TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

Diploma Thesis

PCI Express based Embedded System

System Specification, Design, Simulation and Implementation

realized to attain the academic degree of "Diplom-Ingenieur" under the supervision of

o.Univ.- Prof. Dr. Dietmar Dietrich
and
Dipl.-Ing. Herbert Nachtnebel
as the supervising assistant

applied at the
Technical University of Vienna
Department of Electrical Engineering
Institute of Computer Technology (E384)

in cooperation with
SIEMENS AG Austria
Department of Program and System Engineering (PSE),
Chip, Electronic and Software (CES) *under the supervision of*
Dipl.-Ing. Majid Ghameshlu

by
Faraj Nassar
0326747
Leopoldauer Platz 19
1210 Vienna

Vienna, October 2007

To My Family

Abstract

The platform of Today's PC consists of many local buses with different requirements, to allow the communication of different devices with each other. Nowadays, many of these modern electronic devices are demanding a high bandwidth, even higher than what already existing input and output (IO) bus systems can deliver, of most interest is the Peripheral Component Interconnect (PCI) bus. These bus systems are reaching their practical limits and are facing serious problems and shortcomings that prevent them from being able to provide the bandwidth and features needed by the electronic industry, which keeps needing to an increased bandwidth as well as to a simple electrical connectivity.

All these factors together have motivated the engineering of a new IO bus system, the so-called Peripheral Component Interconnect Express (PCIe), which has been adopted as a general purpose IO device interconnect in different applications, such as desktop, server, mobile, workstations, computing and communication platforms.

Within this diploma work, the theory of PCIe was summarized and presented in monthly-based presentations (PCIe tutorial). Some of the available PCI Express Intellectual Property (IP) solutions in the market were studied and compared.

In addition to that, a PCIe-based embedded data communication system was specified, designed, simulated, and synthesized. This system utilizes the Xilinx Microblaze soft processor core, the Xilinx PCIe core, and the Philips PX1011A physical layer.

Data communication between the designed PCIe-based intelligent Endpoint device (in the PCIe topology) and the system memory, as well as the Central Processing Unit (CPU), through the Root complex, was simulated.

Keywords: IP, Microblaze, PCI, PCIe Core, PCIe Endpoint, On Chip Peripheral Bus (OPB), OPB IPIF, OPB to PCIe Bridge, Philips PX1011A PHY, USER LOGIC.

Kurzfassung

Die Architekturen heutiger PCs bestehen aus vielen Bussystemen mit unterschiedlichen Anforderungen, welche die Kommunikation der unterschiedlichen Geräte miteinander erlauben. Heutzutage verlangen viele dieser modernen elektronischen Geräte eine hohe Bandbreite, oft höher als es die verfügbaren Eingangs- und Ausgangsbussysteme erlauben. Die größte Bedeutung hat die Peripheral Component Interconnect (PCI) Busfamilie, aber auch diese Bussysteme erreichen ihre praktischen Grenzen und beinhalten ernsthafte Probleme und Mängel, welche verhindern, dass die, durch die elektronische Industrie geforderte, immer höhere Bandbreite, und die einfache elektrische Anbindung erreicht werden.

Alle diese Faktoren zusammen haben die Entwicklung eines neuen IO Bussystems motiviert, dem sogenannten Peripheral Component Interconnect Express (PCIe) Bus, welcher als universelles Eingangs- und Ausgangsbussystem in den unterschiedlichsten Anwendungen, wie Desktopcomputer, mobile Endgeräten, Workstations, sowie Rechen- und Kommunikationsplattformen eingesetzt wird.

Im Rahmen dieser Diplomarbeit wurde die Theorie von PCIe zusammengefasst und in monatlichen Präsentationen vorgestellt (PCIe Tutorial). Einige der vorhandenen PCIe Intellectual Property (IP) Lösungen im Markt wurden untersucht und verglichen.

Zusätzlich wurde ein PCIe basiertes embedded Datenübermittlungssystem spezifiziert, entworfen, simuliert und synthetisiert. Dieses System verwendet den Xilinx Microblaze Processor Core, den Xilinx PCIe Core und die Philips PX1011A physikalische Ebene.

Außerdem wurde die Datenkommunikation zwischen dem entworfenen PCIe basierten intelligenten Endpunkt-Gerät (in der PCIe Topologie) und dem Systemspeicher, sowie der Zentraleinheit (CPU), durch den Verbindungsblock, simuliert.

Stichwörter: IP, Microblaze, PCI, PCIe Core, PCIe Endpunkt, On-chip Peripheral Bus (OPB), OPB IPIF, OPB zur PCIe Brücke, Philips PX1011A PHY, USER LOGIK.

Acknowledgment

This PCI Express based Embedded System was conducted as a Master thesis work in the field of Microelectronics, in cooperation with the Institute of Computer Engineering at the Technical University of Vienna and the program and System Engineering (PSE) Department, Chips, Electronics and Software (CES) Division of SIEMENS AG Austria.

First of all, I would like to express my deepest thanks to Mr. Johann Notbauer (M.Sc.), for giving me the opportunity to carry out this project in his department at SIEMENS, under the direct supervision of Mr. Majid Ghameshlu (M.Sc.), who consistently helped me do my best, due to his wide experience in this field. Many thanks to Mr. Ghameshlu for being patient and helpful all the time.

I am also grateful for the support and help provided by my colleagues at SIEMENS AG Austria.

I also want to express my appreciation to Mr. Herbert Nachtnebel (M.Sc.) for being my direct supervisor at the Technical University of Vienna, and for his support and contribution to this work.

In addition, I want to thank Prof. Dietmar Dietrich, the Head of the Computer Engineering Institute at the Technical University of Vienna, and Prof. Gottfried Strasser, professor at the Institute of Solid State Electronics and the Technical Director of the Microstructure center of the Technical University of Vienna, for being my examiners.

Finally, I want to thank my family and friends for their constant encouragement and support.

Contents

1 Introduction 1

1.1 Objectives 1
1.2 Method, Software and Hardware 1
1.3 Tasks and Time Plan 2
1.4 Outline 3

2 PCI Express Theory 4

2.1 Evolution of IO Bus Systems 4
2.2 Peripheral Component Interconnect (PCI) 5
2.2.1 PCI Architecture 5
2.2.2 PCI Key Features 7
2.2.3 PCI practical Limitations and challenges 7
2.3 Peripheral Component Interconnect Express (PCIe) 8
2.3.1 PCIe Introduction 8
2.3.2 PCIe Topology 9
2.3.3 PCIe Key Features 10
2.3.4 PCIe Architecture 10
2.3.5 PCIe Future Prospective 16

3 PCIe Endpoint (EP) Design 17

3.1 Design Overview 17
3.2 Philips PX1011A PHY 22
3.2.1 Block Diagram 22
3.2.2 Operation Principle 22
3.2.3 Interfaces 23
3.3 Xilinx PCIe Core 26
3.3.1 Functionality and Features 27
3.3.2 Block Diagram and Functionality 28
3.3.3 Core Interfaces 29
3.3.4 Core Generation and Configuration 39
3.4 Xilinx Microblaze Soft Processor Core 42
3.4.1 Microblaze Block Diagram 42
3.4.2 Microblaze Interfaces 43
3.4.3 Local Memory Bus (LMB) 44
3.4.4 On-Chip Peripheral Bus (OPB) 44

3.5 Microblaze PCIe Peripheral 46
3.5.1 OPB to PCIe Bridge 47
3.5.2 On-chip Peripheral Bus Intellectual Property Interface (OPB IPIF) 51
3.5.3 USER LOGIC Model 53
3.5.3.1 Register Read 56
3.5.3.1 Register Write 56
3.5.3.3 Software accessible Register Bank 57
3.5.3.4 PCIe Transmission State Machine 65
3.5.3.5 PCIe Receiving State Machine 70
3.5.3.6 PCIe Configuration space Access READ/WRITE State Machine 735

4 PCIe Endpoint Simulations 80
4.1 PCIe Testbench 80
4.1.1 Philips PHY Simulation Model 81
4.1.2 Xilinx PCIe Downstream Port Simulation Model 81
4.1.3 Design Under Test (DUT) 83
4.1.3.1 Xilinx PCIe PIPE IP Simulation Model 83
4.1.3.2 Microblaze based System Simulation Model 84
4.2 C Application Program 84
4.3 Simulation Flow 94
4.4 Test Cases Summary 104

5 PCIe Endpoint Implementation 106

6 Conclusion and Future Work 107

Appendix A: PCI Express IP Providers 108

Appendix B: Xilinx WebCases 121

Appendix C: Project Directory Structure 122

Bibliography 124

List of Acronyms and Abbreviations

A

ACK/NAK	Acknowledged /Not Acknowledged
AGP	Accelerated Graphics Port
ASIC	Application Specific Integrated Circuits
ASPM	Active State Power Management
Attr	Attribute

B

B	Byte
BAR	Base Address Register
BE	Byte Enable
BIOS	Basic Input Output System
BRAM	Block Random Access Memory

C

CA	Completer Abort
CES	Chips, Electronics and Software
CFG	Configuration interface
CMM	Configuration Management Module
CPl	Completion without data
CPLD	Completion with Data
CPU	Central Processing Unit
CRS	Configuration Request Retry Status

D

DDR	Dual Data Rate
DLLP	Data Link Layer Packet
DLMB	Data Local Memory Bus
DMA	Direct Memory Access
DUT	Design Under Test
DW, DWORD	Doubleword

E

ECRC	End to End Cyclic Redundancy Check
EDK	Embedded Development Kit
EISA	Extended Industry Standards Architecture
ELF	Executable linked Format
EOF	End-of-Frame
EP	Endpoint

F	
FIFO	First In First Out
Fmt	Format
FPGA	Field Programmable Gate Arrays
FSB	Front Side Bus
FSL	Fast Simplex Link
G	
Gbytes/s	Gigabytes per second
Gbps	Giga bit per second
H	
HDD	Hard Disk Drive
HDL	Hardware Description Language
HDR	Header
I	
ICH	IO Controller Hub
IDE	Integrated Drive Electronics
ILMB	Instruction Local Memory Bus
IO	Input and Output
IP	Intellectual Property
IPIC	IP Interconnect
IPIF	Intellectual Property Interface
ISA	Industry Standards Architecture
ISE	Integrated Software Environment
K	
Kbytes	2^{10} Bytes
L	
LCRC	Link Cyclic Redundancy Check
LLM	Data Link Layer Module
LMB	Local Memory Bus
LTSSM	Link Training and Status State Machine
LVDS	Low Voltage Differential Signal
M	
M.Sc.	Master of Science
MAC	Media Access Controller
Mbytes/s	Megabytes per second
MCH	Memory Controller Hub
MGTs	Multi-Gigabit Transceivers
MSI	Message Signalled Interrupt
O	
OPB	On-Chip Peripheral Bus
OPB IPIF	OPB Intellectual Property Interface

P	
PCI	Peripheral Component Interconnect
PCI-SIG	PCI-Special Interest Group
PCI-X	Peripheral Component Interconnect-X
PCIe	Peripheral Component Interconnect Express
PHY	Physical
PIPE	Physical Interface for PCI Express
PLD	Programmable Logic Device
PLI	Programming Language Interface
PLM	Physical Layer Module
PLPs	Physical Layer Packets
PPM	Programmed Power Management
PXPIPE	Philips PHY Specification PIPE
Q	
QDR	Quad Data Rate
QoS	Quality of services
R	
R	Reserved
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
RX	Receiver
S	
SC	Successful Completion
SCSI	Small Computer System Interface
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SOF	Start-of-Frame
SYS	System interface
T	
TC	Traffic Class
TD	TLP Digest
TLP	Transaction Layer Packet
TLM	Transaction Layer Module
TPI	Test Programming Interface
TRN	Transaction interface
TXPLL	Transmitter Phase Locked Loop
TX	Transmitter
U	
UR	Unsupported Request
USB	Universal Serial Bus

V

VESA	Video Electronics Standards Association
VHDL	VHSIC-HDL Very High Speed Integrated Circuit Hardware description Language

List of Figures

Chapter 1 Introduction 1

Chapter 2 PCI Express Theory 4

Figure 2.1 - *Evolution of IO bus systems 4*
Figure 2.2 - *An Example of 33 MHz PCI Bus Based System [ABS04] 6*
Figure 2.3 - *An Example of 66 MHz PCI Bus Based System [ABS04] 6*
Figure 2.4 - *PCI Express Link [ABS04] 8*
Figure 2.5 - *PCI Express Topology 9*
Figure 2.6 - *PCI Configuration Model [ABS04] 11*
Figure 2.7 - *PCI Express Architecture 12*
Figure 2.8 - *PCI Express TLP Assembly/Disassembly 12*
Figure 2.9 - *Header for a 32-bit Memory Write TLP 13*

Chapter 3 PCIe Endpoint Design 17

Figure 3.1 - *Basic Memory Transactions 17*
Figure 3.2 - *Endpoint Design 19*
Figure 3.3 - *Xilinx Virtex-5 LXT PCI Express Endpoint Block [UG197] 19*
Figure 3.4 - *Spartan-3 PCI Express Starter Kit [Xilinx] 20*
Figure 3.5 - *Complete PCIe Endpoint Device 21*
Figure 3.6 - *Block Diagram of PX1011A [GL05] 23*
Figure 3.7 - *PCIe Differential Transmitter/Receiver [ABS04] 24*
Figure 3.8 - *PIPE Interface [PPHY] 25*
Figure 3.9 - *PXPIPE Interface [PPHY] 25*
Figure 3.10 - *Two-chip Solution [XP05] 27*
Figure 3.11 - *Top-level Functional Blocks Diagram and Interfaces of Xilinx PCIe Core [XUG167] 28*
Figure 3.12 - *PCIe Configuration Space [XUG167] 30*
Figure 3.13 - *PCIe Component Name [XUG167] 39*
Figure 3.14 - *PCIe ID Initial Values [XUG167] 39*
Figure 3.15 - *PCIe Base Address Registers (BARs) Configuration 40*
Figure 3.16 - *PCIe Capabilities and Device Capabilities Register Configuration [XUG167] 41*
Figure 3.17 - *PCIe Link Capabilities Register Configuration [XUG167] 41*
Figure 3.18 - *PCIe Advanced Settings [XUG167] 42*
Figure 3.19 - *Microblaze Block Diagram [XUG081] 43*
Figure 3.20 - *Local Memory Bus (LMB) 44*

Figure 3.21 *- OPB Slave Attachment [XTZ02] 45*
Figure 3.22 *- PCIe Peripheral Attachment as Slave [XTU02] 46*
Figure 3.23 *- PCIe Peripheral 47*
Figure 3.24 *- OPB to PCIe Bridge 47*
Figure 3.25 *- OPB to PCIe Bridge Interfaces/Transaction Interfaces 48*
Figure 3.26 *- OPB to PCIe Bridge Interfaces/Configuration Interfaces 49*
Figure 3.27 *- Basic OPB Read Transaction [XTU02] 50*
Figure 3.28 *- Basic OPB Write Transaction [XTU02] 51*
Figure 3.29 *- Full Set of OPB IPIF Features [XTU02] 52*
Figure 3.30 *- OPB IPIF Features for Register Access [XTU02] 52*
Figure 3.31 *- OPB IPIF Top-level Block Diagram, Register Interface Only [XDS414] 53*
Figure 3.32 *- USER LOGIC Interfaces 54*
Figure 3.33 *- Read Transaction from IP that utilizes Register Decodes [XDS414] 55*
Figure 3.34 *- Write transaction to IP that utilizes Register Decodes [XDS414] 56*
Figure 3.35 *- USER LOGIC Internal structure 57*
Figure 3.36 *- Register Bank, Base Address and Offset are in Hexadecimal 58*
Figure 3.37 *- Registers Big-Endian Format [XUG081] 58*
Figure 3.38 *- STATUS Register 58*
Figure 3.39 *- CONTROL Register 59*
Figure 3.40 *- MWR/MRD HDR DW1 Register 60*
Figure 3.41 *- MWR/MRD HDR DW2 Register 62*
Figure 3.42 *- MWR/MRD HDR DW3 Register 62*
Figure 3.43 *- REC. MWR/MRD/CPLD DW1 Register 63*
Figure 3.44 *- REC. CPLD DW2 Register 63*
Figure 3.45 *- REC. CPLD DW3 Register 64*
Figure 3.46 *- REC. MWR/MRD DW2 Register 64*
Figure 3.47 *- PCIe Transmission State Machine 66*
Figure 3.48 *- PCIe Transmission State Machine State Diagram 68*
Figure 3.49 *- Memory Write TLP with a 3 DW Header and Payload [XUG167] 69*
Figure 3.50 *- Memory Read TLP with a 3 DW Header without Payload [XUG167] 70*
Figure 3.51 *- PCIe Receiving State Machine 71*
Figure 3.52 *- PCIe Receiving State Machine State Diagram 73*
Figure 3.53 *- Received 32-bit Addressable Memory Write TLP [XUG167] 74*
Figure 3.54 *- Received 32-bit Addressable Memory Read TLP [XUG167] 74*
Figure 3.55 *- PCIe Configuration Space Access READ/WRITE State Machine 76*
Figure 3.56 *- PCIe Configuration Space Access READ/WRITE State Machine State Diagram 78*
Figure 3.57 *- Reading of PCIe Configuration Space [XUG167] 79*

Chapter 4 PCIe Endpoint Simulation 80

Figure 4.1 *- PCIe Testbench Top-level* 80
Figure 4.2 *- PX1011A Packaged Model [PUG05]* 81
Figure 4.3 *- PCIe Downstream Port Model 82*
Figure 4.4 *- Functional Block Diagram of the PCIe Downstream Port Model 82*
Figure 4.5 *- Top level of DUT Model 83*

Figure 4.6 - *Top-level of Microblaze Based System Simulation Model 84*
Figure 4.7 - *Segment 1: Initialization and Configuration of the PCIe Core 85*
Figure 4.8 - *Segment 2: PCIe Core Configuration Space Read 86*
Figure 4.9 - *Segment 3: PCIe Core Configuration Space Write 86*
Figure 4.10 - *Segment 4: Bus Master Enable 87*
Figure 4.11 - *Segment 5: Generating of a Memory Write TLP 88*
Figure 4.12 - *Segment 6: Generating of a Memory Read TLP 89*
Figure 4.13 - *Segment 7: Receiving of a CPLD 90*
Figure 4.14 - *Segment 8: Receiving of a Memory Write TLP 91*
Figure 4.15 - *Segment 9: Receiving of a Memory Read TLP 92*
Figure 4.16 - *Segment 10: Generation of a CPLD TLP 93*
Figure 4.17 - *Simulation Flow Stage 1 94*
Figure 4.18 - *Symbol Lock/PXPIPE Waveforms 95*
Figure 4.19 - *PCIe Downstream Port Waveforms 96*
Figure 4.20 - *Simulation Flow Stage 1 (Continued) 96*
Figure 4.21 - *Setting of PCIe Configuration Space 97*
Figure 4.22 - *Transmission Output Logging 98*
Figure 4.23 - *Reception Output Logging 98*
Figure 4.24 - *Transmitting Output Logging/Writing BAR0 Starting Address 99*
Figure 4.25 - *Waveforms of Configuration Write and CPL TLPs 99*
Figure 4.26 - *Simulation Flow Stage 2 100*
Figure 4.27 - *PCIe Core Configuration Space Access Waveforms 100*
Figure 4.28 - *Simulation Flow Stage 3 101*
Figure 4.29 - *Simulation Flow Stage 3 Waveforms 102*
Figure 4.30 - *Simulation Flow Stage 4 Waveforms 102*
Figure 4.31 - *Simulation Flow Stage 4 103*
Figure 4.32 - *Test Cases 1 and 2 104*

5 PCIe Endpoint Implementation 106

6 Conclusion and Future Work 107

Appendix A: PCIe IP Providers 108

Figure A.1 – *PCIe IP Providers 108*

Appendix B: Xilinx WebCases 121

Appendix C: Project Directory Structure 122

Figure C.1 - *Project Directory Structure 122*
Figure C.1 - *Project Directory Structure (Cont.) 123*

Bibliography 124

List of Tables

Chapter 1 Introduction 1

Chapter 2 PCI Express Theory 4

Chapter 3 PCIe Endpoint Design 17

Table 3.1 **-** *PHY Interfaces to the PCIe Link 24*
Table 3.2 **-** *Philips PHY Interfaces to the Xilinx PCIe Core [KPE05] 26*
Table 3.3 **-** *PCIe Core Specification [XUG167] 27*
Table 3.4 **-** *PXPIPE Transmit Data Interface Signals [XUG167] 31*
Table 3.5 **-** *PXPIPE Receive Data Interface Signals [XUG167] 31*
Table 3.6 **-** *Clock and Reference Signals [XUG167] 31*
Table 3.7 **-** *PXPIPE Command Interface Signals [XUG167] 32*
Table 3.8 **-** *PXPIPE Status Interface Signals [XUG167] 32*
Table 3.9 **-** *Xilinx PCIe PIPE Endpoint Core Transaction Receive Interfaces [XUG167] 34*
Table 3.9 **-** *Xilinx PCIe PIPE Endpoint core Transaction Receive Interfaces (Cont.) [XUG167] 35*
Table 3.10 **-** *Xilinx PCIe Core Configuration Interfaces [XUG167] 36*
Table 3.10 **-** *Xilinx PCIe PIPE Endpoint Core Configuration Interfaces (Cont.) [XUG167] 37*
Table 3.10 **-** *Xilinx PCIe PIPE Endpoint Core Configuration Interfaces (Cont.) [XUG167] 38*
Table 3.11 **-** *OPB Global Signals 45*
Table 3.12 **-** *OPB Interface Signals 45*
Table 3.12 **-** *OPB Interface Signals (Cont.) 46*
Table 3.13 **-** *PCIe Transmission State Machine Transition Table 67*
Table 3.14 **-** *PCIe Receiving State Machine Transition Table 72*
Table 3.15 **-** *Registers Mapped Directly onto the Configuration Interface of the Core 75*
Table 3.16 **-** *PCIe Configuration Space Access READ/WRITE State Machine Transition Table 77*

Chapter 4 PCIe Endpoint Simulation 80

5 PCIe Endpoint Implementation 106

Table 5.1 **-** *Device Utilization 106*

6 Conclusion and Future Work 107

Appendix A: PCIe IP Providers 108

Table A.1 *- Features of Physical Layer and Data Link Layer 110*
Table A.1 *- Features of Physical Layer and Data Link Layer (Cont.) 111*
Table A.1 *- Features of Physical Layer and Data Link Layer (Cont.) 112*
Table A.2 *- Features of the Transaction Layer 113*
Table A.2 *- Features of the Transaction Layer (Cont.) 114*
Table A.2 *- Features of the Transaction Layer (Cont.) 115*
Table A.2 *- Features of the Transaction Layer (Cont.) 116*
Table A.3 *- General Key Features 117*
Table A.3 *- General Key Features (Cont.) 118*
Table A.3 *- General Key Features (Cont.) 119*
Table A.3 *- General Key Features (Cont.) 120*

Appendix B: Xilinx WebCases 121

Table B.1 *- WebCase Summary 121*

Appendix C: Project Directory Structure 122

Table C.1 *- Project Directory Structure 122*

Bibliography 124

1 Introduction

1.1 Objectives

The main purpose of this diploma work is to demonstrate the capabilities of the third generation IO Interconnect bus system, the so-called PCI Express. To achieve this purpose, two sub-objectives are aimed to:

- Preparing a kind of PCIe tutorial (PowerPoint- Presentation) for the fast entry in the PCI Express technology. These presentations provide the Know-How required for someone to use this technology for the fist time. In addition, some of the available solutions in the market for the implementation of PCI Express are to be studied, discussed and compared.

- Designing of a PCI Express-based embedded system for customer reference. In this system an intelligent Endpoint device, employing this technology, should be able to write a double word (DW = 32 bits) to a location within the system memory and read this data back. This system should also enable data communication between the CPU through the Root Complex and this Endpoint device.

1.2 Method, Software and Hardware

PCI Express theory has been acquired through an extensive reading of two reference books. Namely, the PCI Express System Architecture and PCI System Architecture by MindShare, Incorporated. The PCI Express Base Specification v1.0 – 2002 and others were additional valuable references as well.

XILINX, which leads the Programmable Logic Device (PLD) market, one of the fastest growing segments of the semiconductor industry, was the source of most of the used Intellectual Property (IP) solutions, which are functions designed for the implementation in the Field Programmable Gate Array (FPGA) devices. Throughout this documentation the Intellectual Property will be referred to as *IP*. Xilinx provides many of these ready optimized and compiled IP solutions. Xilinx accompanies these with data sheets, manuals and detailed descriptions, which provide a paramount help. For the use of the available development tools, Xilinx reports several tutorials and demos, which were extensively used to assist the designing of the embedded system. Xilinx also gives the opportunity for consultancy and technical supporting, through what it calls webcases. During this diploma work, several webcases were opened, discussed, and successfully solved with engineers from this company.

Xilinx PCI Express Physical Interface for PCI Express (PIPE) Endpoint 1-Lane IP core was used to implement the protocol layers of the PCI Express architecture. In the remainder of this documentation, this core will be referred to as *PCIe core*.

An evaluation licence of this core was received in a package, along with the Spartan-3 PCI Express Starter Kit. On which the designed system is to be implemented. The core was generated, configured and customized using the Xilinx CORE generator.

The Xilinx Spartan-3 FPGA and Philips PX1011A PHY demonstrate a two-chip solution for designing such a system. The Microblaze processor soft core IP was embedded in the Endpoint, to make it an intelligent device. The Microblaze based embedded system and the PCIe PIPE core, are to be implemented in the Spartan-3 FPGA.

The XPS (Xilinx Platform Studio), a part of Xilinx EDK (Embedded Development Kit) 8.2i, and the SDK (Software Development Kit) were used to design the Microblaze based system.

PX1011A PHY is a discrete chip used to implement the physical layer of the PCIe protocol. For the simulation, a behaviour model of this chip was received as a packaged model from NXP Semiconductors. This model interfaces the simulation tool using the Verilog HDL Programming Language Interface (PLI).

Xilinx provides a complete PCIe simulation testbench. In a customized version of this testbench and with the help of the simulation tool ModelSim SE, the whole system was simulated.

Synplify Pro 8.1 and ISE (Integrated Software Environment) 8.2i were used to synthesize the PCI Express based Embedded System.

ISE 8.2i was used to prepare the implementation of the design in the Xilinx programmable logic device.

1.3 Tasks and Time Plan

The tasks carried out were divided into two parts: a theoretical part regarding the theory of PCI Express and the preparing of the power point presentations, and a practical part regarding the implementation of the data transfer system. These two main tasks were further divided into the following subtasks:

- PCI Express theory: reading, studying and researching
- Preparation of PCIe tutorial and presentations to the chip design team at Siemens.
- Overview of the different available PCIe IP solutions in the market, studying and comparing.
- Specification of a microprocessor system (Microblaze) with PCIe links.
- Implementation of the system in Register Transfer Level (RTL) using Very High Speed Integrated Circuit Hardware description Language (VHSIC-HDL or VHDL).
- Software development for the data transfer.

