinx.com/products/boards-and-kits/EK-V6-CONN-G.htm>
- [36] "AN10373 - PCI Express PHY PCB Layout Guideline," Philips Semiconductors, Tech. Rep., April 2005.
- [37] Z. Schoenborn, "Board Design Guidelines for PCI Express Architecture," PCISIG, Conference Presentation, 2004.
- [38] (2012) Fedora Homepage. Fedora. [Online]. Available: <http://fedoraproject.org/>
- [39] (2012) Intel 945G Express Chipset. Intel. [Online]. Available: <http://ark.intel.com/products/27720/Intel-82945G-Memory-Controller>
- [40] "Zynq-7000 All Programmable SoC Overview," Xilinx, Advance Product Specification, August 2012.
- [41] (2009, February) What is Scatter-Gather DMA (Direct Memory Access)? National Instruments. [Online]. Available: <http://digital.ni.com/public.nsf/allkb/65B0708FE161D8C0852563DA00620887>
- [42] W. Gao, Z. Han, and S. Sabatini. (2012, March) PCIe SG DMA controller :: Overview. OpenCores. [Online]. Available: http://opencores.org/project,pcie_sg_dma,overview

- [43] D. Smekhov. (2012, October) PCIe_DS_DMA :: Overview. OpenCores.
[Online]. Available: <http://opencores.org/project,pcie.ds.dma>