- Simulation and functionality verification.
- FPGA prototyping including measurements (optional).
- Documentation of the work, including an experience report.

This project was accomplished in four phases, over duration of 8 months.

1.4 Outline

After the brief introduction in chapter one, chapter two will summarize the most important aspects of the PCI Express bus system. It starts with a short introduction to the evolution of IO bus systems. In this chapter, the PCI bus architecture, its key features, practical limitations, challenges and shortcomings are discussed. Furthermore, the PCIe bus system is introduced. Its topology and architecture are then discussed. The functionality of each layer in the PCIe architecture is finally illustrated through an example of a Memory Write Transaction.

Chapter three is dedicated to the design of a PCI Express based Endpoint. First, an overview of the design is given. The Microblaze based Endpoint device is presented next. The complete design is overviewed. Then all the components and IPs building up the system are discussed. The PX1011A PHY physical layer, its block diagram, operational principle and interfaces are demonstrated. After this, the PCIe core, its block diagram, functionality, features, interfaces, generation and configuration are discussed. The Microblaze core, its interfaces, the Local Memory Bus (LMB) and the On-chip Peripheral Bus (OPB) are also explained in this chapter. The final section of this chapter concentrates on the design of the Microblaze PCIe peripheral. This includes a detailed description of the developed OPB to PCIe Bridge, its internal structure, interfaces, and functionality.

Chapter four presents the simulation of the designed PCIe Endpoint. It introduces the simulation models of each functional block in this Endpoint device. In this chapter the PCIe Downstream Port simulation model provided by Xilinx is explained. Its integration into the PCIe Testbench is also demonstrated. Then, a detailed description of the C application program executed by the Microblaze is given. The next section in this chapter provides the simulation flow, followed by a summary of the conducted testcases.

Finally, chapter 5 brings some conclusions and highlights future work.

2 PCI Express Theory

2.1 Evolution of IO Bus Systems

Since the 1980s till nowadays, many bus systems have been developed to serve different electronic devices, computing and communication platforms.

Figure 2.1 depicts the evolution of IO bus systems. The first IO buses generation, which is located at the bottom of the figure, was introduced in the 1980s, including the Industry Standard Architecture (ISA), which enables a very low bandwidth of 16.7 Mbytes/s, a sufficient one at that time. Extended ISA (EISA) and Video Electronics Standards Association (VESA) are other buses of this generation.

In the 1990s, the second IO buses generation was started with different buses. In 1993 the PCI 33 MHz bus was released. At that time, a 32-bit version of this bus was enough to deliver a bandwidth of 133 Mbytes/s, which met the bandwidth requirements of the available IO peripherals. A 64-bit version of this PCI bus delivers a bandwidth of 266 Mbytes/s [AS99].

However, due to the increase in the processor speeds and the bandwidth needs of new developed IO technologies, the PCI bus frequency was increased in 1995 from 33 to 66 MHz, to increase the bandwidth from 133 Mbytes/s to 266 Mbytes/s for a 32-bit PCI, and from 266 Mbytes/s to 533 Mbyte/s for a 64-bit PCI, correspondingly [ABS04].

Several practical limitations of the PCI 66 MHz bus and the emerging of new high end system technologies that continued asking for higher bandwidths led in 1999, to the releasing of a new generation of the PCI called the PCI-X bus.

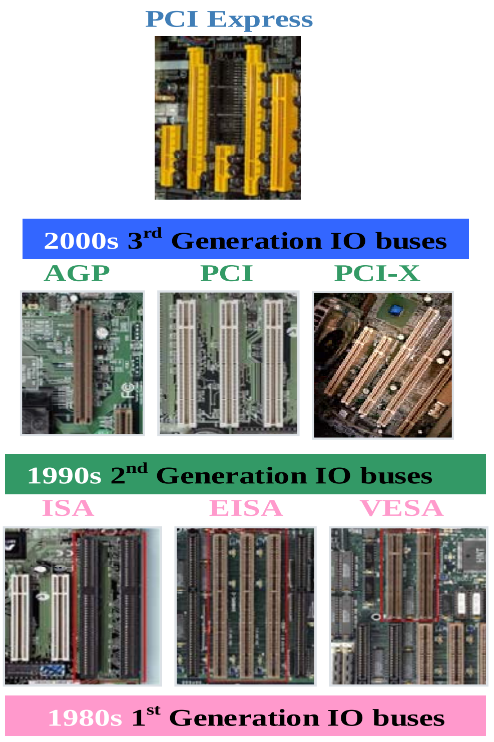

PCI Express
2000s 3rd Generation IO buses
AGP
PCI
PCI-X
1990s 2nd Generation IO buses
ISA
EISA
VESA
1980s 1st Generation IO buses

Figure 2.1 - *Evolution of IO bus systems*

The PCI-X bus has frequencies of 66 and 133 MHz and enables a bandwidth up to 1.066 Gbytes/s. These frequencies were increased to 266 and 533 MHz in the first quarter of 2002, to increase the bandwidth provided up to 4 Gbytes/s [ABS04].

Another bus system in the second generation is the Accelerated Graphics Port (AGP). A x1 AGP bus, for example, enables a bandwidth of 266 Mbyte/s and a x8 AGP can enable a bandwidth of up to 2.1 Gbytes/s.

However, in order to meet the higher bandwidth requirements and to satisfy the bandwidth hungry devices, a new bus system was still needed.

The third and latest generation IO bus system is the PCIe, which was released in the second quarter of 2002. It evolved from the PCI and overcame the limitations of the PCI. The PCI Express (which is currently being adopted as general purpose IO devices interconnect in different applications) began shipping in standard desktop PCs in 2004. A x1 PCIe bus provides theoretically a bandwidth of 500 Mbytes/s, a x16 PCIe can provide up to 8 Gbytes/s, and a x32 provides 16 Gbytes/s [ABS04].

Next, the PCI bus system including its architecture, key features, practical limitations, and challenges will be explained.

After that, an illustration of the PCIe bus system architecture, key advantages, and future prospectives will follow.

2.2 Peripheral Component Interconnect (PCI)

2.2.1 PCI Architecture

Figure 2.2 illustrates an example of a *33 MHz PCI* bus based system, which consists of a processor bus to PCI bus Bridge, called the *North Bridge*, to which the Accelerated Graphics Port (AGP), system memory, and the 33 MHz PCI buses are connected. The PCI bus is bridged to the ISA bus over the so-called *South Bridge*, to which additionally the Integrated Device Electronics bus (IDE) and the Universal Serial Bus (USB) are connected.

The PCI bus is a multi-drop parallel interconnect which uses a shared bus topology (the bus bandwidth is shared) to allow data communication among the different devices that share the bus including the CPU.

The PCI bus operating at 33 MHz and 32 bits provides a peak theoretical bandwidth of 132 Mbytes/s. A bandwidth of 266 Mbytes/s is possible by extending the bus to 64 bits [ABS04].

Theoretically, up to 32 devices can be connected on a PCI bus. Due to some signal timing restrictions, the PCI bus cannot support more than 10-12 loads (or 5-6 connectors); each connector is equivalent to 2 loads [ABS04]. However, it is possible to connect more devices to the PCI bus by implementing a PCI-to-PCI bridge, as depicted in figure 2.2.

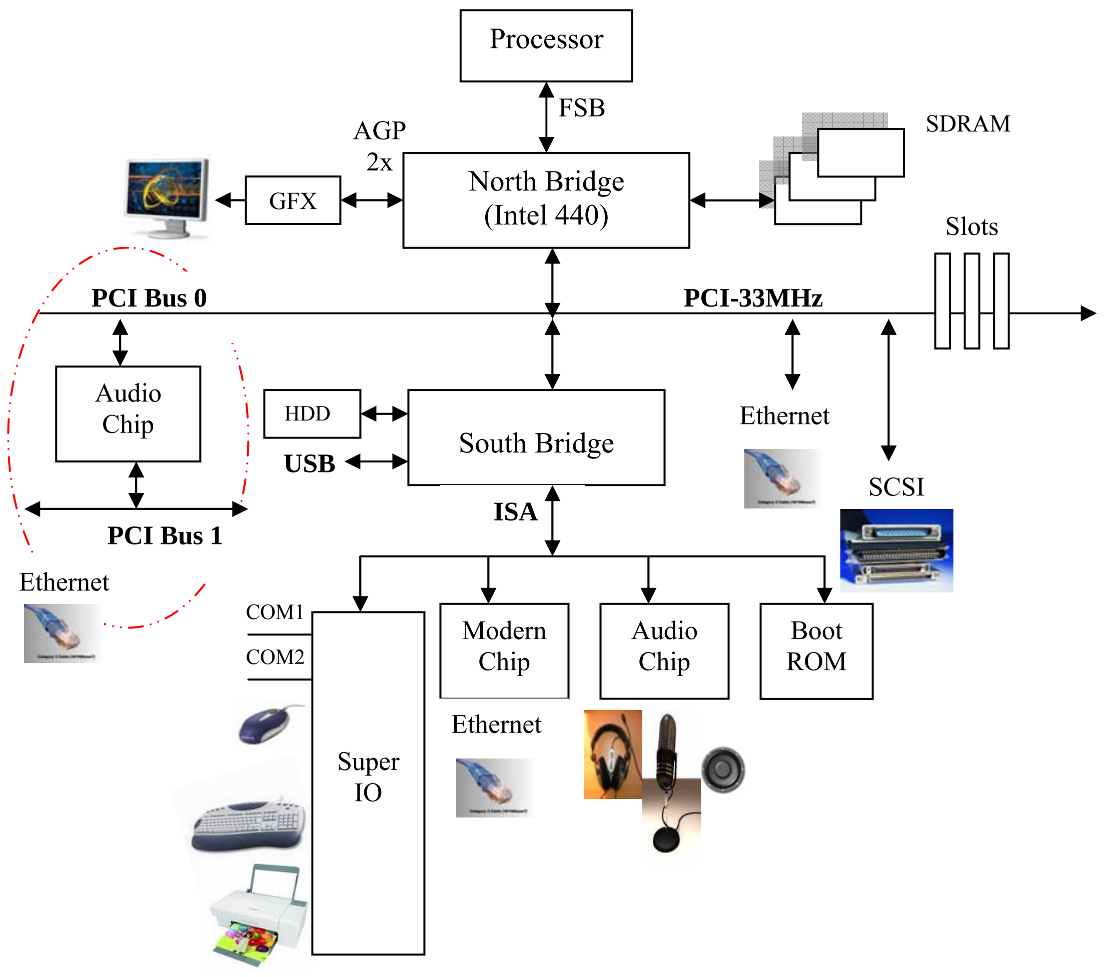


Figure 2.2 *- An Example of 33 MHz PCI Bus Based System [ABS04]*

Figure 2.3 shows an example of a *66 MHz PCI* bus based system, in which the latest generation of Intel PCI chipsets is used, where North and South bridges are replaced with a Memory Controller Hub (MCH) and an IO Controller Hub (ICH), respectively. A Hub link connects both of these hubs together. The figure also shows each 66 MHz PCI bus is accessed over a P64H (PCI 64-bit Hub) bridge connected to the MCH via Hub Link buses.

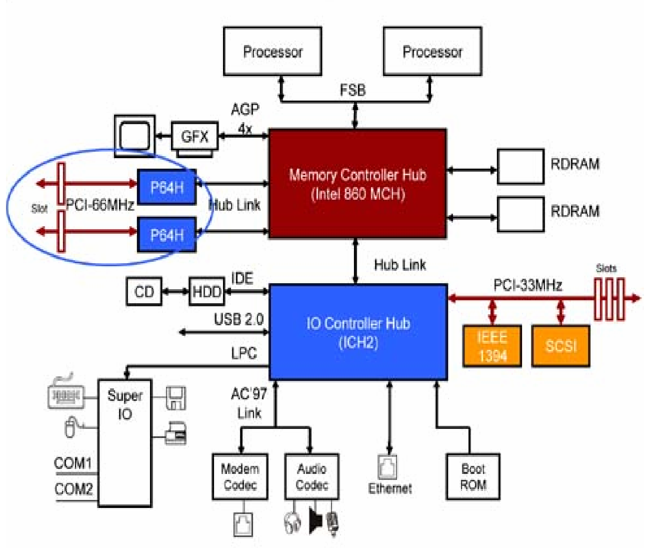


Figure 2.3 *- An Example of 66 MHz PCI Bus Based System [ABS04]*

The 66 MHz PCI bus system supports a bandwidth requirement of 533 Mbytes/s, and one connector to which a device can be connected, while the PCI-X can support from 8 to 10 loads or 4 connectors at 66 MHz and 3 to 4 or 1 to 2 connectors at 133 MHz. The peak bandwidth achievable with 64-bit/133 MHz PCI-X is 1064 Mbytes/s. A further improvement to the PCI-X is the *PCI-X 2.0* bus, which supports either Dual Data Rate (DDR) or Quad Data Rate (QDR) data transport, and provides a peak bandwidth capability of 4256 Mbytes/s for a 64-bit 533 MHz effective PCI-X bus [ABS04].

2.2.2 PCI Key Features

The PCI bus overcame the limitations of its predecessors and had several advantages over them.

Referring to figure 2.2, one can see a kind of partitioning into two hubs, the MCH and the ICH. Indeed, this provides a kind of processor independency and buffered separation. Separating the CPU local bus from the PCI bus, gives the ability to run simultaneous cycles on the CPU and PCI buses. It also allows the CPU local bus to increase its frequency accompanied by a change in the memory bus, independent of the PCI bus speed and loading.

The PCI bus provides a bus mastering connectivity, where the PCI devices arbitrate to access the bus and master the bus transaction directly instead of waiting for the CPU to serve them. This results in reducing the overall latency.

Another advantage of the PCI bus is the plug and play operation, which allows devices to be automatically detected and configured.

2.2.3 PCI practical Limitations and challenges

The PCI bus has limited bandwidth capabilities, which makes it an unsatisfying choice for several applications, which require a higher bandwidth.

In the industry, two ways are followed to adapt the performance of a bus system to the devices' requirements: increasing the number of signals, or increasing the signal frequency. In both cases, the bus system reaches its limitations. Both solutions also add extra costs to the development phase.

The PCI bus's frequency cannot be scaled up, and its voltage cannot be scaled down. It faces some time restrictions and stringent signal routing rules.

The PCI bus implements a shared bus topology, in which many devices share the same bus. Some of these devices can monopolize more than 80% of the available PCI bus bandwidth.

PCI bus efficiency is reduced. This reduction is due to several factors:

- Masters and slaves are allowed to insert wait states in the bus cycle. Slower devices will make the transfer on the bus slower.
- The transfer size on the bus is not indicated, which leads to an inefficiency in the buffer management within both the master and slave devices.

- The handling of delayed transactions on the PCI bus is inefficient.
- The architecture of PCI follows strict ordering rules as defined by the PCI specification.
- The way the PCI architecture handles the interrupts is inefficient, because many devices share the same PCI interrupt signal, which imposes additional time latency in discovering which of these devices has generated the interrupt.

The PCI bus does not support real-time data transfer services. As many applications today require the data streaming from video and audio devices, the bus must set some priorities in processing these time-dependent data in a process called the Quality of Services (QoS).

This bus also does not provide advanced power management features, which are required by many modern electronic devices.

All these limitations and challenges have motivated the developing of a new IO bus generation. The PCI Express bus system was the result of the developments carried out by Intel. This PCIe bus system was brought to the market in 2004, and is now used as a general IO Interconnect in diverse applications.

The PCIe bus system is discussed next. Its topology, architecture and layer structure are explained.

2.3 Peripheral Component Interconnect Express (PCIe)

2.3.1 PCIe Introduction

Unlike the PCI bus, the PCIe bus is serial. Figure 2.4 shows a PCIe *Link*, which implements a high performance, high speed, point-to-point, dual simplex, low-pin-count and differential signalling Link for interconnecting devices. This bus system was developed to overcome the limitation of the original PCI bus.

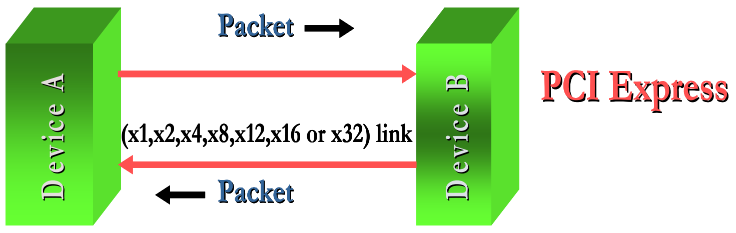


Figure 2.4 *- PCI Express Link [ABS04]*

The PCIe link shown in the figure implements the physical connection between two devices. A PCIe interconnect is constructed of either a x1, x2, x4, x8, x12, x16 or x32 point-to-point link. A x1 Link has 1 *Lane* or 1 differential signal pairs in each direction, transmitter and receiver, with a total of 4 signals. Correspondingly, x32 Link has 32 Lanes or 32 signal pairs for each direction, with a total of 128 signals [ABS04].

PCIe employs a packet-based communication protocol with a split transaction. Communication in this bus system includes the transmission and reception of packets called Transaction Layer packets (TLPs).

The transactions supported by PCIe protocol can be grouped into four categories: Memory, IO, Configuration, and Message transactions.

2.3.2 PCIe Topology

The PCIe topology shown in figure 2.5 contains different components. A Root Complex, PCIe switches, PCIe Endpoints, Legacy Endpoints, and optional PCIe to PCI bridges.

The *Root Complex* connects the CPU and the memory to the PCIe fabric. For instance, an Intel chipset could be used as a Root Complex.

The main purpose of the Root Complex is to generate transaction and configuration requests on behalf of the CPU.

PCIe implements a switch-based topology in order to interconnect multiple devices. These *Switches* implement multiple, logical, and virtual bridges.

Shown in the figure are switches with one upstream port that points in the direction of the root complex, and two downstream ports, which point in the opposite direction. These switches can have any number of ports.

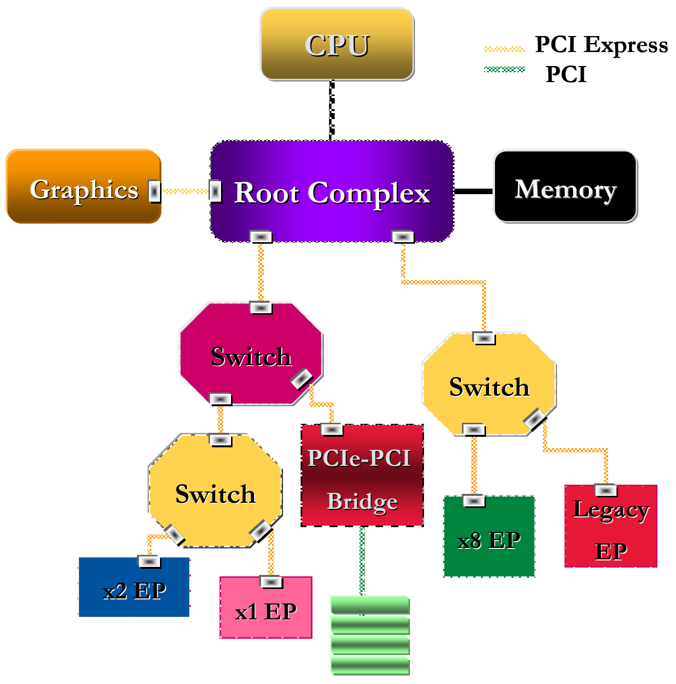


Figure 2.5 *- PCI Express Topology*

PCIe Endpoint (EP) is a device which can be a requester that originates a PCIe transaction or a completer that responds to a PCIe transaction addressed to it.

As mentioned above, these Endpoints can posses a x1, x2, up to x32 link. PCIe Endpoints are peripheral devices such as Ethernet, USB or graphic devices. *Legacy Endpoint* does not support all the transaction like the PCIe Endpoint**.**

In order to connect some PCI devices to the PCIe fabric, a *PCIe to PCI Bridge* must be used.

2.3.3 PCIe Key Features

The shared bus topology used for PCI is replaced with a shared switch, which provides each device, with a direct access to the bus.

In a PCIe based system, unlike the parallel PCI bus system, data is sent serially in packet based protocol.

PCIe bus has an advantageous attribute of frequency and bandwidth scalability, because it implements a point-to-point interconnect, which limits the electrical load on the link, allowing transmission and reception frequencies to be scaled up. Multiple lanes can be used to increase the bandwidth of the PCIe link.

PCIe supports the same address spaces as PCI: memory, IO, and configuration address spaces. Additionally, it enhances the configuration address space by extending it from 256 Bytes to 4 Kbytes [PXS05].

The same transaction types supported by PCI and PCI-X are used by the PCIe. These include Memory Read and Memory Write, IO Read and IO Write, Configuration Read and Configuration Write. The PCIe bus also supports a new transaction type called Message transaction.

PCIe offers a new feature, called the Quality of Service (QoS). This new feature allows the routing of packets from different devices with different priorities.

PCIe uses a flow control mechanism. This ensures that the TLP won't be transmitted unless there is enough space in the receiving device.

PCIe uses Message Signalled Interrupt (MSI) style for handling interrupts. In order to interrupt the CPU, a Memory Write packet is used to write an interrupt vector to the Root Complex, which in-turn interrupts the CPU.

Other features supported by PCIe are the advanced power management features, which enable the design of low power mobile devices. PCIe also supports hot plug and hot swap features. Signalling of such features is carried out in-band using packet based messaging instead of side-band signals. This has the advantage of keeping the device pin count low.

PCIe applies the same programming model as PCI and PCI-X. It also has a configuration model which is compatible with PCI configuration model, shown in figure 2.6. It is also compatible with existing operating systems, bus enumeration and configuration software for PCI/PCI-X [ABS04].

2.3.4 PCIe Architecture

PCIe has a layered architecture as depicted in figure 2.7. It consists of the Transaction Layer, the Data Link Layer and the Physical Layer. On the top of these three layers resides the Software Layer, or device core. Each of these layers is further divided into two sections: transmitter and receiver.

The transmitter is responsible for processing the Transaction Layer Packets requested from the device core before being transmitted across the PCIe link. At the same time, the receiver processes the incoming TLPs before sending them to the device core.

To demonstrate the functionality of PCI Express protocol and for the purpose of this diploma work, 32-bit addressable Memory Write/Read and Completion with Data (CPLD) TLPs will be considered.

Figure 2.8 shows the assembly and disassembly of a PCIe TLP. It also illustrates the contribution of each layer to this TLP.

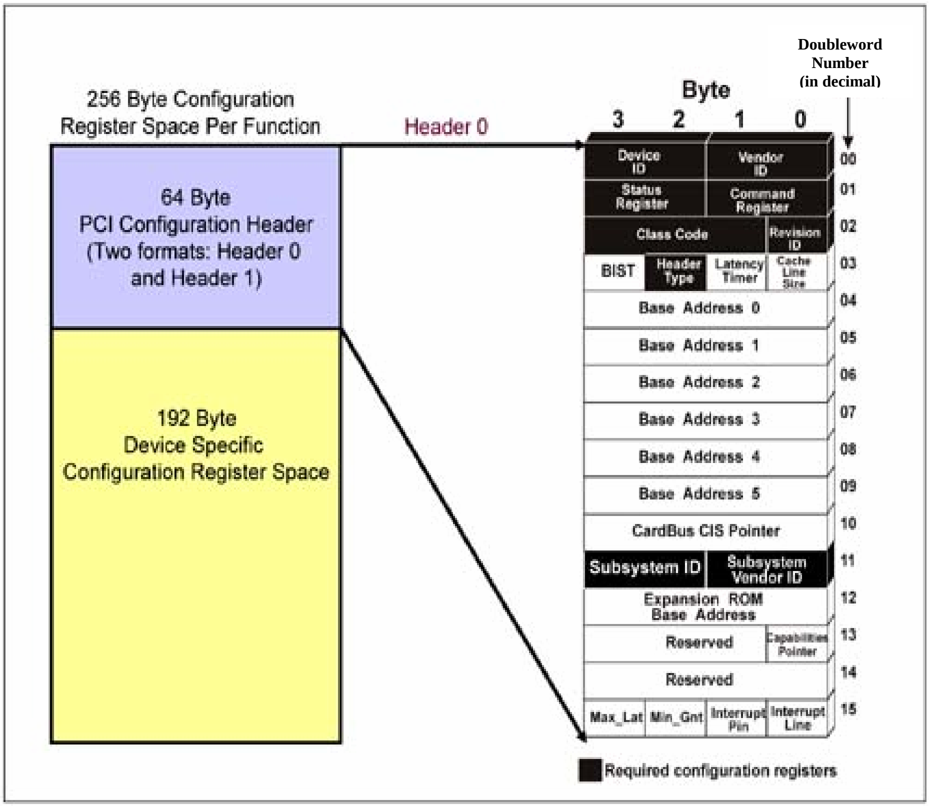


***Figure 2.6** - PCI Configuration Model [ABS04]*

The Memory Write TLP is considered to be a posted transaction where the requester transmits a request TLP to the completer. This in turn does not return a completion TLP back to the requester. Unlike the Memory Read TLP where the completer is supposed to return a completion TLP back to the requester. The completer returns either a Completion with Data (CPLD), if it is able to provide the requested data, or a Completion without data (CPl), if it fails to obtain the requested data.

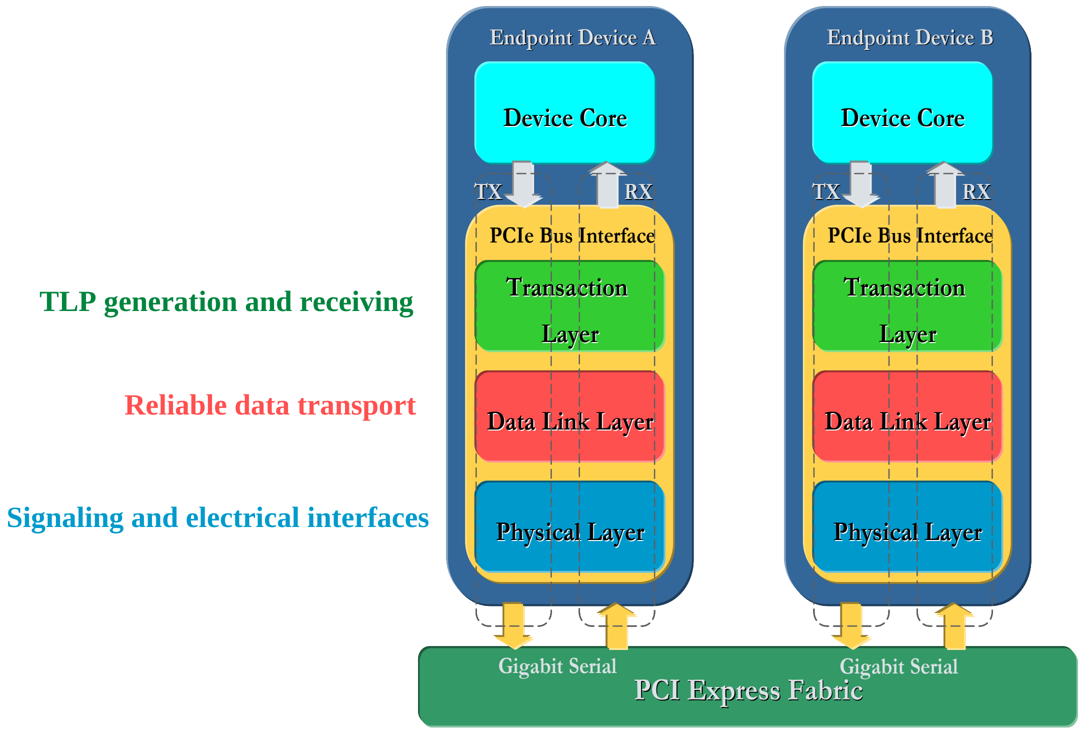


Figure 2.7 *- PCI Express Architecture [XP05]*

In the illustration below, the core of device B issues a Memory Write request in order to write some data to a memory mapped location within device A.

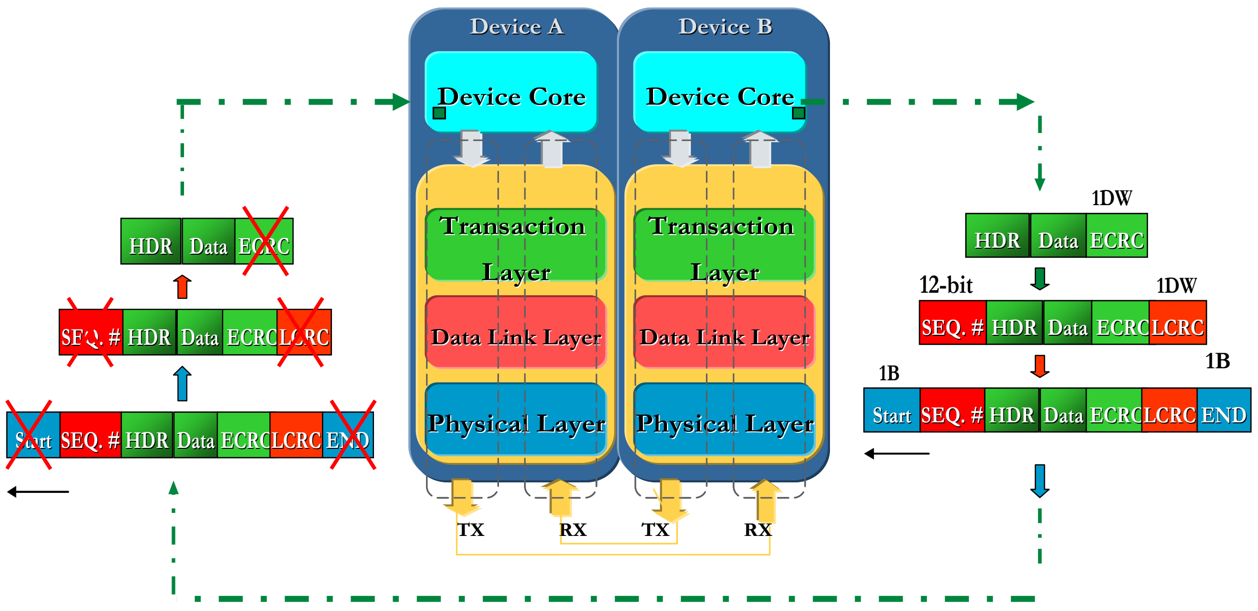


Figure 2.8 *- PCI Express TLP Assembly/Disassembly*

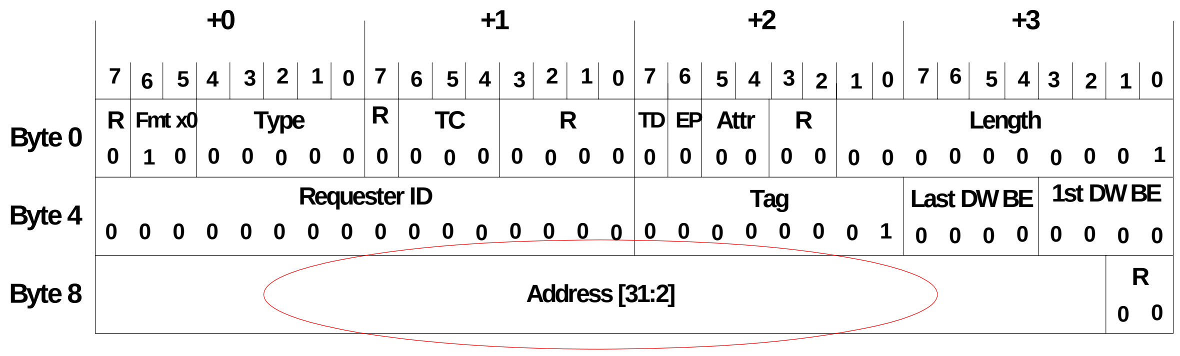
Device Core

The core of device B, which could be the Root Complex core logic or Endpoint core logic, sends to the transaction layer the information required to assemble the TLP. This information contains the Header (HDR) and the Data Payload (if it exists), because some TLPs do not have data payload, as in the case of Memory Read TLPs.

The size of the Header can vary between 3 and 4 DWs depending on the TLP. 3 DWs are used for 32-bit addressable Memory and CPLD TLPs, while the Header with 4 DWs is dedicated to 64-bit addressable Memory TLPs. The maximum size of the data payload is 4Kbytes (1024 DW) [ABS04].

Figure 2.9 depicts the Header of a 32-bit addressable Memory Write request to write data of 1 DW payload to a memory mapped location of 32-bit address within device A[1].

This Header consists of 3 DWs. In the case of 64-bit addressable Memory TLPs, one more DW is used. Bytes 12 to 15 must be added to the Header shown in the figure.



	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	R	Fmt x0		Type					R	TC			R				TD	EP	Attr		R		Length									
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								Last DW BE				1st DW BE			
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Byte 8	Address [31:2]																														R	
																															0	0

***Figure 2.9** - Header for a 32-bit Memory Write TLP*

The figure also shows the different fields in this Header. The following is a detailed explanation of each field:

- Byte0 [7]: *R* (Reserved bit): This bit should be set to zero.
- Byte0 [6:5]: *Fmt* (Packet Format) and Byte0 [4:0]: *Type* (TLP packet Type field) are used in a combination that specifies the transaction type, header size, and whether data payload is present or not (Byte0 [6:0]):

  0000000b = *Memory Read (3DW without data)*
  0100000b = *Memory Read (4DW without data)*
  1000000b = *Memory Write (3DW with data)*
  1100000b = *Memory Write (4DW with data)*
  0001010b = *Completion (3DW without data)*
  1001010b = *Completion (3DW with data)*

[1] refer to [ABS04] for CPLD & Configuration TLPs

- Byte1 [7]: *R* (Reserved bit): This bit should be set to zero.
- Byte1 [6:4]: *TC* (Traffic Class): These 3 bits are used to determine the traffic class to be applied to the TLP. There are seven different traffic classes. In this example, the default traffic class was applied to the transmitted TLP:

  000 = *Traffic Class 0 (Default Class)*
  001 = *Traffic Class 1*
  010 = *Traffic Class 2*
  011 = *Traffic Class 3*
  100 = *Traffic Class 4*
  101 = *Traffic Class 5*
  110 = *Traffic Class 6*
  111 = *Traffic Class 7*

- Byte1 [3:0]: *R* (Reserved bits): These bits should be set to zeros.
- Byte2 [7]: *TD* (TLP Digest Field Present): If set = 1, the optional 32-bit Cyclic Redundancy Check (CRC) field is included with this TLP. All receivers must check the presence of this field when this TD is set to 1.
- Byte2 [6]: EP (Poisoned data): When set = 1, the payload data with this TLP should be considered corrupted, although the transaction completes normally.
- Byte2 [5:4]: *Attr* (Attributes): Bit 5 = Relaxed ordering: If set = 1, the PCI-X relaxed ordering is enabled for this TLP. Otherwise, strict PCI ordering is used. Bit 4 = No Snoop.
- Byte2 [3:2]: *R* (Reserved bits): These bits should be set to zeros.
- Byte2 [1:0] and Byte3 [7:0]: *length*, TLP data payload transfer size (in DW). Maximum transfer size is 10 bits; 2^{10} = 1024 DW (4Kbytes). Encoding [ABS04]:

  00 0000 0001b = *1DW*
  00 0000 0010b = *2DW*
  .
  .
  11 1111 1111b = *1023 DW*
  00 0000 0000b = *1024 DW*

- Byte4 [7:0] and Byte5 [7:0]: *Requester ID*: Indicates the identification number of the device that generates the TLP. This number is indicated for the purpose of returning a completion TLP.

  Byte4 [7:0]: *bus number*,
  Byte5 [7:3]: *device number and*
  Byte5 [2:0]: *function number*.

- Byte6 [7:0]: *Tag*: These bits are used to identify each transmitted request issued by the requester. Upon the sending of one request, the next sequential tag is assigned. By default, only 5 bits are used for this tag, which allows 32 outstanding transactions at a time. This number can be extended to 256 tags by having 8 bits used, when configuring the PCIe core by setting the extended tag bit in the PCIe control register = 1.
- Byte7 [7:4]: *Last DW BE*: These bits are used to qualify the bytes in the last sent DW. These byte enables are active high.

A value of "0" indicates that the concerned byte should not be written by the completer of the TLP. It is written otherwise. Since we have the valid transfer data are within only 1 aligned DW, the Last DW BE must be = 0000b.

- Byte7 [3:0]: *1st DW BE*: These bits are used to qualify the bytes in the first sent DW. Since we have the valid transfer data are within only 1 aligned DW, the 1st DW BE must be = 1111b.
- Byte8 [7:0], Byte9 [7:0], Byte10 [7:0] and Byte11 [7:2]: *Address*: 32-bit addressable memory mapped location. This targeted address is used to route the Packet in the PCIe fabric to the intended device.
- Byte11 [1:0]: R (Reserved bits): These bits should be set to zero. Doing so forces the 32-bit start address to be DW aligned.

In figure 2.8, the PCIe based transmitter and receiver are illustrated. The following is an explanation of the role each of the PCIe layers plays when transmitting and receiving TLPs.

Transaction Layer

The main functionality of the Transaction Layer is the generation of TLPs to be transmitted across the PCIe link and the reception of TLPs received from the PCIe link.

Transaction Layer employs the split transaction protocol, by associating the incoming completion TLP of a certain tag with the transmitted non posted TLP of the same tag.

In this layer, Transmission buffers are included to store the TLPs that wait to be transmitted as well as to store the received TLPs. This layer provides a flow control mechanism, ensuring that the TLP won't be transmitted unless there is enough space in the receiving device.

Also in this layer, the Quality of Service protocol is implemented, which prioritizes the transmission and receiving of TLPs.

The contribution of this layer to the transmitted packet is shown in figure 2.8. This layer appends a 32-bit End to End Cyclic Redundancy Check (ECRC). This ECRC is generated based on the whole TLP from the first byte of the Header to the last byte of the data Payload, in order to check for CRC errors in the header and the data Payload. These 32 bits are stripped out of the incoming TLPs before being forwarded to the core of the receiving device (as shown in figure 2.8).

Data Link Layer

This layer is responsible for ensuring a reliable data transport on the PCIe link. The received TLP from the transaction layer is concatenated with a 12-bit sequence ID and a 32-bit Link CRC (LCRC) as shown in figure 2.7. The LCRC is calculated based on all the bytes within the TLP in addition to the sequence ID. These added bits are stripped out from the incoming TLP by the same layer in the receiving device before being transferred to the Transaction Layer (as shown in figure 2.8).

The Data Link Layer applies a replay mechanism (ACK/NAK) to ensure the transmission of the TLPs across the link. Before sending the TLP, it copies it into a replay buffer. The sequence ID is used to associate this copy with a received ACK/NAK Data Link Layer Packet (DLLP) from the targeted receiver.

This ACK/NAK packet indicates whether the transmitted TLP has been received with or without errors. If no errors are found, the reply buffer is cleared. Until then the stored TLP is sent again and again until it is received properly.

Physical Layer

The physical layer of a PCIe device is responsible for driving and receiving the Low Voltage Differential Signals (LVDS) at a high speed rate of 2.5 Gbps each way. It interfaces the device to the PCIe fabric. Such an interface is scalable to deliver a higher bandwidth. The physical layer supports for example x1, x2, x4, x8, x12, x16, and x32 lane widths.

The TLPs and DLLPs are transferred to this layer for the purpose of transmission across the link. This layer also receives the incoming TLPs from the link and sends them to the Data Link Layer.

In this layer the data clock is embedded using an 8b/10 encoding algorithm, in order to obtain the high data rate.

Figure 2.8 shows the contribution of this layer to the transmitted packets. It appends 8-bit Start and End framing characters to the packet before being transmitted. The physical layer of the receiving device in-turn stripes out these characters after recognizing the starting and ending of the received packet, and then forwards it to the Data Link Layer.

In addition to that, the physical layer of the transmitter issues Physical Layer Packets (PLPs) which are terminated at the physical layer of the receiver, such PLPs are used during the *Link Training and Initialization* process. In this process the link is automatically configured and initialized for normal operation; no software is involved. During this process the following features are defined: link width, data rate of the link, polarity inversion, lane reversal, bit/symbol lock per lane, and lane-to-lane de-skew (in case of multi-lane link).

2.3.5 PCIe future prospective

The current PCIe bus represents the first PCIe bus generation demonstrating a bandwidth capability of 2.5 Gbps. The second and third generations of this bus are expected in the future, and will have bandwidths of 5 Gbps and 10 Gbps, respectively [ABS04].

The layered architecture of PCIe allows such an increase in the bandwidth by redesigning the physical layer only. No modification is required on the other layers. Such architecture leaves the door open for using optical fibers for instance, as a medium to carry packets in the PCIe fabric.

3 PCIe Endpoint Design

3.1 Design Overview

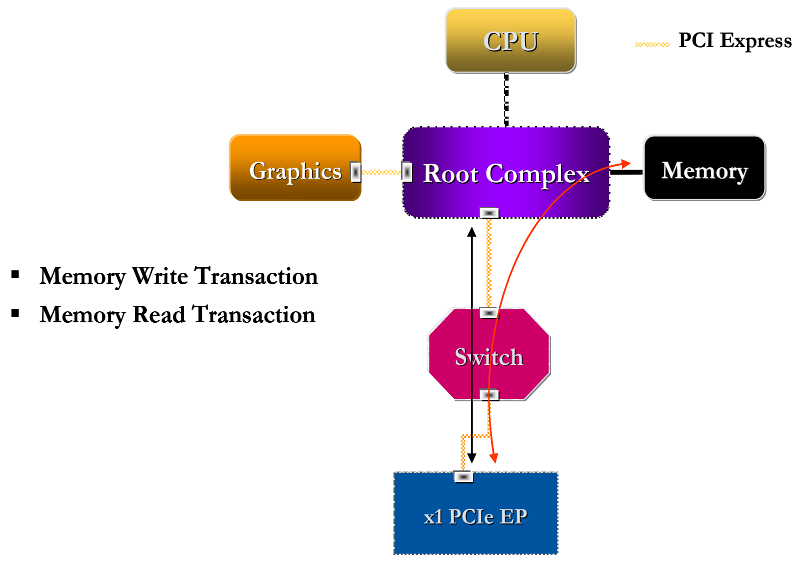


Figure 3.1- *Basic Memory Transactions*

Figure 3.1 shows the PCIe topology again. For design purposes, the x1 PCIe Endpoint will be considered.

In this illustration, the Endpoint is an intelligent device which acts as a target for downstream TLPs from the CPU through the Root Complex and as an initiator of upstream TLPs to the CPU.

In this diploma work, the PCIe Endpoint was designed. This Endpoint generates or responds to Memory Write/Read transactions. Since the used PCIe core supports up to six 32-bit Base Address Registers (BARs) used to route the TLP [XUG167], the behavior of this Endpoint can be easily extended by reconfiguring this core to have memory and IO address spaces.

When the Endpoint acts as a receiver, the CPU issues a store register command to a memory mapped location in the Endpoint. This is done by having the Root Complex generate a Memory Write TLP with the required memory mapped address in the Endpoint, the payload size (a Doubleword in this design), byte enables and other Header contents. These will be discussed later in the device core section of this Endpoint.

This TLP moves downstream through the PCIe fabric to the Endpoint. Routing of the TLP in this case is based on the address within its Header. A termination of the transaction takes place when the Endpoint receives the TLP and writes the data to the targeted local register.

To read this data back, the CPU issues a load register command from the same memory mapped location in the Endpoint. This is done by having the Root Complex generate a Memory Read TLP with the same memory mapped address and other Header contents. This TLP moves downstream through the PCIe fabric to the Endpoint. Again, routing here is based on the same address within the Header.

Once the Endpoint receives this Memory Read TLP, it generates a Completion with Data TLP (CPLD). The Header of this CPLD TLP includes the ID number of the Root Complex, which is used to route this TLP upstream through the fabric to the Root Complex, which in-turn update the targeted CPU register and terminates the transaction.

The other way around, is to have the Endpoint act as a bus master and initiate a Memory Write TLP to write 1 DW to a location within the system memory. This TLP is routed upstream toward the Root Complex which in turn writes the data to the targeted location in the system memory.

If the Endpoint wants to read the data it has written, it generates a Memory Read TLP with the same address. The TLP is steered to the Root Complex, which in-turn accesses the system memory, gets the required data and generates a Completion with this data. This CPLD TLP is routed downstream to the Endpoint through the PCIe fabric. The Endpoint receives this TLP, updates its local register and terminates the transaction.

As mentioned in chapter 2, the PCIe core can be integrated in different devices composing the PCIe fabric. For instance, the core can be implemented in the Root Complex, in the PCIe switch, and in the PCIe Endpoint. For the purpose of this diploma work, the focus will be on designing a PCIe Endpoint.

When designing a PCIe Endpoint, several issues have to be considered. Figure 3.2 shows the layered structure of a PCIe Endpoint device. In the figure, the way this Endpoint was designed is depicted.

The physical layer provides the electrical transceivers, which drive and receive the dual-simplex low voltage differential signals at the 2.5 Gbps data rate. There are two different solutions for the physical layer. In the first solution, this layer can be integrated with the other layers in the same chip. Doing so increases the complexity of this chip and provides a higher integration level. This integrated solution has one key advantage when designing using an FPGA. It uses less number of IO pins, which enables easier timing closure.

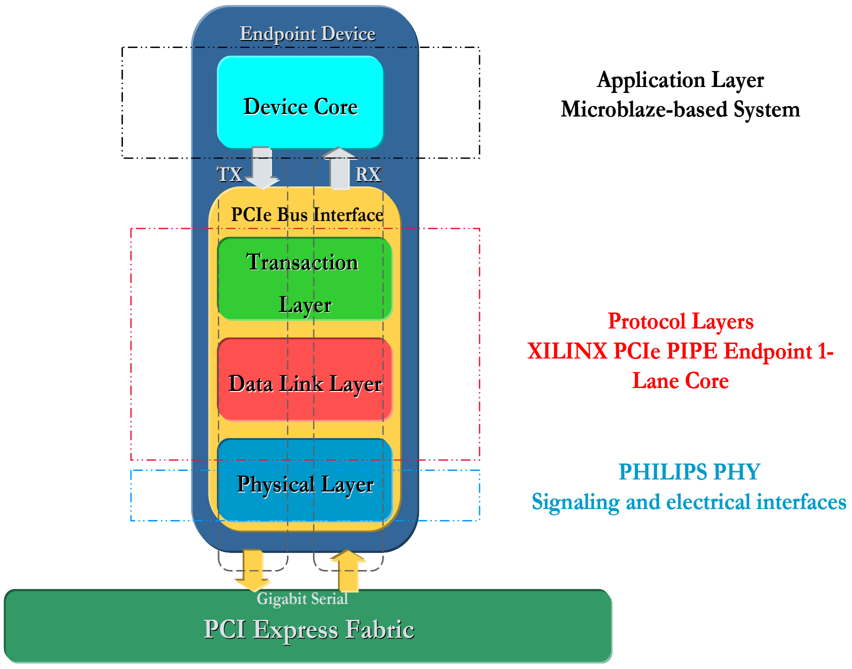


Figure 3.2 - *Endpoint Design*

An example of this integrated solution is offered by Xilinx in their newly introduced Xilinx Virtex-5 PCIe Endpoint block shown in figure 3.3.

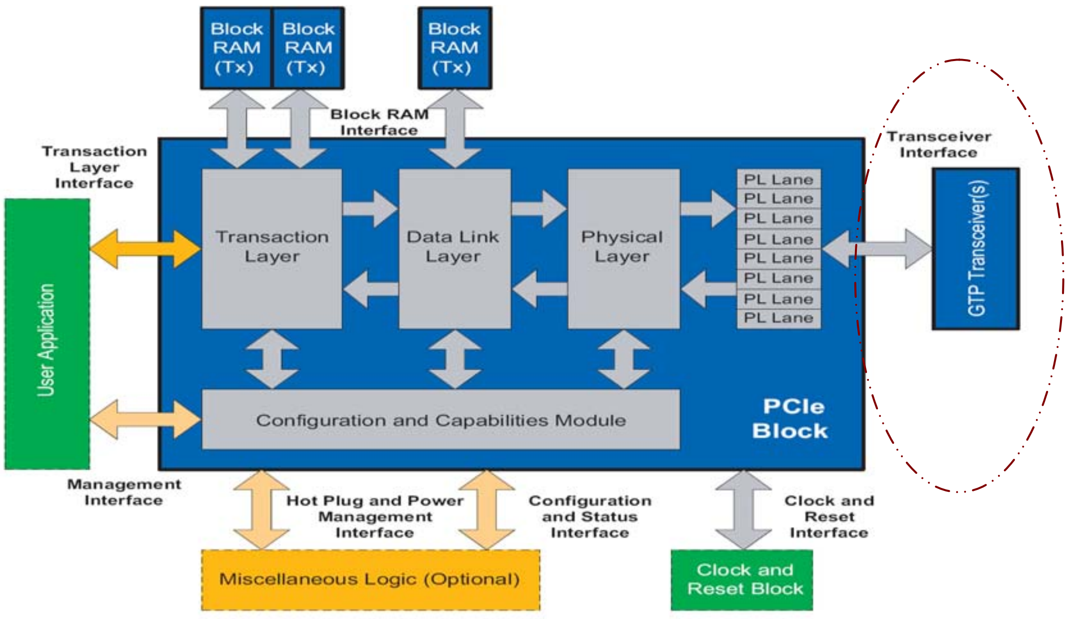


Figure 3.3 - *Xilinx Virtex-5 LXT PCI Express endpoint block [UG197]*

Unlike the first solution mentioned above which is quite expensive, the second solution offers a low cost way of implementing the PCIe Endpoint. In this solution, the physical layer exists in one chip, and the other layers are designed in another chip.

In this two-chip solution, a smaller FPGA with external PHY can be used. Within this diploma work, the discrete PHY, PX1011A from Philips was used.

This PHY supports x1 PCIe designs. Having the practical bandwidth provided by x1 PCIe is 2.0 Gbps requires an internal interface of 8 bits runs at 250 MHz or an interface of 16 bits runs at 125 MHz. This solution has the disadvantage of higher number of IO pins.

The protocol layers containing the logical sub-layer of the physical layer, the data link layer and the transaction layer are implemented using the Xilinx PCIe core.

A Microblaze based embedded system was built up to implement the Application layer of the designed PCIe Endpoint. In this Microblaze processor embedded system, the PCIe core is attached as a slave to the processor, which in-turn tries to access the configuration space of this core, reading from and writing to this space.

In the application layer, the Microblaze is responsible for sending the required Header and data payload to the transaction layer of the PCIe core, which generates a TLP and forwards it to the data link layer. The Data link layer appends a 12-bit sequence number and a 32-bit LCRC, to ensure a reliable data transport. The TLP is then forwarded to the physical layer to be transmitted across the PCIe link.

When a TLP is received by the PCIe Endpoint, the Header and the payload, if exists, will be forwarded to the Microblaze for further processing. The Microblaze also controls the transmitting and receiving of TLPs.

The protocol layers and the application layer are to be implemented on a Xilinx® Spartan-™3/E FPGA, as depicted in figure 3.4, which shows a Spartan-3 PCI Express Starter Kit from Xilinx.

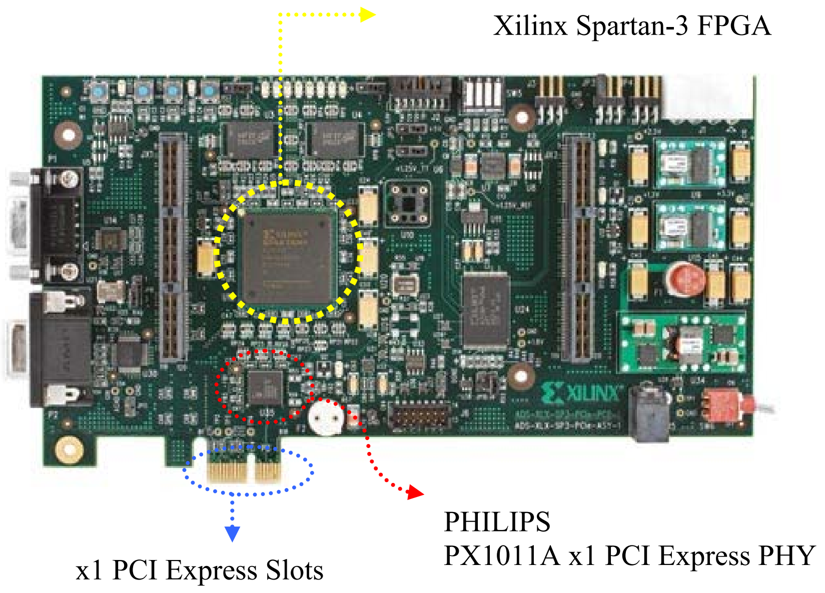


Figure 3.4 *- Spartan-3 PCI Express Starter Kit [Xilinx]*

Figure 3.5 shows the complete designed PCIe Endpoint. This system embeds the Xilinx Microblaze, which implements a 32-bit Reduced Instruction Set Computer (RISC) and operates at a frequency of 50MHz. Having the Microblaze as a soft core processor enables the design of a unique and customized PCIe peripheral device to be connected as a slave to it.

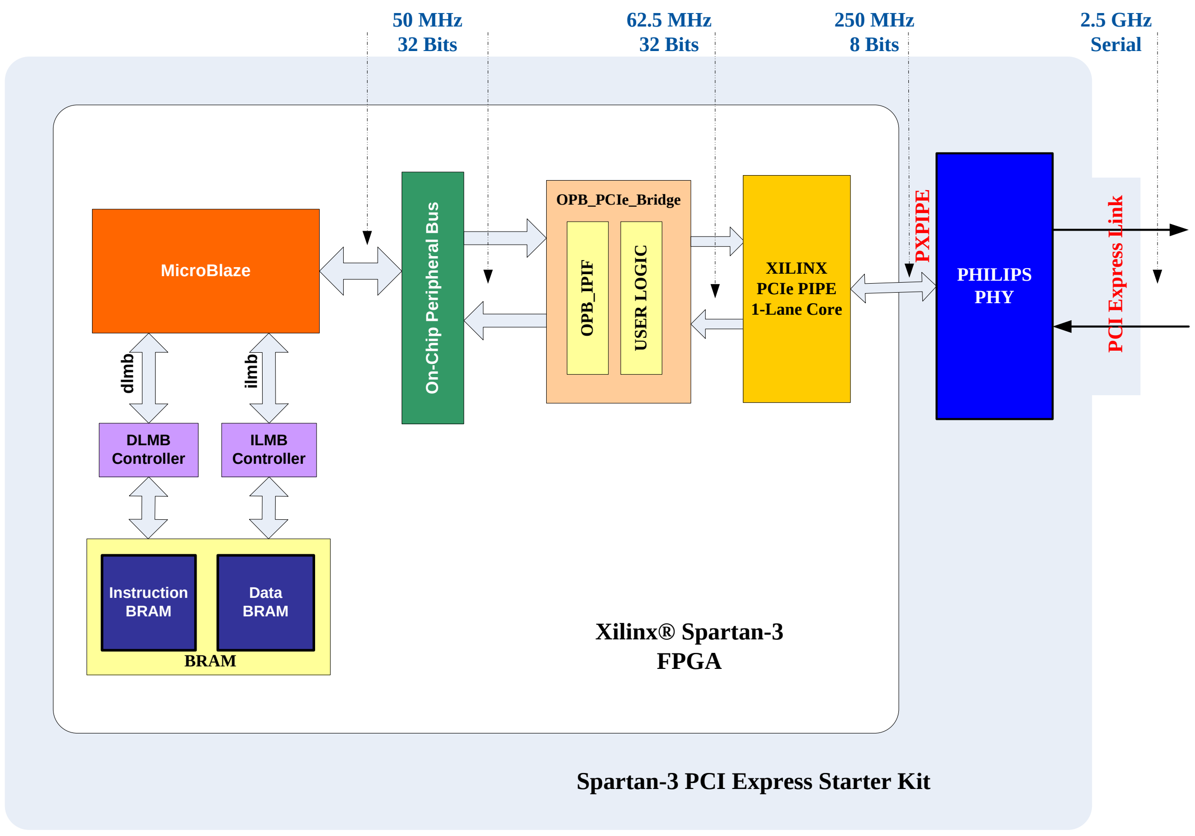


***Figure 3.5** - Complete PCIe Endpoint device*

The Microblaze has different bus interfaces, connecting it with different peripherals. For example, the Local Memory Bus (LMB) allows the communication between the processor and the Block Random Access Memory (BRAM), which is initialized with the application program to be executed by the Microblaze.

The Microblaze has a Harvard structure, in which the BRAM consists of two sections, data and instructions. These sections are accessed by the processor through memory controllers over the local memory bus.

Xilinx On-Chip Peripheral Bus (OPB), which implements the IBM CoreConnect On-Chip Peripheral Bus, has two 32-bit separate paths for data and address. This bus is used to connect peripherals to the Microblaze, which masters the bus. Several peripherals can be attached to the Microblaze as slaves.

The PCIe core can not be directly connected to the OPB as a slave, because of the incompatibility of its interfaces with the OPB protocol. To fulfill this compatibility issue, a bridge was developed to bridge the OPB and the PCIe core.

This bridge interfaces the OPB with its standard protocol through the OPB Intellectual Property Interface (OPB IPIF) from one side, and the PCIe core through the USER LOGIC model from the other side. This USER LOGIC model implements the logic needed to transmit/receive TLPs across the PCIe link and to access the configuration space of the PCIe core. The PCIe core transaction interfaces are synchronized with a clock of 62.5 MHz generated from the core as indicated in the figure.

The PCIe core interfaces with the Philips PHY using the Philips PHY Specification Physical Interface for PCI Express (PXPIPE), defined by Philips Semiconductors, which implements an extended version of the Physical Interface for PCI Express (PIPE), defined by Intel. PXPIPE is a 250 MHz source synchronous interface, which provides two clocks, one for transmission, and the other for reception.

Depicted in the figure are the interfaces of Philips PHY to the PCIe link, which are the low voltage differential signals (LVDS) driven at a high data rate of 2.5 Gbps.

In the following sections of this chapter, the components building up the PCIe Endpoint device are discussed in details[1].

3.2 Philips PX1011A PHY

Philips PHY is a standalone transceiver, which is optimized for usage with digital Application Specific Integrated Circuits (ASICs) and low cost FPGAs. This device implements a x1 PCIe physical layer. It provides a receiving bit error rate of less than 10^{-12} and comes in a small package used for chip to chip communication [KPE06].

3.2.1 Block Diagram

Figure 3.6 illustrates the block diagram of the PX1011A. It interfaces the Media Access Controller (MAC) of the physical layer of the protocol layers from the upper side, as well as the PCIe fabric from the other side.

3.2.2 Operation Principle

For the transmission of TLPs, the PX1011A receives words of 8 bits from the MAC, accompanied by a control bit that indicates whether the 8-bit word is data or a control character. The data is clocked in at a rate of one word per cycle of a 250 MHz clock. A First in First Out (FIFO) is used to compensate the phase difference between the interfacing clock and the internal 250 MHz transmitting clock generated by the Phase Locked Loop of the transmitter (TXPLL).

The data is first buffered in a FIFO. For the purpose of a high transmission rate, the transmission clock of 2.5 GHz is embedded by decoding the data using an 8b/10b encoder. The resulting symbols of 10 bits are then serialized and differentially transmitted across the transmission line.

[1] Excluding the PX1011A and the USER LOGIC, all the components are *IPs* provided by Xilinx

When the PX1011A receives the serial differential data from the transmission line, it recovers a clock from the incoming signal. This clock is used to sample the serial data. The sampled data is then forwarded to a serial to parallel converter, which converts the serial data into 10-bit symbols.

Once the 10-bit symbols are available, the symbol boundaries must be recognized. This is done by detecting a special 10-bit character called the "comma" character, which is used for symbol synchronization.

After the symbol synchronization, the synchronized 10-bit characters are passed through an elastic buffer that compensates the frequency difference between the recovered clock and the locally generated transmission clock. 8-bit data words are obtained by decoding the 10-bit symbols using an 8b/10 decoder. The resulting data is then stored in a register before being outputted to the protocol layers.

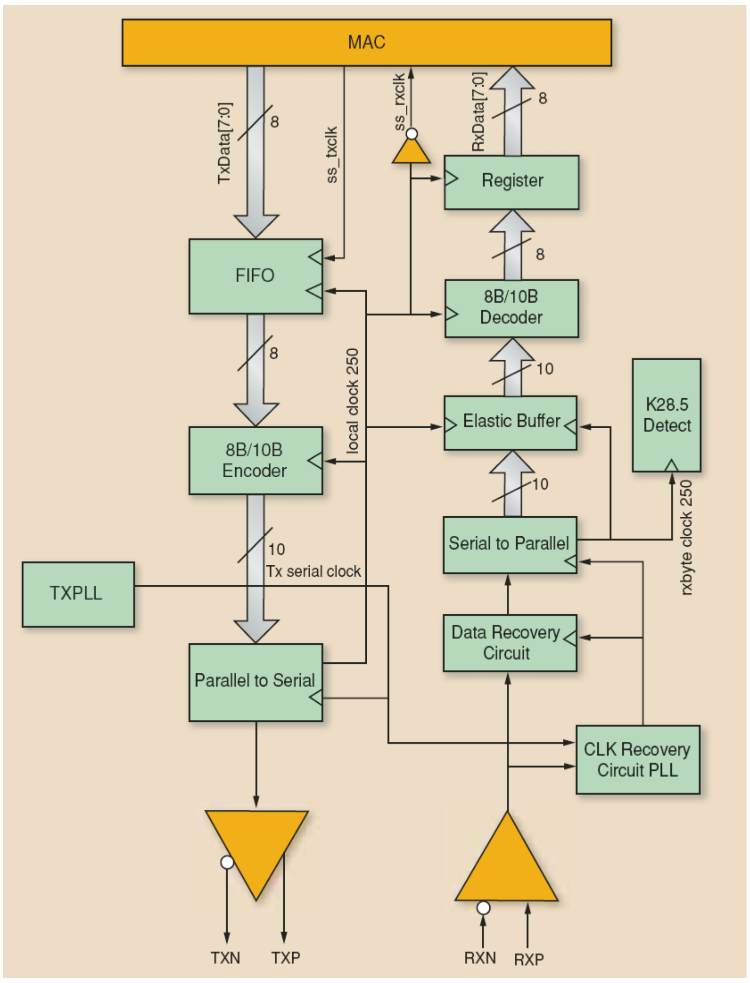


Figure 3.6 *- Block diagram of PX1011A [GL05]*

3.2.3 Interfaces

Interfaces to PCIe Link

The electrical part of the PX1011A physical layer interfaces the PCIe fabric with two Low Voltage Differential Signals (LVDS) to drive and receive the high data rate data of 2.5 Gbps. Figure 3.7 shows the electrical characteristics of a PCIe signal. A positive difference between the D+ and D- lines indicates the transmission of logic "1", while a negative difference implies a logic "0" on the link. Having a voltage difference of zero leads to a high impedance "tri-state" link, and forces the link to stay in the electrical idle state [ABS04].

Table 3.1 summarizes the interfaces of this layer to the PCIe link.

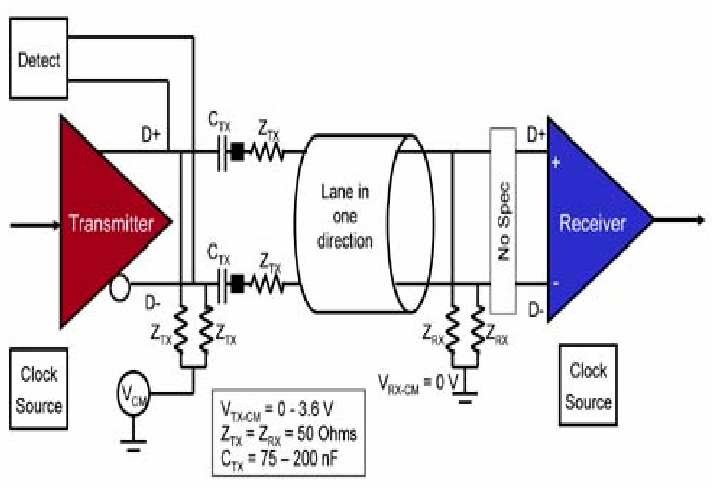


Figure 3.7 *- PCIe Differential Transmitter/Receiver [ABS04]*

Table 3.1 *- PHY Interfaces to the PCIe Link*

Signal	I/O	Description
TX_P	O	Positive transmission signal
TX_N	O	Negative transmission signal
RX_P	I	Positive receiving signal
RX_N	I	Negative receiving signal
REFCLK_P	I	Reference clock of 100 MHz
REFCLK_N	I	out of phase version of REFCLK_P

Interfaces to PCIe core

Figure 3.8 illustrates the PHY Interface for the PCI Express Architecture (*PIPE*), defined by Intel. This kind of interface assigns a single 250 MHz clock, referred to as PCLK in the figure. This clock synchronizes both the transmitting and receiving of data. This clock is outputted from the PHY as shown in the figure. Intel first introduced an 8-bit data interface.

Due to a timing budget problem, this interface was further improved to a 16-bit data interface. The newly introduced interface has the disadvantage of requiring more pins than the previous one and imposes an extra latency in converting from 16 to 8 bits.

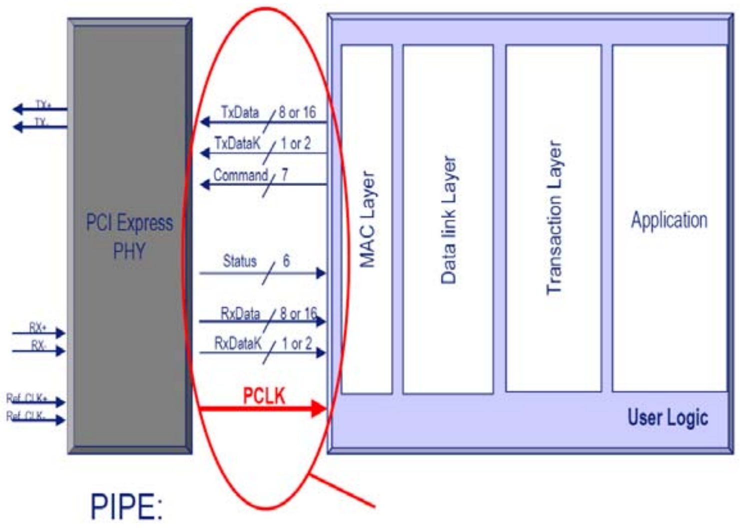


Figure 3.8 **-** *PIPE Interface [PPHY]*

Philips Semiconductors provided a version of the PIPE interface named *PXPIPE*. This interface employs the source synchronous clocking. Instead of having one clock for both directions, it provides two clocks: one for transmitting and another for receiving, as shown in figure 3.9.

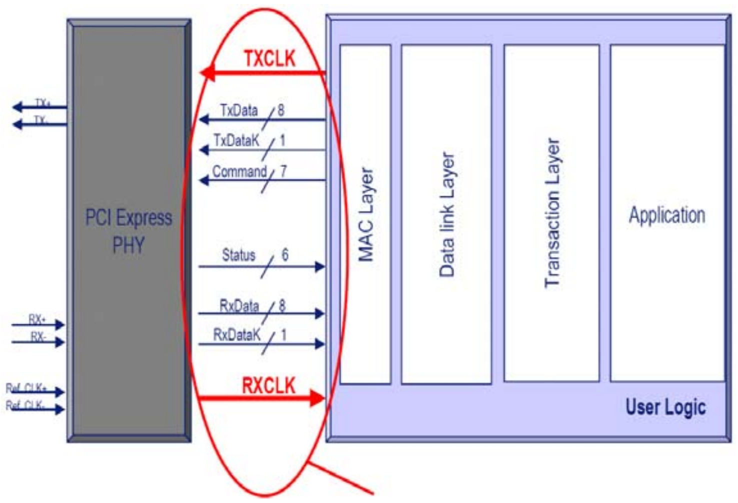


Figure 3.9 **-** *PXPIPE Interface [PPHY]*

The interface signals with the Xilinx PCIe core are summarized in table 3.2.

***Table 3.2** - Philips PHY Interfaces to the Xilinx PCIe Core [KPE05]**

Signal	I/O	Description
TXDATA[7:0]	I	8-bit transmit data from the FPGA to the PHY.
TXDATAK	I	Data or control for the symbols of transmit data. A value of "0" indicates a data byte; a value of "1" indicates a control byte.
RXDATA[7:0]	O	8-bit receive data from the PHY to the FPGA.
RXDATAK	O	Data or control for the symbols of receive data. A value of "0" indicates a data byte; a value of "1" indicates a control byte.
TXCLK	I	Source synchronous 250 MHz clock for transmit from the FPGA. All the data and input signals to the PHY are synchronized to this clock
RXCLK	O	Source synchronous 250 MHz clock for received data bound for the FPGA.
RXDET_ LOOPB	I	Enables the Philips PHY to begin a receiver detection operation or to begin loopback.
TXIDLE	I	Forces Philips PHY TX output to electrical idle when asserted in all power states.
TXCOM	I	When high, sets the running disparity to negative. Used when transmitting the compliance pattern.
RXPOL	I	Active high, signals the PHY to perform a polarity inversion on the receive data.
RESET_N	I	Output Active low PHY reset from FPGA.
PWRDWN1, PWRDWN0	I	Power up or down the transceiver. Power states [1:0]: 00 - P0, normal operation 01 - P0s, low recovery time (2.5 µs), power saving state 10 - P1, longer recovery time (64us max), lower power state 11 - P2, lowest power state.
RXVALID	O	Indicates symbol lock and valid data on RXDATA and RXDATAK
PHYSTATUS	O	Communicates completion of several Philips PHY functions, including power management state transitions, and receiver detection.
RXIDLE	O	Indicates receiver detection of an electrical idle. This is an asynchronous signal.
RXSTATUS2, RXSTATUS1, RXSTATUS0	O	Encodes receiver status and error codes for the received data stream and receiver detection. Encoding [2:1] 000 - Received data OK 001 - 1 SKP added 010 - 1 SKP removed 011 - Receiver detected 100 - 8B/10B decode error 101 - Elastic Buffer overflow 110 - Elastic Buffer underflow 111 - Receive disparity error

***** The direction is defined from the perspective of the PHY.

3.3 Xilinx PCIe Core

The product name of this core is Xilinx PCIe Physical Interface for PCI Express (PIPE) Endpoint 1-Lane core.

This core implements a high performance serial Interconnect intellectual property, which is optimized for the use with the Spartan-3/3E device families of Xilinx, as indicated in table 3.3.

The table also shows some of the core's specifications, such as the complexity when targets the XC3S1000-4 device of the Xilinx Spartan-3 family.

Table 3.3 *- PCIe Core Specification [XDS321]*

LogiCORE™ Facts				
Core Specifics				
Supported Device Family	Spartan™-3, Spartan-3E			
Resource Used[1] XC3S1000-4	Product	LUT	FF	Block RAM
	1-Lane Endpoint	5408-5708[(2)]	3920-4017[2]	8
Special Features	Digital Clock Manager Block RAM			

1. The precise number of slices depends on the user configuration of the interface and the level of resource sharing with adjacent logic. 2. This range indicates resources used for a 2BAR–7BAR implementation

This core shows a compliance with the PCI Express Base Specification v1.1, and a backward compatibility with the existing PCI software model.

3.3.1 Features and Applications

Several features make the Xilinx PCIe core one of the most desirable core in implementing PCIe based serial interconnects with Xilinx FPGAs. The most important features are listed here [XDS321]:

- Flexibility, scalability, and reliability, due to its compliance to the PCIe base specification and compatibility with the PCI software model.
- Meeting the PCIe transaction ordering rules.
- Implementing 32-bit datapaths.
- Employment of six programmable and configurable Base Address Registers (BARs) and an expansion ROM BAR.
- Providing error and detection of corrupted packets.
- Supporting Message Signaling Interrupt (MSI).
- Providing of PCI/PCIe power management functions:
  - Active State Power Management (ASPM)
  - Programmed Power Management (PPM).
- Offering a two-chip solution with the Philips PX1011A PHY, to demonstrate a capable transceiver to provide a high data rate of 2.5 Gbps, buffering and clock compensation, clock and data recovery as well as 8b/10b encoding and decoding. Figure 3.10 shows the two-chip low cost solution.

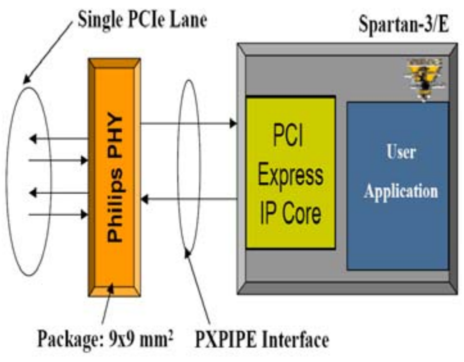


Figure 3.10 *- Two-chip solution [XP05]*

- Supporting a maximum transaction payload of up to 512 bytes.
- Supporting packet-based full-duplex communication and back-to-back transactions.
- Enabling of data flow control.
- Full configurability using the Xilinx CORE Generator.

The PCIe PIPE Endpoint can be used in many applications. For instance, it can be used in test and medical imaging equipments, graphic boards, data communication, telecommunication networks, chip to chip communications and server applications.

3.3.2 Block Diagram and Functionality

A top-level functional block diagram of the Xilinx PCIe core is shown in figure 3.11. This core consists of four different functional blocks, namely the *Transaction Layer Module (TLM), the Data Link Layer Module (LLM), the Physical Layer Module (PLM) and the Configuration Management Module (CMM).*

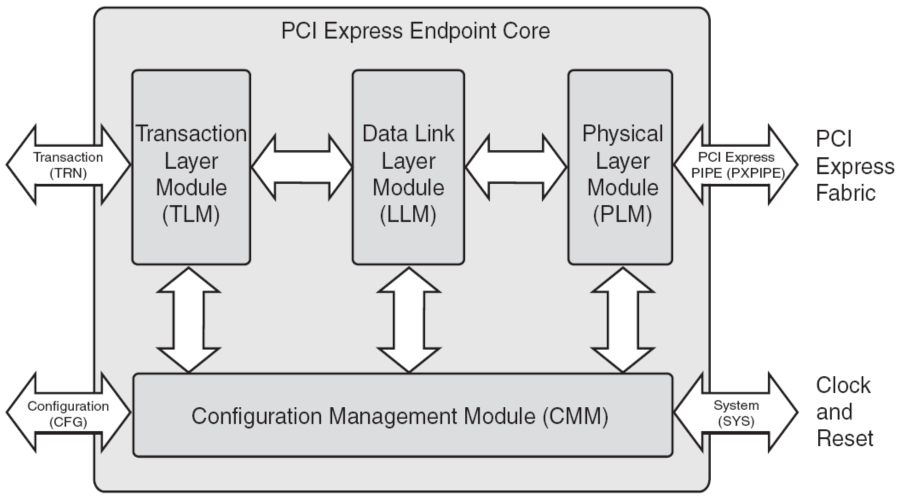

PCI Express Endpoint Core
Transaction (TRN)
Transaction Layer Module (TLM)
Data Link Layer Module (LLM)
Physical Layer Module (PLM)
PCI Express PIPE (PXPIPE)
PCI Express Fabric
Configuration (CFG)
Configuration Management Module (CMM)
System (SYS)
Clock and Reset

***Figure 3.11** - Top-level functional blocks diagram and Interfaces of Xilinx PCIe Core [XUG167]*

Each of the four modules is further divided into receive and transmit parts. As mentioned previously, the received part processes the incoming TLPs while the transmit part processes the TLPs to be transmitted. Theses four modules implement the functionality of each layer of the PCIe architecture.

The *transaction layer module* generates the transaction layer packets (TLPs), which are used for the purpose of transactions communication, such as Read and Write memory transactions.

The transaction layer of the PCIe PIPE core uses a pipelined, full split-transaction protocol, employs flow control of TLPs in addition to other features.

The main purpose of the *Data Link Layer Module* is to implement the functionality of the Data Link Layer in providing a reliable transport of the TLPs across the PCIe link. It does this by detecting and recovering errors and generating Data Link Layer Packets (DLLP).

The *Physical Layer Module* interfaces the Data Link layer module from one side, while interfaces the Philips PHY through the PXPIPE from the other side. It is responsible for initializing the physical link and scrambling /de-scrambling the transmitted/received data.

The *Configuration Management Module* enables the communication between the different modules of the core to support the generation and reception of TLPs. It implements configuration space registers, which support the PCI configuration space as well as a new PCIe extended space.

Programmed Power Management (PPM) and Active state Power Management (ASPM) are the power management functions supported by this configuration management module. This module also provides error reporting and tracking. It receives Configuration Reads and Writes, and transmits a completion with or without data. In addition to that, Message Signaling Interrupt is implemented by this module.

Figure 3.12 shows the PCIe configuration space. A type 0 configuration space is implemented in this module, consisting of 64 bytes (the type 0 configuration space header) plus 192 bytes, used for the purpose of the PCIe extended capabilities. A new operating system is needed to access theses extended PCIe capabilities.

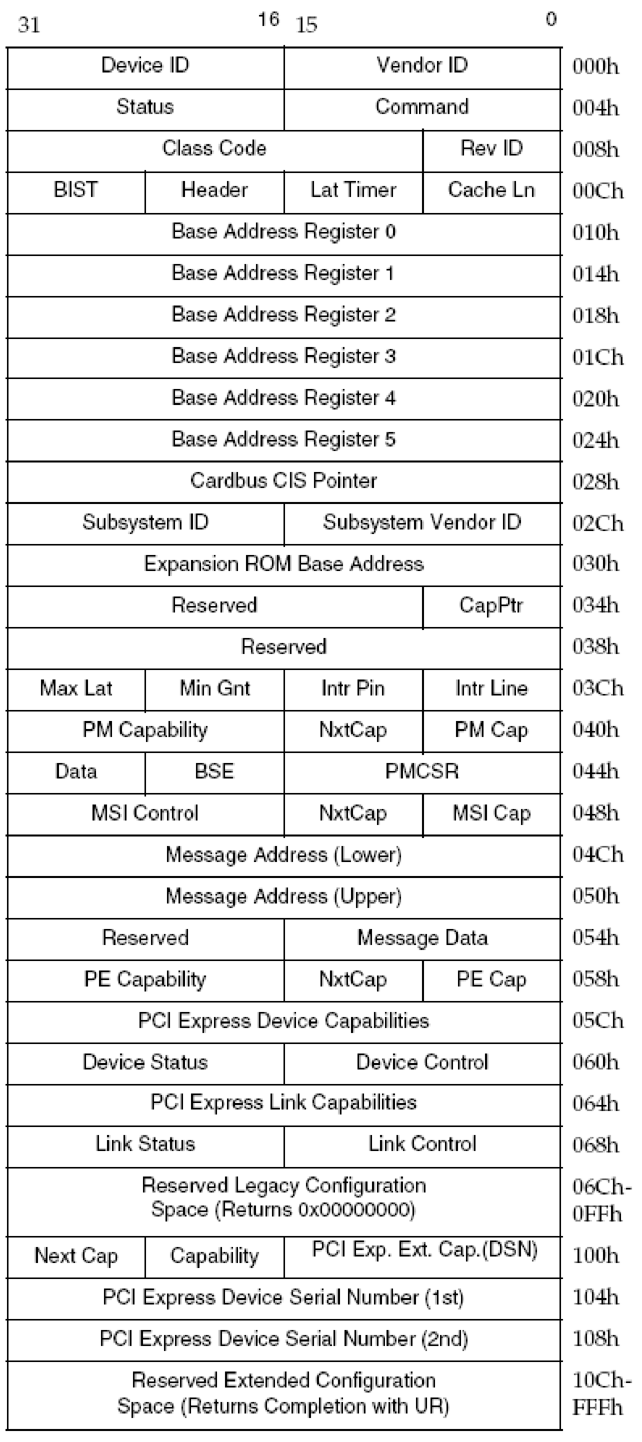
Within this configuration space, the Base Address Registers (BARs) exist. The PCIe core uses the addresses stored in these registers to route TLPs. When the information in the Header of a TLP indicates that address routing is to be used, then the PCIe core compares the address in the TLP Header with the implemented BARs. It claims and forwards the TLP to the user logic, if it founds a match in the address. Otherwise, it blocks this TLP. The initialization of these BARs will be explained later in this chapter.

3.3.3 Core Interfaces

The PCIe core poses four different interfaces: System interface (SYS), PCI Express PIPE (PXPIPE), Transaction interface (TRN) and Configuration interface (CFG).

System Interface (SYS)

For a hard reset of the core and the external physical layer PHY, a system reset signal (sys_reset_n) is used as an asynchronous input to the core. This reset signal is an active low. Practically, this signal is connected to a sideband reset signal.



31	16	15	0	
Device ID		Vendor ID		000h
Status		Command		004h
Class Code			Rev ID	008h
BIST	Header	Lat Timer	Cache Ln	00Ch
Base Address Register 0				010h
Base Address Register 1				014h
Base Address Register 2				018h
Base Address Register 3				01Ch
Base Address Register 4				020h
Base Address Register 5				024h
Cardbus CIS Pointer				028h
Subsystem ID		Subsystem Vendor ID		02Ch
Expansion ROM Base Address				030h
Reserved			CapPtr	034h
Reserved				038h
Max Lat	Min Gnt	Intr Pin	Intr Line	03Ch
PM Capability		NxtCap	PM Cap	040h
Data	BSE	PMCSR		044h
MSI Control		NxtCap	MSI Cap	048h
Message Address (Lower)				04Ch
Message Address (Upper)				050h
Reserved		Message Data		054h
PE Capability		NxtCap	PE Cap	058h
PCI Express Device Capabilities				05Ch
Device Status		Device Control		060h
PCI Express Link Capabilities				064h
Link Status		Link Control		068h
Reserved Legacy Configuration Space (Returns 0x00000000)				06Ch-0FFh
Next Cap	Capability	PCI Exp. Ext. Cap.(DSN)		100h
PCI Express Device Serial Number (1st)				104h
PCI Express Device Serial Number (2nd)				108h
Reserved Extended Configuration Space (Returns Completion with UR)				10Ch-FFFh

Figure 3.12 *- PCIe Configuration Space [XUG167]*

PCI Express PIPE (PXPIPE)

In tables 3.4 to 3.8, the interfaces of this core to the discrete PHY from Philips are illustrated.

Table 3.4 *- PXPIPE Transmit Data Interface Signals [XUG167]*

Signal	I/O	Description
TXDATA[7:0]	O	8-bit transmit data from the FPGA to the Philips PHY.
TXDATAK	O	Data/Control for the symbols of transmit data. A value of 0 indicates a data byte; a value of 1 indicates a control byte.

Table 3.5 *- PXPIPE Receive Data Interface Signals [XUG167]*

Signal	I/O	Description
RXDATA[7:0]	I	8-bit receive data from the Philips PHY to the FPGA.
RXDATAK	I	Data/Control for the symbols of received data. A value of 0 indicates a data byte; a value of 1 indicates a control byte.

Table 3.6 *- Clock and Reference Signals [XUG167]*

Signal	I/O	Description
TXCLK	O	Source synchronous 250 MHz clock (from FPGA) for transmit clock from MAC input. All the data and the input signals to the PHY are synchronized to this clock
RXCLK	I	Source synchronous 250 MHz clock (to FPGA) for received data bound for the MAC output
fast_train_simulation_only	I	Used for Simulation Only, active high. Causes link training time-out counters to be smaller than normal for faster link training.
two_plm_auto_config	I	Used for Simulation Only, active high. PCI Express specification non compliant link train with another similarly configured core.

Table 3.7 *- PXPIPE Command Interface Signals [XUG167]*

Signal	I/O	Description
TXDETECTRX_ LOOPBACK	O	Enable the Philips PHY to begin a receiver detection operation or to begin loopback.
TXELECIDLE	O	Forces Philips PHY TX output to electrical idle when asserted in all power states.
TXCOMPLIANCE	O	When high, sets the running disparity to negative. Used when transmitting the compliance pattern.
RXPOLARITY	O	Active high, signals the PHY to perform a polarity inversion on the receive data.
RESETN	O	Output Active low PHY reset from FPGA.
POWERDOWN[1:0]	O	Power up or down the transceiver. Power states: 00 - P0, normal operation 01 - P0s, low recovery time latency, power saving state 10 - P1, longer recovery time (64us max) latency, lower power state 11 - Reserved for P2, lowest power state.

Table 3.8 *- PXPIPE Status Interface Signals [XUG167]*

Signal	I/O	Description
RXVALID	I	Input Indicates symbol lock and valid data on RxData and RxDataK
PHYSTATUS	I	Used to communicate completion of several Philips PHY functions including power management state transitions, and receiver detection.
RXELECIDLE	I	Indicates receiver detection of an electrical idle. This is an asynchronous signal.
RXSTATUS[2:0]	I	Encodes receiver status and error codes for the received data stream and receiver detection. 000 - Received data OK 001 - 1 SKP added 010 - 1 SKP removed 011 - Receiver detected 100 - 8B/10B decode error 101 - Elastic Buffer overflow 110 - Elastic Buffer underflow 111 - Receive disparity error

The core interfaces the user application logic with different signals, to enable the transmission and reception of TLPs. These interfaces are divided into three sections: the Common Transaction Interface, the Transmit Transaction Interface and the Receive Transaction Interface. The following is an explanation of each section.

Common Transaction Interface Signals

trn_clk: *Transaction Clock*: An output 62.50 MHz clock signal. All transaction and configuration interfaces are synchronized to the rising edge of this clock. This signal is not available whenever the sys_reset_n is asserted.

trn_reset_n: *Transaction Reset*: An active low output reset signal. This signal is used to reset user logic, which interfaces with the transaction and configuration signals. The deassertion of this signal is synchronized to trn_clk.

trn_lnk_up_n: *Transaction Link Up*: An active low output signal. This signal is activated, when the PCIe core and the upstream PCIe link are ready and can start exchanging packets, and deactivated when they are trying to establish communication or when data is lost because of some error on the link.

Transmit Transaction Interface Signals

These are the interfaces the core needs to transmit TLPs across the PCIe link.

trn_tsof_n: *Transmit Start-of-Frame (SOF):* An active low input signal that indicates the start of a packet.

trn_teof_n: *Transmit End-of-Frame (EOF):* An active low input signal that signals the end of a packet.

trn_td [31:0]: *Transmit Data:* 32-bit input packet data to be transferred to the transaction layer of the core.

trn_terrfwd_n: *Transmit Error Forward*: An active low input signal. This signal is used to indicate that the associated packet is error-poisoned.

trn_tsrc_rdy_n: *Transmit Source Ready*: An active low signal that indicates the availability of valid data from the user logic application.

trn_tdst_rdy_n: *Transmit Destination Ready*: An active low signal that indicates that the PCIe core is ready to receive data on trn_td [31:0]**.** The simultaneous assertion of trn_tsrc_rdy_n and trn_tdst_rdy_n represents a successful transfer of one DWORD of data on trn_td [31:0].

trn_tsrc_dsc_n: *Transmit Source Discontinue:* An active low signal indicates that the user application is discarding the current packet.

trn_tbuf_av [3:0]: *Transmit Buffers Available*: Number of transmit buffers available in the core. The maximum number is 6. Each transmit buffer can hold one packet with up to 512 bytes of payload.

Receive Transaction Interface Signals

These are the interfaces the core needs to receive TLPs across the PCIe link. Tables 3.9 and 3.10 list and explain these signals.

Table 3.9 *- Xilinx PCIe PIPE Endpoint Core Transaction Receive Interfaces [XUG167]*

Signal	I/O	Description
PCIE_TRN_RSOF_N	I	**Receive Start-of-Frame** (**SOF**): Signals the start of a packet. Active low.
PCIE_TRN_REOF_N	I	**Receive End-of-Frame** (**EOF**): Signals the end of a packet. Active low.
PCIE_TRN_RD[31:0]	I	**Receive Data:** Packet data being received.
PCIE_TRN_RERRFWD_N	I	**Receive Error Forward:** Marks the current packet in progress as error-poisoned. Asserted by the core at EOF. Active low.
PCIE_TRN_RSRC_RDY_N	I	**Receive Source Ready:** Indicates that the PCI Express Endpoint core is presenting valid data on trn_rd [31:0]. Active low.
PCIE_TRN_RDST_RDY_N	O	**Receive Destination Ready:** Indicates that the User Application is ready to accept data on **PCIE_TRN_RD [31:0]**. Active low. The simultaneous assertion of **PCIE_TRN_RSRC_RDY_N** and **PCIE_TRN_RDST_RDY_N** marks the successful transfer of one DWORD of data on **PCIE_TRN_RD [31:0]**.
PCIE_TRN_RSRC_DSC_N	O	**Receive Source Discontinue:** Indicates that the PCI Express Endpoint core is aborting the current packet. Asserted when the physical link is going into reset. Active low.
PCIE_TRN_RNP_OK_N	O	**Receive Non-Posted OK:** The User Application asserts this whenever it is ready to accept a Non-Posted Request packet. This allows Posted and Completion packets to bypass Non-Posted packets in the inbound queue if necessitated by the User Application. Active low. When the User Application approaches a state where it is unable to service Non-Posted Requests; it must deassert **PCIE_TRN_RNP_OK_N** after SOF of the second-to-last Non-Posted packet it can accept.

trn_rbar_hit_n[6:0]	I	**Receive BAR Hit:** Indicates BAR(s) targeted by the current receive transaction. Active low.
		trn_rbar_hit_n[0] => BAR0
		trn_rbar_hit_n[1] => BAR1
		trn_rbar_hit_n[2] => BAR2
		trn_rbar_hit_n[3] => BAR3
		trn_rbar_hit_n[4] => BAR4
		trn_rbar_hit_n[5] => BAR5
		trn_rbar_hit_n[6] => Expansion ROM Address
		Note that, if two BARs are configured into a single 64-bit address, both corresponding trn_rbar_hit_n bits will be asserted.

Table 3.9 - *Xilinx PCIe PIPE Endpoint Core Transaction Receive Interfaces (Cont.) [XUG167]*

Signal	I/O	Description
TRN_RFC_PH_AV[7:0]	I	**Receive Posted Header Flow Control Credits Available:** The number of Posted Header FC credits available to the remote link partner.
TRN_RFC_PD_AV[11:0]	I	**Receive Posted Data Flow Control Credits Available:** The number of Posted Data FC credits available to the remote link partner.
TRN_RFC_NPH_AV[7:0]	I	**Receive Non-Posted Header Flow Control Credits Available:** The number of Non-Posted Header FC credits available to the remote link partner.
TRN_RFC_NPD_AV[11:0]	I	**Receive Non-Posted Data Flow Control Credits Available:** The number of Non-Posted Data FC credits available to the remote link partner.
TRN_RFC_CPLH_AV[7:0]	I	**Receive Completion Header Flow Control Credits Available:** The number of Completion Header FC credits available to the remote link partner. Note that this value and **PCIE_TRN_RFC_CPLd_AV [11:0]** are hypothetical quantities reflecting credit availability that would be advertised to the remote link partner if the PIPE core were not required to advertise infinite Completion credits.
TRN_RFC_NPH_AV[7:0]	I	**Receive Non-Posted Header Flow Control Credits Available:** The number of Non-Posted Header FC credits available to the remote link partner.
TRN_RFC_NPD_AV[11:0]	I	**Receive Non-Posted Data Flow Control Credits Available:** The number of Non-Posted Data FC credits available to the remote link partner.
TRN_RFC_CPLH_AV[7:0]	I	**Receive Completion Header Flow Control Credits Available:** The number of Completion Header FC credits available to the remote link partner. Note that this value and **PCIE_TRN_RFC_CPLd_AV [11:0]** are hypothetical quantities reflecting credit availability that would be advertised to the remote link partner if the PIPE core were not required to advertise infinite Completion credits.
TRN_RFC_NPH_AV[7:0]	I	**Receive Non-Posted Header Flow Control Credits Available:** The number of Non-Posted Header FC credits available to the remote link partner.
TRN_RFC_NPD_AV[11:0]	I	**Receive Non-Posted Data Flow Control Credits Available:** The number of Non-Posted Data FC credits available to the remote link partner.
TRN_RFC_CPLd_AV[11:0]	I	**Receive Completion Data Flow Control Credits Available:** The number of Completion Data FC credits available to the remote link partner.

Configuration Interface

The core enables the user to access its configuration space. In this version of the core a writing access of the registers is not supported. Tables 3.11 to 3.13 describe these interfaces.

Table 3.10 **-** *Xilinx PCIe Core Configuration Interfaces [XUG167]*

Signal	I/O	Description
CFG_DO[31:0]	I	**Configuration Data Out**: This is a 32-bit data output port used to obtain read data from the configuration space inside the PCI Express endpoint.
CFG_RD_WR_DONE_N	I	**Configuration Read Write Done:** This active-low read-write done signal indicates a successful completion of the user configuration register access operation. For a user configuration register read operation, the signal validates the cfg_do [31:0] data-bus value. For a user configuration register write operation, the assertion signals the completion of a successful write operation. *Not supported for write operations.*
CFG_DI[31:0]	O	**Configuration Data In**: This is a 32-bit data input port used to provide write data to the configuration space inside the core. *Not supported.*
CFG_DWADDR[9:0]	O	**Configuration DWORD Address**: This is a 10-bit address input port used to provide a configuration register DWORD address during configuration register accesses.
CFG_WR_EN_N	O	**Configuration Write Enable**: This is the active low write enable for configuration register access. *Not supported.*
CFG_RD_EN_N	O	**Input Description:** Configuration Read Enable: This is the active low read enable for configuration register access.
CFG_INTERRUPT_N	O	**Configuration Interrupt**: This is the active low interrupt request signal. The User Application may assert this to cause appropriate interrupt messages to be transmitted by the PCI Express PIPE core.
CFG_INTR_RDY_N	I	**Configuration Interrupt Ready:** This is the active low interrupt grant signal. The assertion on this signal indicates that the PIPE core has successfully transmitted the appropriate interrupt message.
CFG_TURNOFF_OK_N	O	**Configuration Turnoff OK**: This is the active low power turn-off ready signal. The User Application may assert this to notify the PCI Express PIPE core that it is safe for power to be removed.
CFG_TO_TURNOFF_N	I	**Configuration To Turnoff**: This output signal notifies the user that a PME_TURN_Off message has been received and the CMM will start polling the **PCIE_CFG_TURNOFF_OK_N** input coming in from the user. Once **PCIE_CFG_TURNOFF_OK_N** is asserted, CMM sends a PME_To_Ack message to the upstream device.

***Table 3.10 -** Xilinx PCIe PIPE Endpoint Core Configuration Interfaces (Cont.) [XUG167]*

Signal	I/O	Description
CFG_BYTE_EN_N[3:0]	I	**Configuration Byte Enable**: This is the active low byte enables for configuration register access signal. *Not supported.*
CFG_ERR_ECRC_N	O	**ECRC Error Report**: The user can assert this signal to report an ECRC error (end-to-end CRC).
CFG_ERR_CPL_TIMEOUT_N	O	**Configuration Error Completion Timeout**: The user can assert this signal to report a completion timed out.
CFG_ERR_CPL_TIMEOUT_N	O	**Configuration Error Completion Timeout**: The user can assert this signal to report a completion timed out.
CFG_ERR_CPL_ABORT_N	O	**Configuration Error Completion Aborted**: The user can assert this signal to report that a completion was aborted.
CFG_ERR_CPL_UNEXPECT_N	O	**Configuration Error Completion Unexpected**: The user can assert this signal to report that an unexpected completion was received.
CFG_ERR_CPL_POSTED_N	O	**Configuration Error Posted**: This signal is used to further qualify any of the PCIE_CFG_ERR_* input signals. When this input is asserted concurrently with one of the other signals, it indicates that the transaction which caused the error was a posted transaction.
CFG_ERR_COR_N	O	**Configuration Error Correctable Error**: The user can assert this signal to report that a correctable error was detected.
CFG_ERR_UR_N	O	**Configuration Error Unsupported Request**: The user can assert this signal to report that an unsupported request was received.
CFG_ERR_TLP_CPL_HEADER[47:0]	O	**Configuration Error TLP Completion Header**: This input to the core accepts the header information from the user when an error is signaled. This information is required so that the core can issue a correct completion, if required.
CFG_BUS_NUMBER[7:0]	I	**Configuration Bus Number**: This output provides the assigned bus number for the device. The user may require this information to form packets.
CFG_DEVICE_NUMBER[4:0]	I	**Configuration Device Number**: This output provides the assigned device number for the device. The user may require this information to form packets.
CFG_FUNCTION_NUMBER[2:0]	I	**Configuration Function Number**: This output provides the function number for the device. The user may require this information to form packets.
CFG_PCIE_LINK_STATE_N[2:0]	I	**PCI Express Link State**: This one-hot encoded bus reports the PCI Express Link State Information to the user. 110b - PCIExpress Link State is "L0" 101b - PCIExpress Link State is "L0s" 011b - PCIExpress Link State is "L1" 111b - PCIExpress Link State is "under transition"

***Table 3.10** - Xilinx PCIe PIPE Endpoint Core Configuration Interfaces (Cont.) [XUG167]*

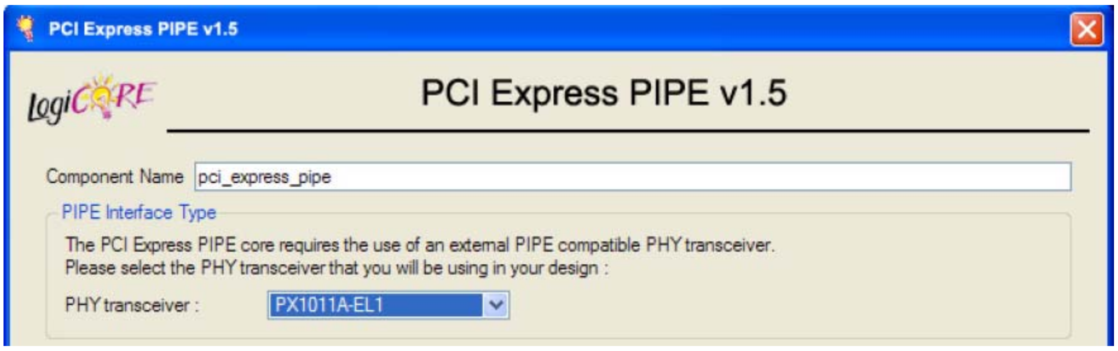
Signal	I/O	Description
CFG_STATUS [15:0]	I	**Configuration Status**: PCI status register output
CFG_COMMAND [15:0]	I	**Configuration Command**: PCI command register output.
CFG_DSTATUS [15:0]	I	**Configuration Device Status**: PCI Express PIPE device status register output.
CFG_DCOMMAND [15:0]	I	**Configuration Device Command**: PCI Express PIPE device command register output.
CFG_LSTATUS [15:0]	I	**Configuration Link Status**: PCI Express PIPE link status register output.
CFG_LCOMMAND [15:0]	I	**Configuration Link Command**: PCI Express PIPE link command register output.
CFG_PM_WAKE_N	O	**Configuration Power Management Wake**: A one-clock cycle active low assertion on this signal enables the core to generate and send a Power Management Wake event to the upstream link partner. **NOTE**: The user is required to assert this input only under stable link conditions as reported on the **PCIE_CFG_PCIE_LINK_STATE_N[2:0]**bus. Assertion of this signal when the PCI Express link is under transition will result in incorrect behavior on the PCI Express link.
CFG_DSN [63:0]	O	**Configuration Device Serial Number**: Serial Number Register fields of the PCI Express Device Serial Number extended capability.
CFG_PCIE_LINK_STATE_N [2:0]	I	**PCI Express Link State**: This one-hot encoded bus reports the PCI Express Link State Information to the user. 110b - PCIExpress Link State is "L0" 101b - PCIExpress Link State is "L0s" 011b - PCIExpress Link State is "L1" 111b - PCIExpress Link State is "under transition"
CFG_LSTATUS [15:0]	I	**Configuration Link Status**: PCI Express PIPE link status register output.
CFG_LCOMMAND [15:0]	I	**Configuration Link Command**: PCI Express PIPE link command register output.
CFG_PM_WAKE_N	O	**Configuration Power Management Wake**: A one-clock cycle active low assertion on this signal enables the core to generate and send a Power Management Wake event to the upstream link partner. **NOTE**: The user is required to assert this input only under stable link conditions as reported on the **PCIE_CFG_PCIE_LINK_STATE_N[2:0]**bus. Assertion of this signal when the PCI Express link is under transition will result in incorrect behavior on the PCI Express link.
CFG_DSN [63:0]	O	**Configuration Device Serial Number**: Serial Number Register fields of the PCI Express Device Serial Number extended capability.

3.3.4 Core Generation and Configuration

The PCIe core is fully configurable and highly customizable. The Xilinx CORE Generator was used to generate and customize this core.

The following figures show some of the important steps in generating and configuring the PCIe core.[1]

In Figure 3.13, the component name is given, which was used as a base name of the output files generated for the core. The physical interface is indicated as well.


PCI Express PIPE v1.5

PCI Express PIPE v1.5

Component Name: pci_express_pipe

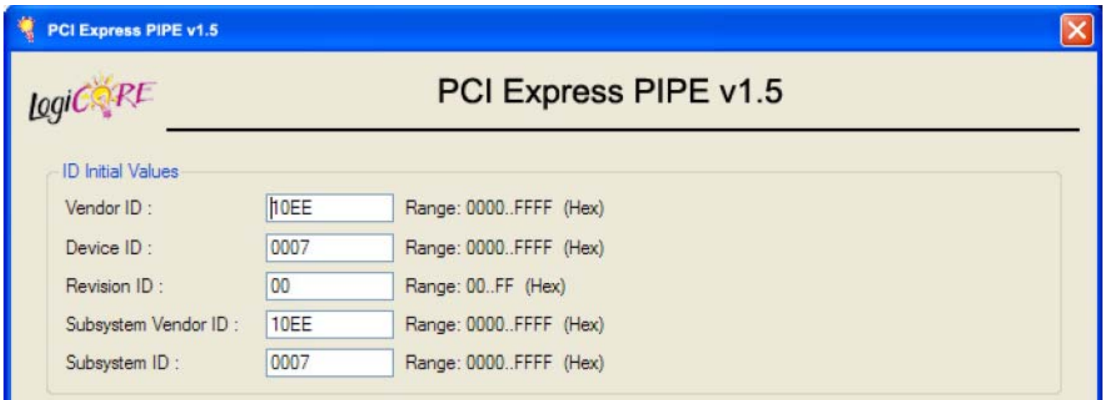
PIPE Interface Type

The PCI Express PIPE core requires the use of an external PIPE compatible PHY transceiver.
Please select the PHY transceiver that you will be using in your design :

PHY transceiver : PX1011A-EL1

Figure 3.13 - *PCIe Component Name [XUG167]*

Figure 3.14 shows the ID initial values screen where different parameters can be set. The Vendor ID identifies the manufacture of the device or application. A default value of 10EE refers to Xilinx. A Device ID can also be set to identify the application.


PCI Express PIPE v1.5

PCI Express PIPE v1.5

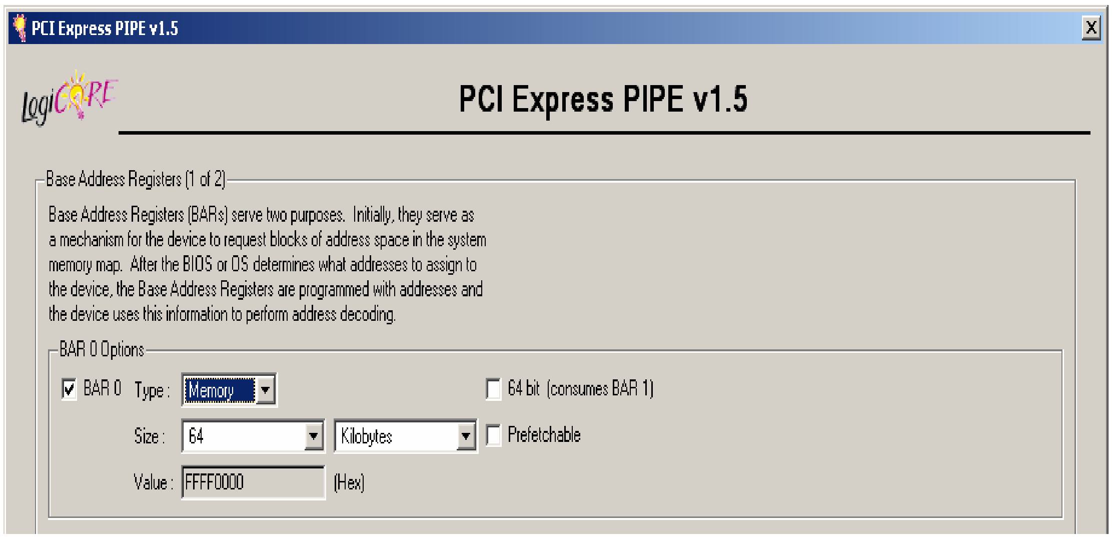
ID Initial Values

Vendor ID :	10EE	Range: 0000..FFFF (Hex)
Device ID :	0007	Range: 0000..FFFF (Hex)
Revision ID :	00	Range: 00..FF (Hex)
Subsystem Vendor ID :	10EE	Range: 0000..FFFF (Hex)
Subsystem ID :	0007	Range: 0000..FFFF (Hex)

Figure 3.14 - *PCIe ID Initial Values [XUG167]*

[1] For more detailed steps, refer to the user guide [XUG167].

In Figure 3.15, the configuration of the Base address registers space is shown. The core was configured to support memory mapped space. Base Address Registers (BARs) are used for two purposes. Firstly, the Endpoint device through these BARs can request blocks of addresses in the system memory map. Secondly, after the operating system or Basic Input Output System (BIOS) defines the addresses to be assigned to the Endpoint device, the BARs are programmed with these addresses and the Endpoint uses this information for the address decoding and recognizing of TLPs.


PCI Express PIPE v1.5

Base Address Registers (1 of 2)

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

[✓] BAR 0 Type : Memory
[ ] 64 bit (consumes BAR 1)
Size : 64 Kilobytes
[ ] Prefetchable
Value : FFFF0000 (Hex)

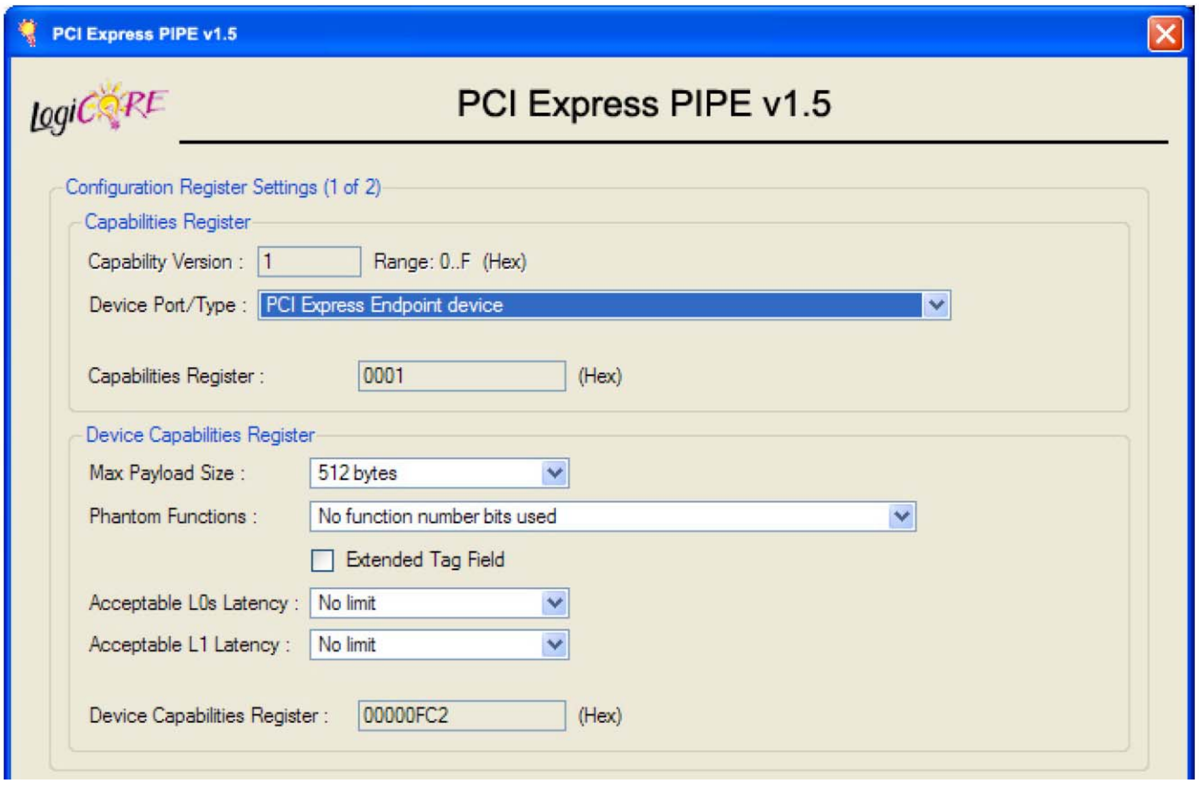
***Figure 3.15** - PCIe Base Address Registers (BARs) Configuration*

The core can be configured to support up to six 32-bit BARs or three 64-bit BARs. Once the core receives a TLP, it compares the address included in the header of the TLP with the address defined by the BAR. If the address matches within the range, the core presents the data at the Transaction interface for the user logic. The data will be blocked otherwise.

The unused BARs are disabled, and the logic that enables their usage is not implemented to reduce the complexity.

In figure 3.16 the capabilities register setting is shown. Here, the PCIe logical device type is determined. The only functionality supported by Xilinx PCIe core is to have it as PCI Express Endpoint device.

The figure also shows the setting of the device capabilities register. In this register, the Maximum Payload size can be configured. This core can support up to 512 bytes as payload to be sent with the packet.



Configuration Register Settings (1 of 2)

Capabilities Register

Capability Version : 1 Range: 0..F (Hex)

Device Port/Type : PCI Express Endpoint device

Capabilities Register : 0001 (Hex)

Device Capabilities Register

Max Payload Size : 512 bytes

Phantom Functions : No function number bits used

☐ Extended Tag Field

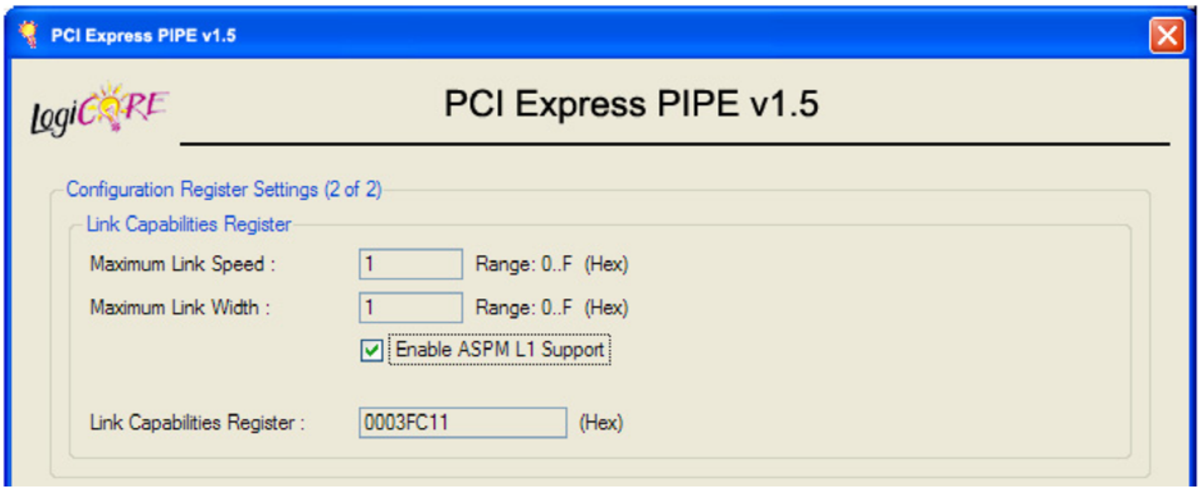
Acceptable L0s Latency : No limit

Acceptable L1 Latency : No limit

Device Capabilities Register : 00000FC2 (Hex)

Figure 3.16 *- PCIe Capabilities and Device Capabilities Register Configuration [XUG167]*

The configuration of the Link Capabilities Register is depicted in figure 3.17. Illustrated are the link speed and width which are set to 1 to indicate a x1 PCIe link, which has a data transfer rate of 2.5 Gbps.



Configuration Register Settings (2 of 2)

Link Capabilities Register

Maximum Link Speed : 1 Range: 0..F (Hex)

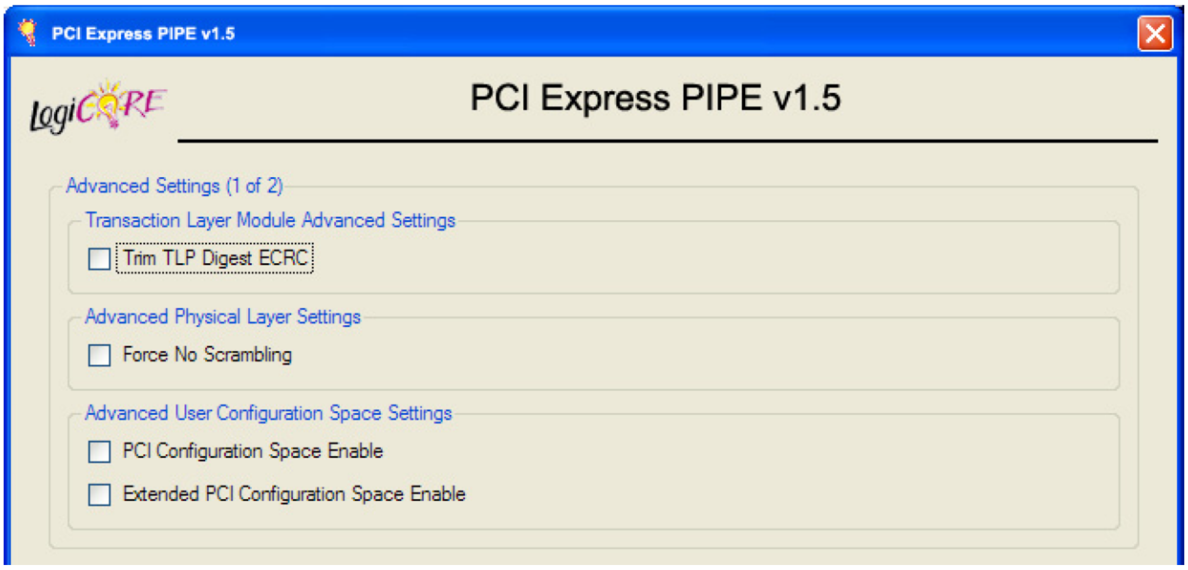
Maximum Link Width : 1 Range: 0..F (Hex)

☑ Enable ASPM L1 Support

Link Capabilities Register : 0003FC11 (Hex)

Figure 3.17 *- PCIe Link Capabilities Register Configuration [XUG167]*

Some of the advanced settings are shown in figure 3.18. For the transaction layer, selecting Trim TLP Digest ECRC will cause the core to drop out any TLP digest of the incoming TLPs before forwarding it to the user logic. Scrambling data TLPs to be transmitted can be deselected in the logical sublayer of the core's physical layer. Enabling and disabling of PCI configuration space is also possible. Furthermore, the extended PCI Configuration space can be enabled or left disabled.


PCI Express PIPE v1.5

Advanced Settings (1 of 2)

Transaction Layer Module Advanced Settings

☐ Trim TLP Digest ECRC

Advanced Physical Layer Settings

☐ Force No Scrambling

Advanced User Configuration Space Settings

☐ PCI Configuration Space Enable

☐ Extended PCI Configuration Space Enable

Figure 3.18- *PCIe Advanced Settings [XUG167]*

3.4 Xilinx Microblaze Soft Processor Core

Xilinx Microblaze processor is a soft IP core optimized for the implementation in Xilinx FPGAs. This core implements a 32-bit reduced instruction set computer. It includes thirty-two 32-bit general purpose registers and implements a 32-bit instruction word with three operands and two addressing modes. This core uses 32-bit address buses.

The Microblaze "soft" processor is built using the FPGA's logic, unlike the "hard" processor which is built using dedicated silicon. It is configurable for the optimal use of the designer.

3.4.1 Microblaze Block Diagram

Figure 3.19 shows the block diagram of this soft core. In the figure both the fixed and the configurable features of this processor are shown. The core uses the Harvard structure by dedicating two different paths for the instruction and the data (as illustrated in the figure).

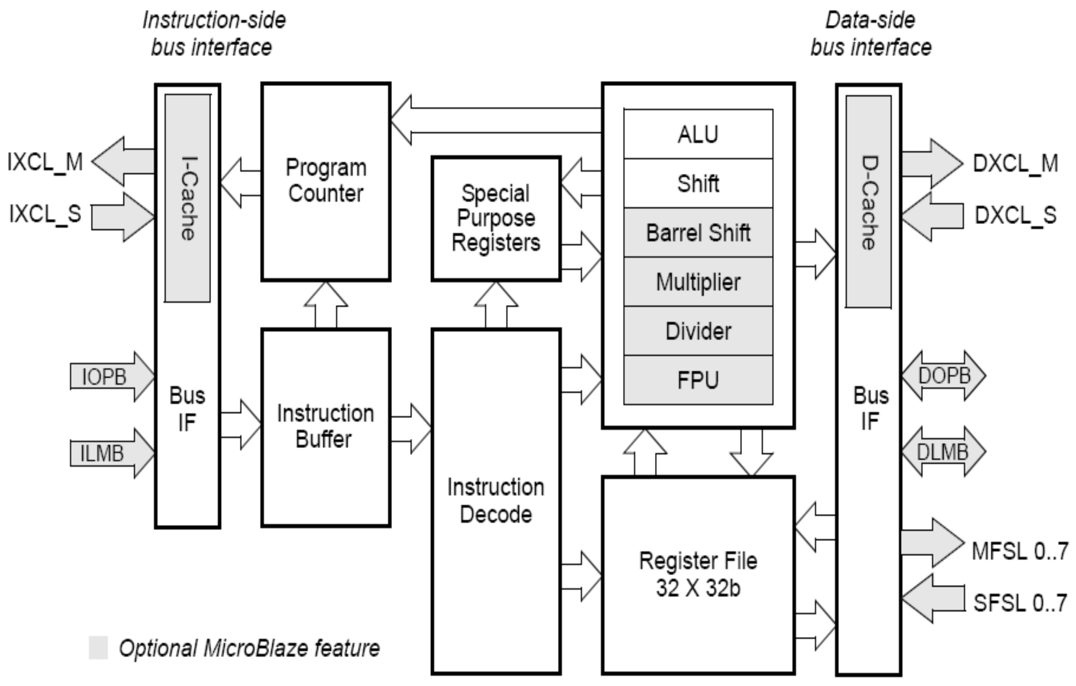


Figure 3.19 *- Microblaze Block Diagram [XUG081]*

3.4.2 Microblaze Interfaces

The Microblaze core has many interfaces. The following is a list of them [XUG081]:

DOPB:	Data interface, On-chip Peripheral Bus
DLMB:	Data interface, Local Memory Bus (BRAM only)
IOPB:	Instruction interface, On-chip Peripheral Bus
ILMB:	Instruction interface, Local Memory Bus (BRAM only)
MFSL 0..7:	FSL master interfaces
SFSL 0..7:	FSL slave interfaces
IXCL:	Instruction side Xilinx CacheLink interface (FSL master/slave pair)
DXCL:	Data side Xilinx CacheLink interface (FSL master/slave pair)
Core:	Miscellaneous signals for: clock, reset, debug, and trace

For the purpose of this work, the LMB and OPB interfaces will be considered[1].

[1] Refer to Xilinx Microblaze Processor Reference Guide [XUG081] for other interfaces and details.

3.4.3 Local Memory Bus (LMB)

The main purpose of the LMB is to access an on-chip Block RAM (BRAM) peripheral, in a single clock cycle. The Microblaze core has two LMB interfaces: The Data Local Memory Bus (DLMB), which provides an interface to the data RAM and the Instruction Local Memory Bus (ILMB), which interfaces the instruction RAM.

The BRAM Block is a dual port configurable memory that can be attached to the Microblaze ILMB and DLMB ports in conjunction with the Local Memory Bus (LMB) Block RAM (BRAM) Interface Controller as illustrated in figure 3.20. This BRAM is initialized with the application program to be executed by the Microblaze.

The dual port feature of the BRAM enables a concurrent access of the ILMB and DLMB sides in a single cycle.

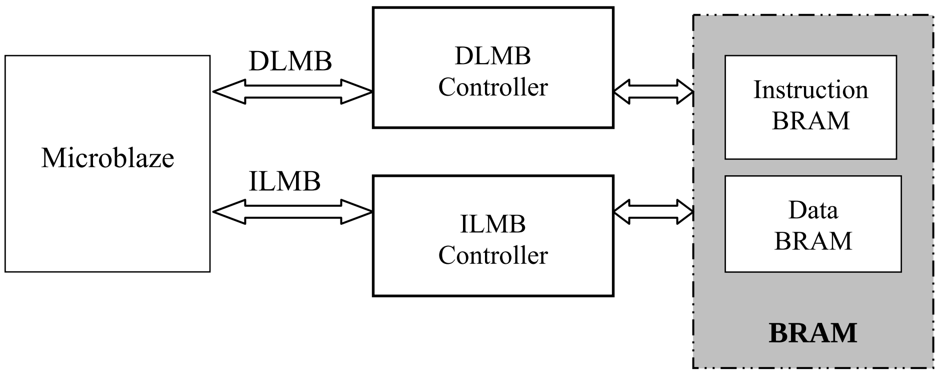


Figure 3.20 – Local Memory Bus (LMB)

3.4.4 On-Chip Peripheral Bus (OPB)

The Microblaze enables the attachment of several peripherals using the OPB interfaces. These peripherals must be connected to the processor using data and address buses.

The OPB implemented in this Microblaze system is a 32-bit configurable version of the IBM's Coreconnect architecture which facilitates the connection of peripherals to the processor. These peripherals must fulfill the compatibility with the OPB protocol.

This bus provides address and data interfaces both of 32-bit. It allows choosing the valid byte on the data bus, by dedicating a byte enabling signal. The OPB employs logic arbiter to arbitrate among masters, in case more than one master is connected to the bus.

Figure 3.21 shows the OPB slave interfaces, and tables 3.14 to 3.16 describe these interfaces.

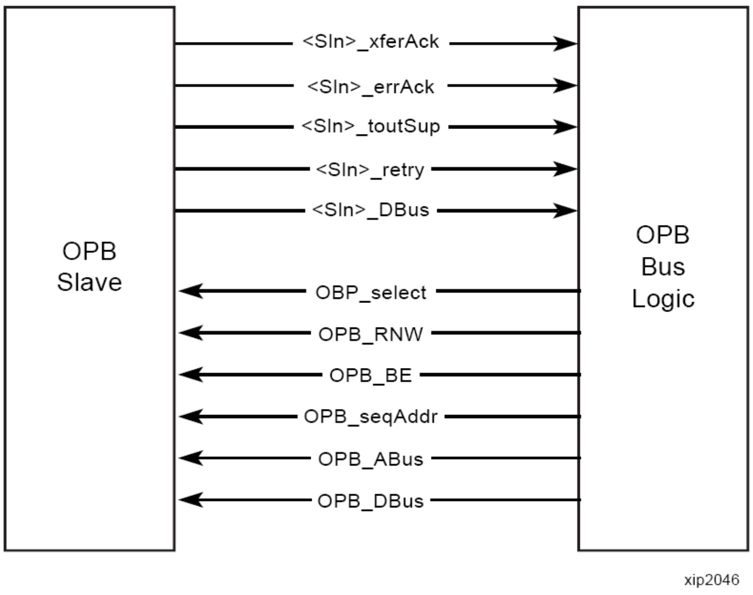


***Figure 3.21** - OPB Slave Attachment [XTU02]*

***Table 3.11** - OPB global signals*

Signal	I/O	Description
OPB_Clk	I	All input signals are synchronized to the rising edge of this clock.
OPB_Rst	I	Active high reset, which is asynchronous to the OPB_Clk. The Microblaze uses the same reset signal.

***Table 3.12** - OPB Interface Signals*

Signal	I/O	Description
OPB_ABus [0:31]	I	Address bus driven by the OPB and received by all slaves. This signal is valid whenever the OPB_Select signal is activated.
OPB_BE	I	Byte-enable indicates which byte is valid within the data path.
OPB_DBus [0:31]	I	Write data bus driven by the OPB and received by all slaves
OPB_RNW	I	(Read not Write) signal, setting this signal to "1" indicates that the master is performing a read operation on the slave, while a "0" values refers to a write operation on the slave.
OPB_select	I	Driven by the OPB to indicate that a transfer on the OPB is taking place.
OPB_seqAddr	I	OPB sequential address indicates that the current transfer will be followed by a transfer to the next sequential address in the same direction.
<Sln>_DBus[0:31]	O	Read data bus driven by the targeted slave. <Sln> refers to the name of the peripheral.

Table 3.12 *- OPB Interface Signals (Cont.)*

Signal	I/O	Description
<Sln>_xferAck	O	OPB transfer Acknowledge. Asserted by the addressed slave to indicate that the data transfer between the OPB master and slave has been accomplished.
<Sln>_retry	O	OPB bus cycle retry. This signal is asserted by an OPB slave to indicate that it is unable to perform the requested transfer at this time.
<Sln>_toutSup	O	Slave time-out Suppress. If an OPB slave wants to delay the bus operation for an extended time, it asserts this signal..
<Sln>_errAck	O	OPB transfer error Acknowledge. The signal is asserted by a slave device to indicate that the slave encountered an error in performing the requested transfer.

3.5 Microblaze PCIe Peripheral

The Xilinx Embedded Development Kit (EDK) delivers many compiled and optimized IPs that implement different functionalities and peripherals. This design tool does not provide all the modules required for the designing of a PCIe peripheral, which imposes a challenging task when carrying out such a design.

The PCIe peripheral has to be attached as a slave to the Microblaze as shown in the figure 3.22. In case of having more than one master existing on the bus, an OPB arbiter is needed to control the communication over the bus.

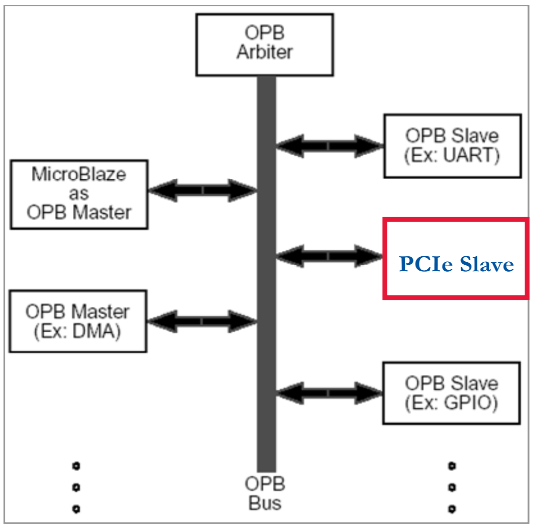


Figure 3.22 *- PCIe Peripheral Attachment as Slave [XTU02]*

A PCIe peripheral (or PCIe slave), shown in figure 3.23, is a device that consists of the protocol layers implemented by the PCIe core and the OPB to PCIe Bridge module. The OPB to PCIe Bridge module implements the standard OPB protocol and the logic needed to transmit/receive TLPs as well as the logic needed to access the configuration space of the PCIe core.

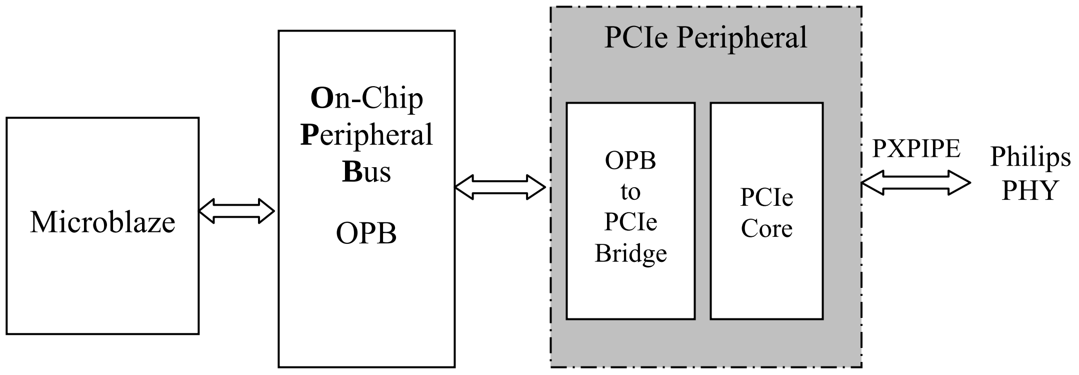


Figure 3.23 *- PCIe Peripheral*

3.5.1 OPB to PCIe Bridge

A module that bridges the OPB to the PCIe protocol layers is not available from Xilinx. Therefore, an effort was made to develop a simplified bridge that adapts the PCIe core to the OPB for the purpose of designing a PCIe peripheral.

Figure 3.24 shows the two modules, which construct this bridge: the OPB IPIF and USER LOGIC modules.

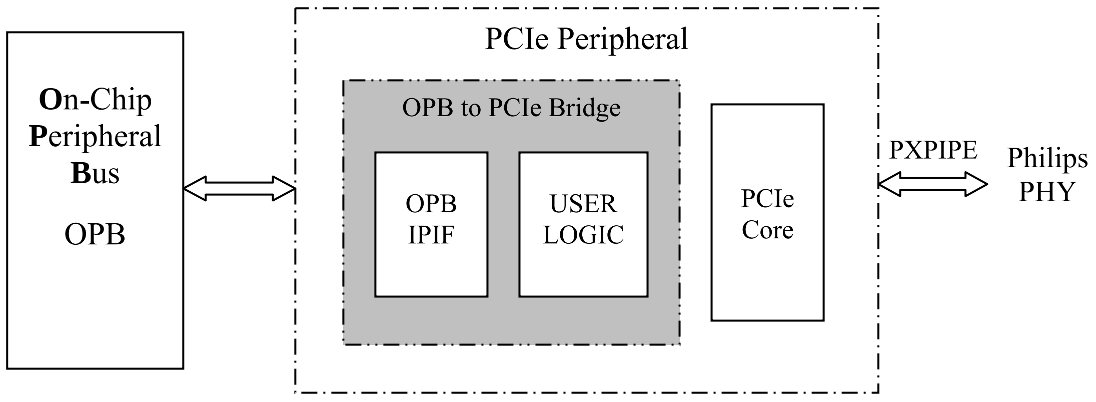


Figure 3.24 *- OPB to PCIe Bridge*

A top level block diagram of the OPB to PCIe Bridge is illustrated in figures 3.25 and 3.26. This bridge is controlled by the Microblaze over the OPB. The standard OPB protocol is implemented on the side of the OPB by the OPB IPIF module.

The interfaces to the PCIe are also shown in the figure. These interfaces are divided into four groups: Transmit Transaction Interface, Receive Transaction Interface, Common Transaction Interface, and Configuration Interface.

These interfaces are driven by the USER LOGIC module, which is explained later. In addition to the bus protocol compatibility, the bridge implements the logic needed for the transmission and reception of TLPs as well as the logic needed for accessing the configuration space of the PCIe core. The way how this is implemented, is explained in the coming sections.

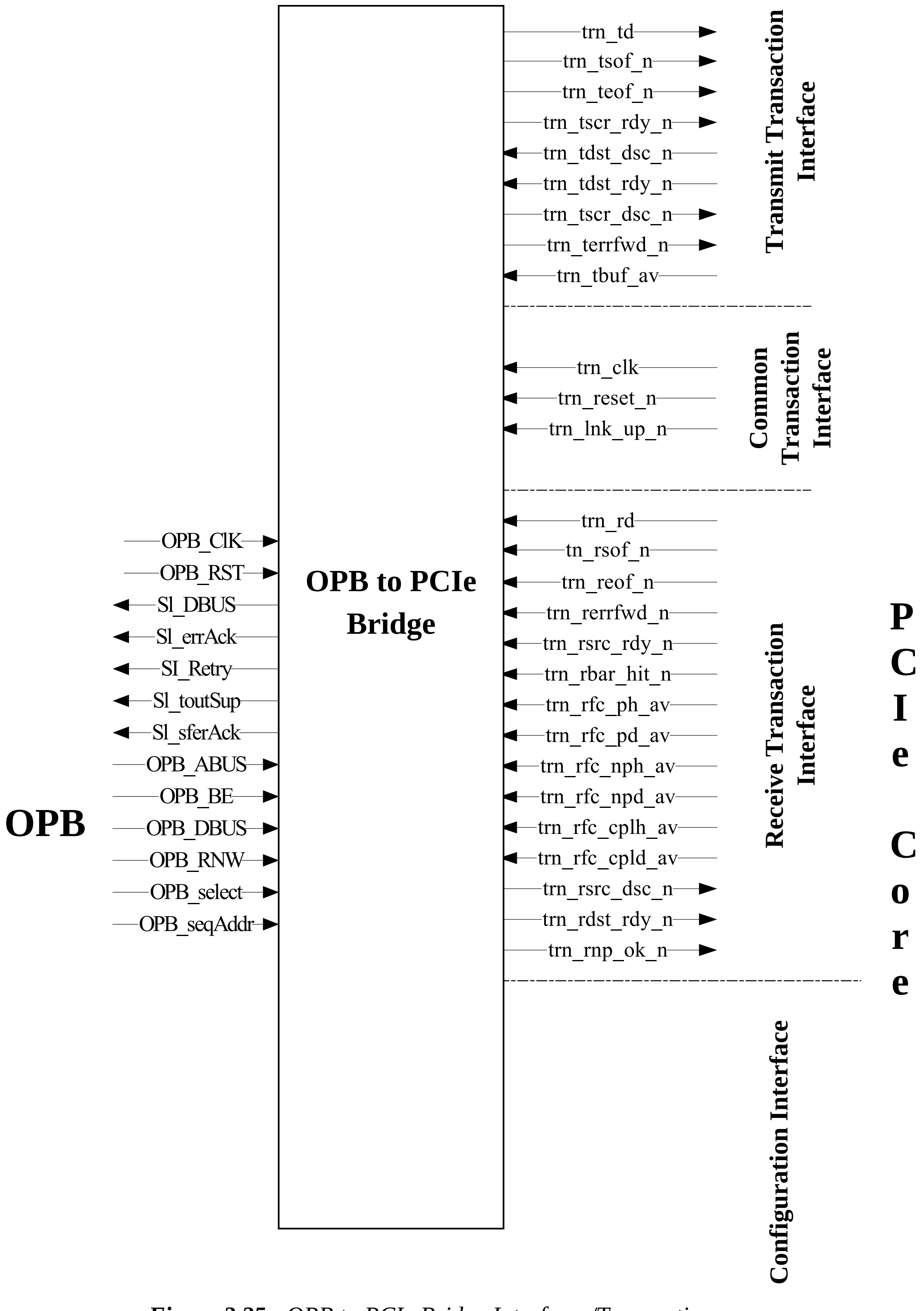


Figure 3.25 *- OPB to PCIe Bridge Interfaces/Transaction Interfaces*

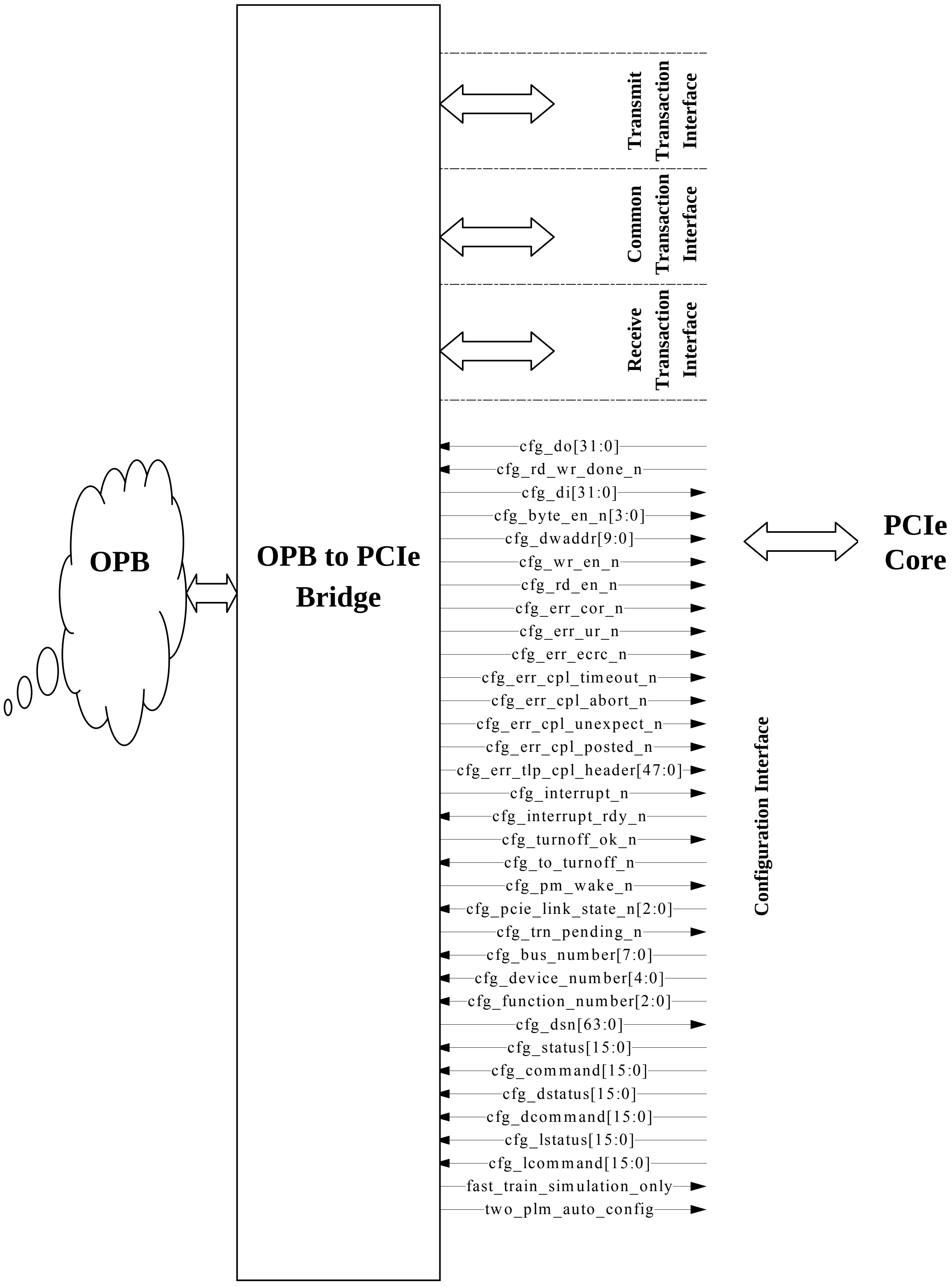


Figure 3.26 *- OPB to PCIe Bridge Interfaces/Configuration Interfaces*

In figure 3.27, a basic OPB read transaction is shown. When the OPB master, the Microblaze in this case, wants to access a register in the PCIe peripheral for the purpose of reading, it first selects the OPB by asserting the OPB_select signal and validates the 32-bit address on the OPB_ABus. While reading, the RNW is asserted to indicate a read not write access. The Microblaze puts a valid Byte Enable (BE) on the bus. Once the targeted slave recognizes this transaction, it loads the bus with the required data using the <Sln>_DBus signal. It asserts the acknowledge signal at the same time to complete the transfer. This in-turn causes the Microblaze to get the valid data and deselect the OPB by deasserting the OPB_select signal.

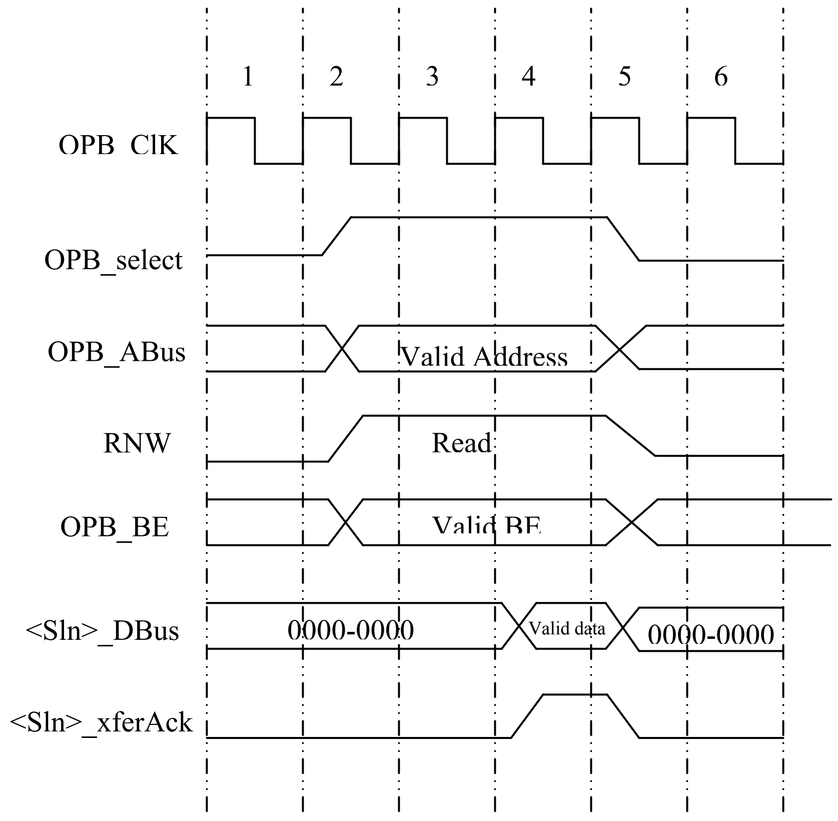


Figure 3.27 *- Basic OPB Read Transaction [XTU02]*

An OPB write transaction is shown in figure 3.28. The Microblaze does the same thing as with the read transaction.

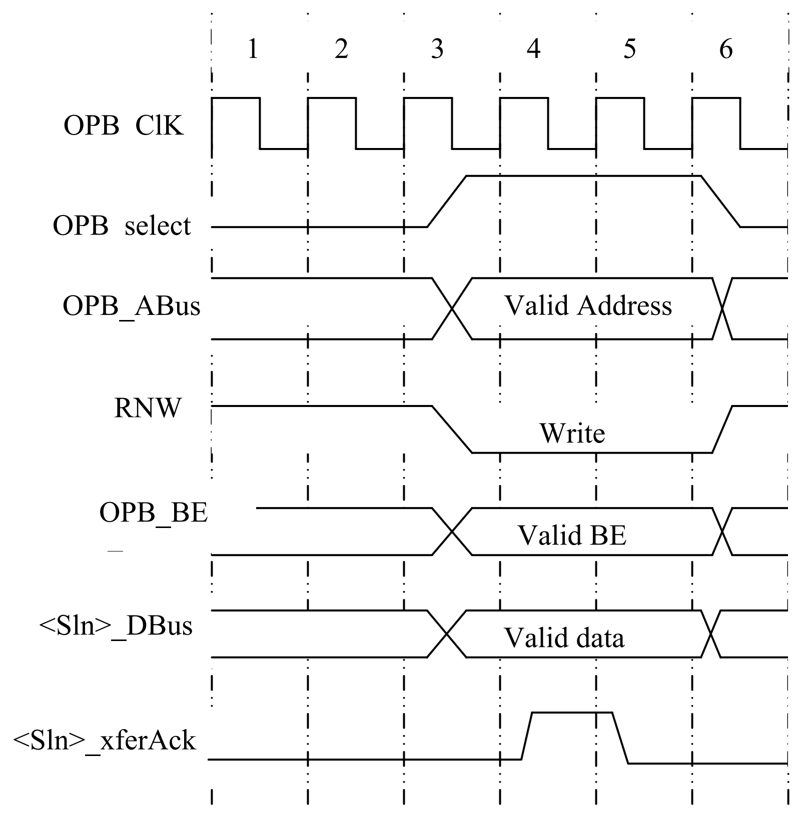


Figure 3.28 *- Basic OPB Write Transaction [XTU02]*

3.5.2 On-chip Peripheral Bus Intellectual Property Interface (OPB IPIF)

Although Xilinx did not provide a ready solution for the bridge, it facilitates its design by providing a module called OPB IPIF, which makes the connection of Xilinx cores or third party IPs to the OPB easier.

This module consists of eight different modules as depicted in figure 3.29. These modules allow an easy connection of the customized core to the processor bus, with making less effort in developing such modules from scratch. The interface to the IP core is called IP Interconnect (IPIC) as shown in the figure.

This OPB IPIF is a highly configurable module. It enables the designer to select the required modules for his optimal usage. A full set of the provided facilities: Master attachment, Slave attachment, Interrupt control, Address Decode, Read FIFOs, Write FIFOs, Direct Memory Access (DMA), and Scatter Gather (automated DMA) is shown in figure 3.29.

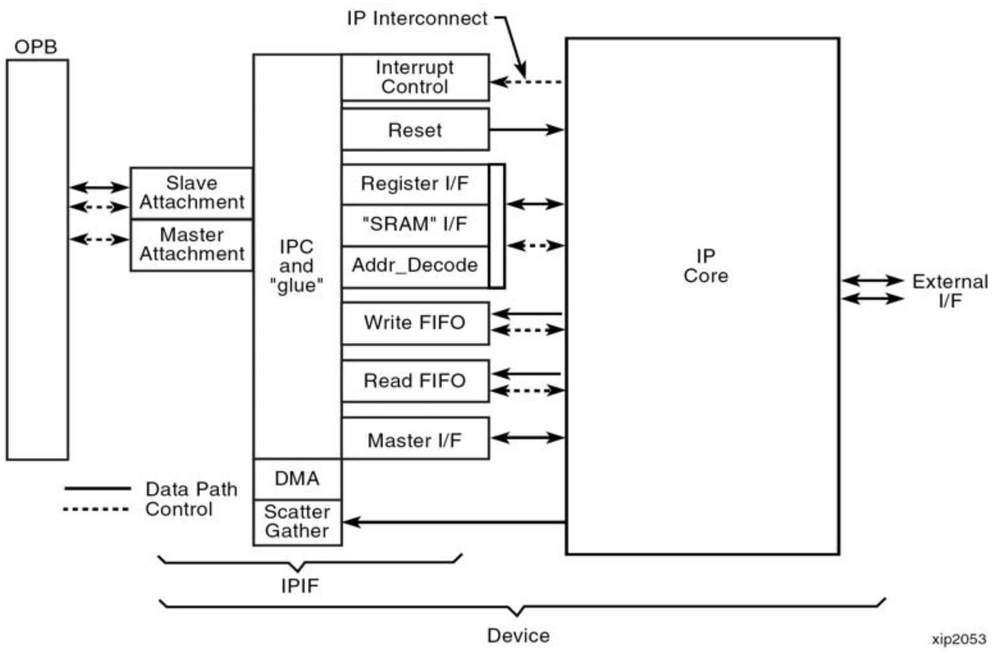


Figure 3.29 *- Full Set of OPB IPIF Features [XTU02]*

The main duties of the embedded processor are to check the configurability of the PCIe core, to access its configuration space by reading from/writing to this space, to control the transmission/reception of TLPs, and to send the Header and Payload of a TLP to the Transaction layer in the PCIe core. This indeed, makes the PCIe peripheral a simple slave that needs not more than input/output data buses, some register address decoding, read/write request and some acknowledge signals. Therefore, a simplified OPB IPIF, with only features that enabled register accessing was used as shown in figure 3.30.

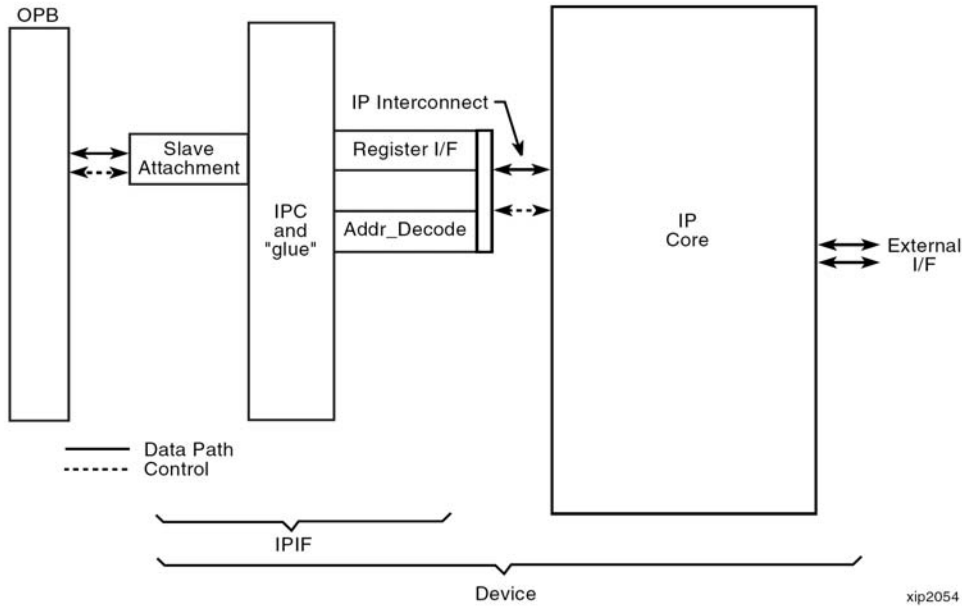


Figure 3.30 *- OPB IPIF Features for Register Access [XTU02]*

When creating the PCIe peripheral, several parameters can be configured in this OPB IPIF. The Base System Builder (BSB) of the Xilinx Platform Studio (XPS) assigns the peripheral base and high addresses. These addresses allow the processor to access the accessible registers implemented in the bridge. The Address and Data widths on the OPB are set to 32 bits. The targeted FPGA family and the number of registers can also be specified, based on the design requirements.

Figure 3.31 shows the OPB IPIF top-level block diagram, which only implements register interfaces. A slave attachment is shown in this figure. Such configuration allows the translation of the OPB standard protocol to some enabling signals serve the accessing of the registers implemented in the bridge. The address decoding unit is responsible for the generation of enable signals to access the targeted registers. These registers are enabled either to read from or to write onto them.

The interfaces to the PCIe core for the purpose of register access are explained in the next section.

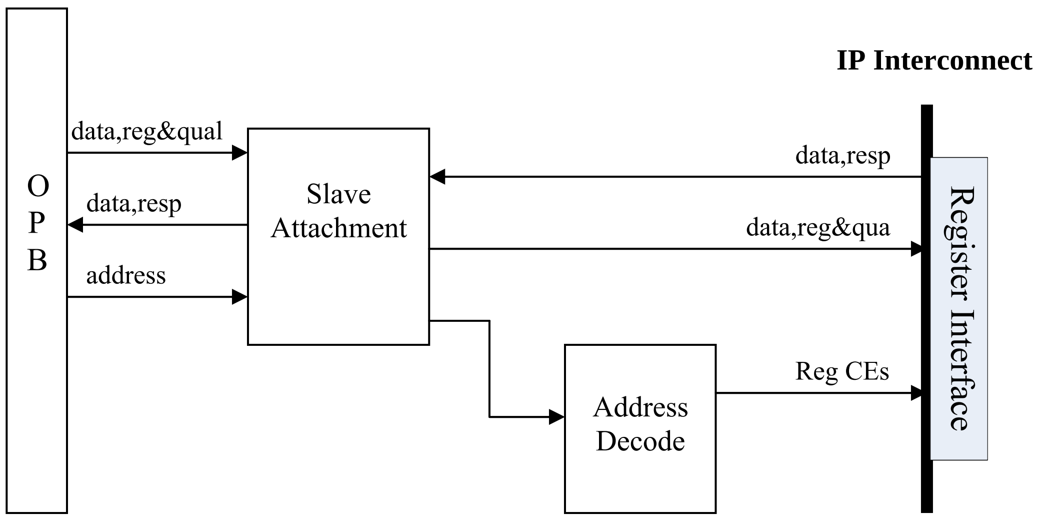


***Figure 3.31** - OPB IPIF Top-level Block Diagram, Register Interface Only [XDS414]*

3.5.3 USER LOGIC

The main functionality of the PCIe peripheral is implemented in this module. First of all, a top level of this USER LOGIC module is illustrated in figure 3.32. This module has two groups of interfaces. It interfaces the OPB IPIF module from one side and the PCIe core from the other side.

The interfaces to the OPB IPIF are not more than a way that enables the processor to access the registers implemented in the PCIe peripheral. In another word, it makes this peripheral compatible with the OPB protocol. In our simplified case, the register interface facility is implemented in the USER LOGIC module.

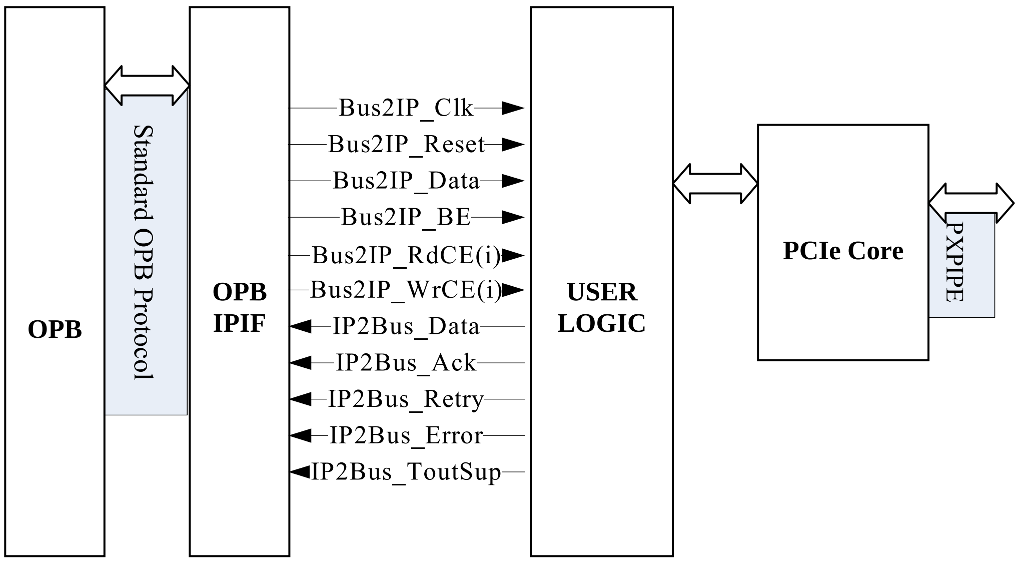


Figure 3.32 - *USER LOGIC Interfaces*

The following is a description of the interfaces to the OPB IPIF (IPIC):

- *Bus2IP_Clk:* This signal is connected to the OPB_Clk signal of 50 MHz, to which the OPB is synchronized.
- *Bus2IP_Reset:* This signal is used to reset the IP, asserted whenever the OPB_Rst signal is activated.
- *Bus2IP_Data:* 32-bit data transferred from the processor to the IP over the OPB.
- *Bus2IP_BE (0 to 3):* Byte Enables to indicate on which byte's location the valid data is available.
- *Bus2IP_RdCE (i):* Register read enables; where *i* indicates the corresponding register under a read transaction.
- *Bus2IP_WrCE (i):* Register write enables; where *i* indicates the corresponding register under a write transaction.
- *IP2Bus_Data:* 32-bit data from the IP to the OPB.
- *IP2Bus_Ack:* IP to bus read or write Acknowledgment. Asserted when the targeted register responses to a read or write transaction.
- *IP2Bus_Retry:* This signal is asserted whenever the PCIe peripheral is unable to perform the requested transfer at this time.
- *IP2Bus_Error:* This signal indicates an error response.
- *IP2Bus_ToutSup:* This signal is asserted by the peripheral whenever its acknowledgment or retry response will take longer than 8 cycles.

Figures 3.33 and 3.34 show examples of basic read and write transactions for IP Interconnect. Reading and writing accesses of a targeted register are illustrated.

Normally, (for each implemented register) a separate decoding for the read and write access exists. This is indicated by the vectors *Bus2IP_RegRDCE (i)* and *Bus2IP_RegWrCE (i)*, respectively as shown in the figures. The figures also show the signals *Bus2IP_RegRd* and *Bus2IP_RegWr*. These signals enable register read and write transactions, respectively. In the designed OPB IPIF module, only two signals serve the same purpose.

As mentioned above, the signal *Bus2IP_RdCE(i)* is used to enable a register read transaction, with the index *i* points to the addressed register and the signal *Bus2IP_WrCE(i)* is used to enable a register write transaction, with the index *i* indicates the requested register [XDS414].

As indicated in the figure, when reading a register, the PCIe peripheral drives the signal *IP2Bus_Data* with a 32-bit non-zero value. The peripheral drives zero otherwise. The peripheral can determine the duration of the transaction because it issues the acknowledgement.

For the purpose of this simplified unit, the acknowledgment signals due to read and write accesses are indicated by one signal, the *IP2Bus_Ack* on the interface. This acknowledgement can be returned in the same cycle as the request, making the transaction as short as one cycle. If the peripheral is unable to return an acknowledgment within 8 cycles, it can then drop the timeout by asserting the *IP2Bus_ToutSup* and holding it until it responds to the transaction [XDS414].

In case of an error, the peripheral can issue an error response by asserting *IP2Bus_Error* as indicated in the figures. If the transaction can be completed successfully, if it is retried, the peripheral asserts the *IP2Bus_Retry*.

For both transactions, the peripheral must drive these signals with zero, in case it is not addressed and accessed by the processor.

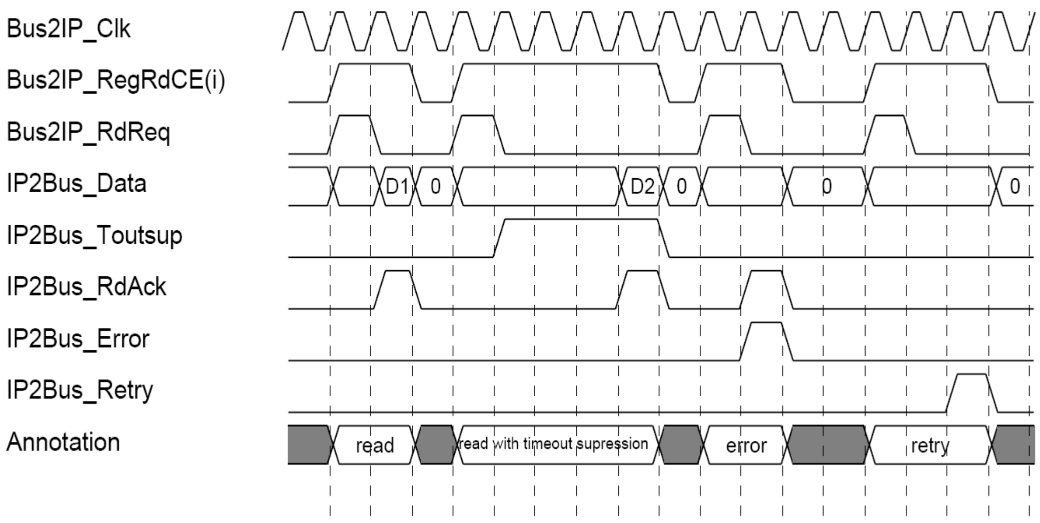


Figure 3.33 *- Read Transaction from IP that utilizes Register Decodes [XDS414]*

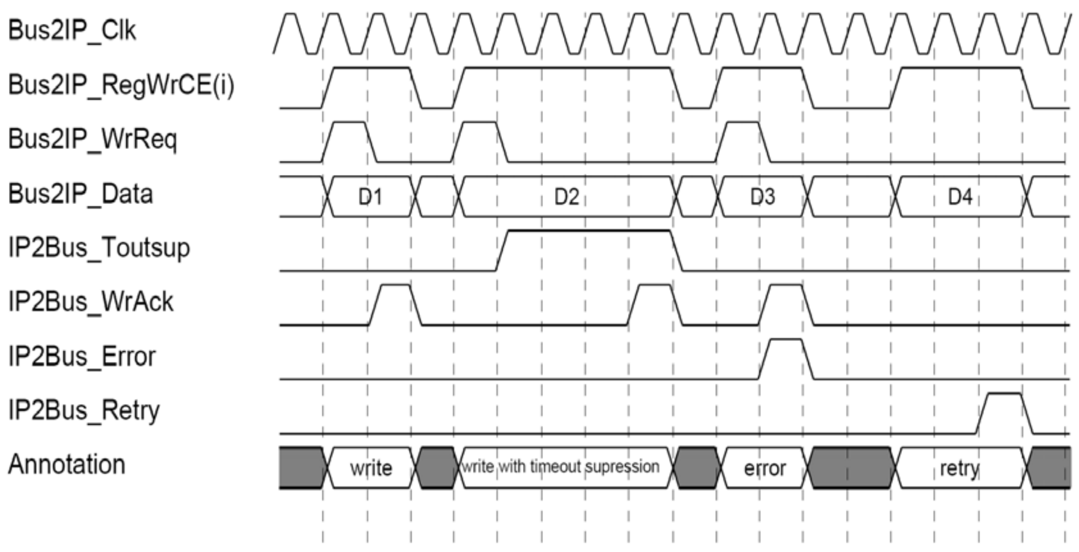


Figure 3.34 *- Write Transaction to IP that utilizes Register Decodes [XDS414]*

The USER LOGIC module consists of several units that implement the functionality of the PCIe peripheral. Figure 3.35 shows the register read, register write, 15 X 32 software accessible register bank, PCIe transmission state machine, PCIe receiving state machine, and PCIe configuration space access read/write units that construct the USER LOGIC. Following is a detailed description of each of these units.

3.5.3.1 Register Read

This unit implements the slave model register read multiplexer. Figure 3.33 shows an example of a read transaction from an addressed register.

3.5.3.2 Register Write

The slave model register write multiplexer is implemented in this unit. An example of a write transaction into an addressed register is illustrated in figure 3.34.

Both Register Read and Register Write units interface the OPB IPIF through the IP Interconnects (IPICs) as shown in figure 3.35.

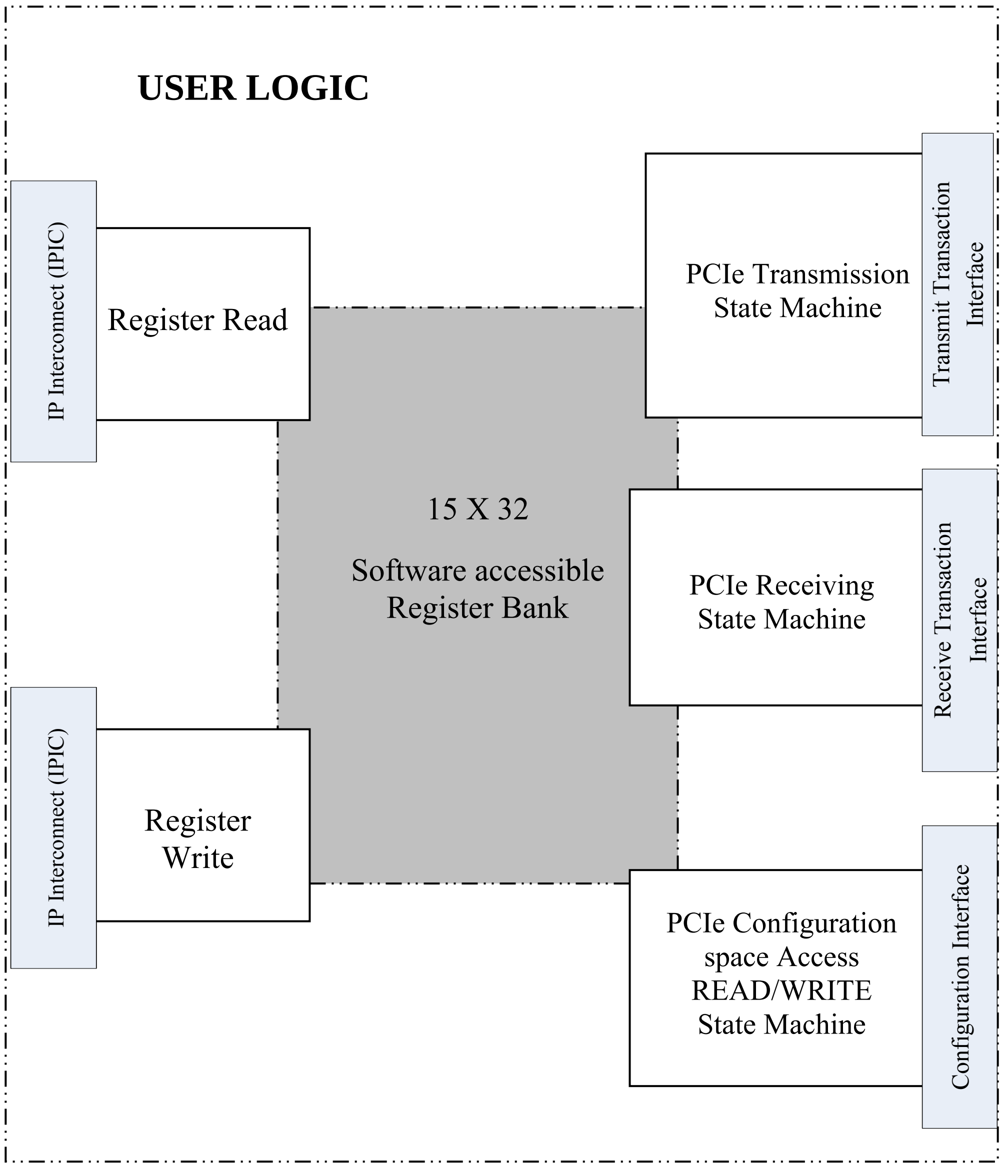

USER LOGIC
IP Interconnect (IPIC)
Register Read
15 X 32
Software accessible Register Bank
IP Interconnect (IPIC)
Register Write
PCIe Transmission State Machine
Transmit Transaction Interface
PCIe Receiving State Machine
Receive Transaction Interface
PCIe Configuration space Access READ/WRITE State Machine
Configuration Interface

Figure 3.35 *- USER LOGIC Internal Structure*

3.5.3.3 Software accessible Register Bank

In order to enable the Microblaze to control the transmission and reception of TLPs as well as to access the configuration space of the PCIe core, fifteen 32-bit registers were used. The Microblaze does access these register by issuing read or write transactions.

These registers can be accessed using the base address assigned to the PCIe peripheral, when creating it. Each register is assigned a unique address, which is the Base address plus an offset as shown in figure 3.36. The figure also shows the names given to these registers.

These registers use Big-Endian bit-reversed format to represent data as depicted in figure 3.37.

	0 31
Base Address + 0x38	PCIe CONFIG. DATA WRITE
Base Address + 0x34	PCIe CONFIG. DATA READ
Base Address + 0x30	REC. MWR/MRD DW4
Base Address + 0x2C	REC. MWR/MRD DW3
Base Address + 0x28	REC. MWR/MRD DW2
Base Address + 0x24	REC. CPLD DW4
Base Address + 0x20	REC. CPLD DW3
Base Address + 0x1C	REC. CPLD DW2
Base Address + 0x18	REC. MWR/MRD/CPLD DW1
Base Address + 0x14	MWR PAYLOAD
Base Address + 0x0F	MWR/MRD HDR DW3
Base Address + 0x0C	MWR/MRD HDR DW2
Base Address + 0x08	MWR/MRD HDR DW1
Base Address + 0x04	CONTROL
Base Address + 0x00	STATUS

Figure 3.36 *- Register Bank, Base Address and Offset are in Hexadecimal*

Byte address	n	n+1	n+2	n+3
Byte label	0	1	2	3
Byte significance	MSByte			LSByte
Bit label	0			31
Bit significance	MSBit			LSBit

Figure 3.37 *- Registers Big-Endian Format [XUG081]*

STATUS Register

The status register, shown in figure 3.38, provides a kind of feedback to the Microblaze by indicating the accomplishment of several tasks. The following is a detailed description of each field in this register:

0 24	25	26	27	28	29	30	31
00… 00	cpld_ transmitted	mem_rd_ received	mem_wr_ received	cpld_ received	mem_rd_ transmitted	mem_wr_ transmitted	cfg_ Command(2)

Figure 3.38 *- STATUS Register*

STATUS [0:24]: These bits are set to zeros.

STATUS [25]: *cpld_transmitted*: Indicates that a completion with data TLP (CPLD) was successfully transmitted.

STATUS [26]: *mem_rd_received:* Indicates that a Memory Read TLP was successfully received.

STATUS [27]: *mem_wr_received:* Indicates a successful reception of a Memory Write TLP.

STATUS [28]: *cpld_received:* Indicates that a CPLD TLP was successfully received.

STATUS [29]: *mem_rd_transmitted:* Indicates that a Memory Read TLP was successfully transmitted

STATUS [30]: *mem_wr_transmitted:* Indicates that a Memory Write TLP was successfully transmitted.

STATUS [31]: *cfg_command (2):* Refers to the master enabling in the command register of the PCIe configuration space. Setting this bit to "1" indicates that the PCIe Endpoint is enabled as a bus master and can initiate TLPs across the PCIe link. Setting this bit to "0" disables the PCIe Endpoint bus mastering. In this case the Endpoint can only respond to TLPs but not initiate them.

CONTROL Register

This register stores the control signals received from the Microblaze for the purpose of controlling the generation of TLPs as well as accessing the configuration space of the PCIe core. Figure 3.39 shows the bits allocation within this register.

0	1	2	3	4	5	6 15	16 31
master_ enable	mem_wr_ gen	mem_rd_ gen	compl_ gen	cfg_ read	cfg_ write	cfg_ dwaddr	0000… 000

Figure 3.39 *- CONTROL Register*

CONTROL [0]: *master_enable*: The Microblaze asserts this bit to confirm the enabling of the PCIe Endpoint as bus master. Deasserting this bit disables the master enabling feature.

CONTROL [1]: *mem_wr_gen:* The Microblaze asserts this bit after sending the information required to generate a Memory Write TLP (Header + Payload) to the USER LOGIC. This enables the generation of a Memory Write TLP and allows the sending of this information to the transaction layer located in the PCIe core. Deasserting this bit deactivates the generation of a Memory Write TLP.

CONTROL [2]: *mem_rd_gen:* The Microblaze asserts this bit after sending the information required to generate a Memory Read TLP (Header + Payload) to the USER LOGIC. This enables the generation of a Memory Read TLP and allows the sending of this information to the transaction layer located in the PCIe core. Deasserting this bit deactivates the generation of a Memory Read TLP.

CONTROL [3]: *compl_gen*: The Microblaze asserts this bit after receiving a Memory Read TLP that request a completion with data.

CONTROL [4]: *cfg_read*: The Microblaze sets this bit to "1" in order to generate a configuration register read cycle. At the same time, it writes a 10-bit address onto CONTROL [6:15] to address the required configuration register in the PCIe core.

CONTROL [5]: *cfg_write*: The Microblaze sets this bit to "1" in order to generate a configuration register write cycle. At the same time, it writes a 10-bit address onto CONTROL [6:15] to address the required configuration register in the PCIe core.

CONTROL [6:15]: *cfg_dwaddr*: A 10-bit address for a DWORD location in the configuration space of the PCIe core. This address points to two 16-bit registers.

CONTROL [16:31]: These bits are set to zeros.

MWR/MRD HDR DW1 Register

This register contains the first header's DW of a Memory Write or a Memory Read TLP. This DW is written by the Microblaze over the OPB to the USER LOGIC. The transferring of the TLP to the PCIe core starts after the reception of all information required from the Microblaze and the activation of the memory generation signal in the control register.

Figure 3.40 shows the MWR/MRD HDR DW1 register and the following is a detailed illustration of each bit:

0	1 2	3 7	8	9 11	12 15	16	17	18 19	20 21	22 31
0	FMT	Type	0	TC	0000	TD	EP	Attr	00	length

***Figure 3.40** - MWR/MRD HDR DW1 Register*

MWR/MRD HDR DW1 [0]: *Reserved bit*: This bit must be set to zero.

MWR/MRD HDR DW1 [1:2]: *FMT* (Packet Format) and MWR/MEMRD HDR DW1 [3:7]: *Type* (TLP packet Type field) are used in a combination that specifies the transaction type, header size, and whether data payload is present or not (MWR/MRD HDR DW1 [1:7]):

0000000b = Memory Read (3DW without data)
0100000b = Memory Read (4DW without data)
1000000b = Memory Write (3DW with data)
1100000b = Memory Write (4DW with data)
0001010b = Completion (3DW without data)
1001010b = Completion (3DW with data)

MWR/MRD HDR DW1 [8]: *Reserved bit*: This bit must be set to zero.

MWR/MRD HDR DW1 [9:11]: *TC* (Traffic Class): These 3 bits are used to determine the traffic class applied to the TLP. There are seven different traffic classes. In our design, the default traffic class was applied to the transmitted TLP:

000 = Traffic Class 0 (Default Class)
001 = Traffic Class 1
010 = Traffic Class 2
011 = Traffic Class 3
100 = Traffic Class 4
101 = Traffic Class 5
110 = Traffic Class 6
111 = Traffic Class 7

MWR/MRD HDR DW1 [12:15]: *Reserved bits*: These bits must be set to zeros.

MWR/MRD HDR DW1 [16]: *TD* (TLP Digest Field Present): If set = 1, the optional 32-bit Cyclic Redundancy Check (CRC) field is included with this TLP. The receiver must check the presence of this field when this TD is set to “1”. This bit is set = 0 by the Microblaze in order to ignore checking this CRC.

MWR/MRD HDR DW1 [17]: *EP* (Poisoned data): When set = 1, the payload data with this TLP should be considered corrupted, although the transaction completes normally. This bit is set = 0 to indicate a valid payload data.

MWR/MRD HDR DW1 [18:19]: *Attr* (Attribute): Bit 18 = Relaxed ordering: If set = 1, The PCI-X relaxed ordering is enabled for this TLP. Strict PCI ordering is used otherwise. Bit 19 = No Snoop. These 2 bits are set to zeros.

MWR/MRD HDR DW1 [20:21]: *Reserved bits*: These bits must be set to zeros.

MWR/MRD HDR DW1 [22:31]: *length*: TLP data payload transfer size (in DW). Maximum transfer size is 10 bits; 2^{10} = 1024 DW (4KB). Encoding:

00 0000 0001b = 1DW

00 0000 0010b = 2DW

.

.

11 1111 1111b = 1023 DW
00 0000 0000b = 1024 DW

In this designed Endpoint, the maximum payload size is 1 DW.

MWR/MRD HDR DW2 Register

In this register, shown in figure 3.41, the second DW of a transmitted Memory Write or Memory Read TLP is stored.

0 … 15	16 … 23	24 … 27	28 … 31
Endpoint ID	Tag	Last DW BE	1st DW BE

***Figure 3.41** - MWR/MRD HDR DW2 Register*

MWR/MRD HDR DW2 [0:15]: *Endpoint ID*: Indicates the identification number of the device that generates this TLP. This number is indicated for the purpose of returning a completion TLP.

MWR/MRD HDR DW2 [0:7]: Bus number,

MWR/MRD HDR DW2 [8:12]: Device number and

MWR/MRD HDR DW2 [12:15]: Function number.

MWR/MRD HDR DW2 [16:23]: *Tag*: These bits are used to identify each outstanding request issued by the requester. Upon the sending of one request, the next sequential tag is assigned. By default, only 5 bits are used for this tag, which allows 32 outstanding transactions at a time. This number can be extended to 256 tags by using 8 bits. This can be done by setting the extended tag bit in the PCIe control register = 1, when configuring the PCIe core.

MWR/MRD HDR DW2 [24:27]: *Last DW BE*: These bits are used to qualify the bytes in the last sent DW. These byte enables are active high. A value of "0" indicates that the concerned byte should not be written by the completer of the TLP. It is written otherwise. Since we have the valid transferred data are within only 1 aligned DW, the Last DW BE must be = 0000b.

MWR/MRD HDR DW2 [28:31]: *1st DW BE*: These bits are used to qualify the bytes in the first sent DW. Since we have the valid transferred data are within only 1 aligned DW, the 1st DW BE must be = 1111b.

MWR/MRD HDR DW3 Register

This register includes a 32-bit memory address to point to the system memory location, onto which the payload accompanying the TLP should be stored. For the purpose of this diploma work, only 32-bit addressing is allowed, although a 64-bit addressing is possible, by reconfiguring the PCIe core. This 64-bit addressing extends the header of the TLP to 4 DWs.

Figure 3.42 shows the MWR/MRD HDR DW3 Register. The bits 30 and 31 are reserved bits and must be set to zero. Doing so forces the address to be a DW aligned.

0 … 29	30 … 31
Addresse [0:29]	00

***Figure 3.42** - MWR/MRD HDR DW3 Register*

MWR PAYLOAD Register

This register holds the data payload to be transmitted across the PCIe link. This designed PCIe Endpoint supports only 1 DW payload.

REC. MWR/MRD/CPLD DW1 Register

In this register, shown in figure 3.43, the first header's DW of a received Memory Write/ Read or CPLD TLP is stored. This DW is sent to the Microblaze over the OPB. The bits allocation and description are the same as those in the MWR/MRD HDR DW1 Register.

0	1 2	3 7	8	9 11	12 15	16	17	18 19	20 21	22 31
0	FMT	Type	0	TC	0000	TD	EP	Attr	00	length

__Figure 3.43__ - REC. MWR/MRD/CPLD DW1 Register

REC. CPLD DW2 Register

The second received DW of a CPLD is stored in this register shown in figure 3.44. The following is a detailed description of each bit in this register:

0 15	16 18	19	20 31
Completer ID	compl_ status	compl_ bcm	Byte Count

__Figure 3.44__ - REC. CPLD DW2 Register

REC. CPLD DW2 [0:15]: *Completer ID*: Indicates the identification number of the completer. This information is not needed for routing the completion TLP.

REC. CPLD DW2 [0:7]: Completer bus number.

REC. CPLD DW2 [8:12]: Completer device number.

REC. CPLD DW2 [12:15]: Completer function number.

REC. CPLD DW2 [16:18]: *compl_status*: Indicates the status of the completion by the completer. Encoding:

000b = Successful Completion (SC)
001b = Unsupported Request (UR)
010b = Configuration Request Retry Status (CRS)
100b = Completer Abort. (CA)

REC. CPLD DW2 [19]: *compl_bcm* (byte modified count): This value is set = 1, only by PCI-X completers. This indicates that the byte count field reflects the first transfer payload rather than the total payload remaining.

REC. CPLD DW2 [20:31]: *Byte Count*: This is the number of bytes to be returned with a completion TLP. Normally, this value can be derived from the length of the TLP. For 1 DW, this value is set = 004x.

REC. CPLD DW3 Register

The third received DW of a CPLD is stored in this register shown in figure 3.45.

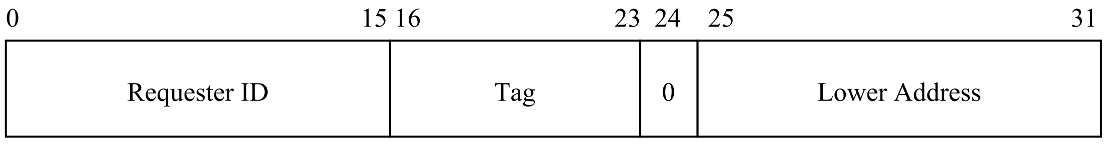


Figure 3.45 *- REC. CPLD DW3 Register*

REC. CPLD DW3 [0:15]: *Requester ID*: This identification number is copied from the request in order to be used in routing the completion back to the original requester.

REC. CPLD DW3 [16:23]: *Tag*: 8-bit tag received with the request. These bits are used by the requester to associate the incoming completion with an outgoing request.

REC. CPLD DW3 [25:31]: Lower Address: These 7 bits are the lower 7 bits of the address of the first valid byte of the data. This address is calculated from the request length and byte enables. In our case, this byte start address is the same as the starting address of the DW, since we only have 1 aligned DW.

REC. CPLD DW4 Register

The received completion data requested by the Endpoint as a consequence of a Memory Read TLP is stored in this register.

REC. MWR/MRD DW2 Register

In this register, the second received DW of a Memory Read or Memory Write TLP is stored. The contents of this register are shown in figure 3.46.

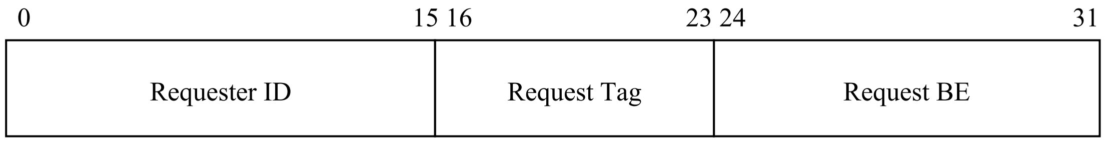


Figure 3.46 *- REC. MWR/MRD DW2 Register*

REC. MWR/MRD DW2 [0:15]: *Requester ID*: Indicates the identification number of the device that generates this TLP (used for returning a completion TLP). In our design, this number is the identification number of the Root Complex which generates a Memory Write/Read TLP to write/read a DW to/from a memory mapped location within the PCIe Endpoint.

REC. MWR/MRD DW2 [16:23]: *Request Tag*: These bits are used to identify each outstanding request issued by the requester.

REC. MWR/MRD DW2 [24:31]: *Request BE*: first and last DW Byte Enables, which are received with the request to qualify the bytes in the first and last DW sent. In case of having only 1 DW, these bits have to be set = 00001111b.

REC. MWR/MRD DW3 Register

The third received header's DW of a Memory Read/Write TLP is stored in this register. This 32-bit address points to the memory mapped location within the PCIe Endpoint, to which the data Payload is to be written, in case of a Memory Write TLP, or from which data is to be read, in case of a Memory Read TLP.

REC. MWR DW4 Register

In case of a received Memory Write TLP, this register is used to store the data payload associated with this TLP.

PCIe CONFIG. DATA READ Register

When the Microblaze generates a PCIe Configuration Read cycles, the required data received from the configuration space of the PCIe core is loaded in this register. This data is the content of two configuration registers within that space.

PCIe CONFIG. DATA WRITE Register

When the Microblaze generates a PCIe Configuration Write cycle, the required data to be written to the addressed configuration register of the PCIe core is located in this register.

3.5.3.4 PCIe Transmission State Machine

The PCIe Transmission State Machine is responsible for transferring the information required to generate a TLP to the transaction layer of the PCIe core. This information is written by the Microblaze over the OPB onto the internal registers of the USER LOGIC module, in case of having a Memory Write/read TLP, or assembled internally in the USER LOGIC, in case of a completion with data TLP.

In case of a Memory Write TLP, this information consists of the header and the data Payload. The header is only needed when generating a Memory Read TLP. For the generation of a CPLD TLP, the header and the completion data are required.

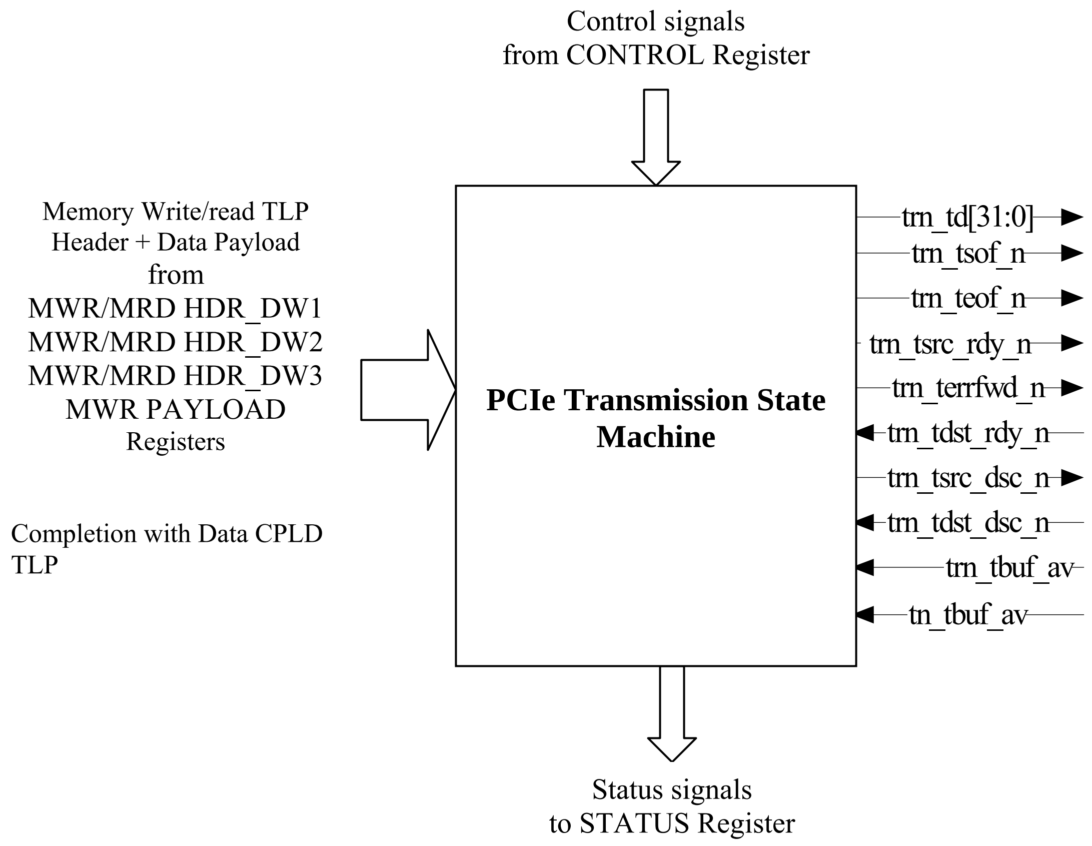


***Figure 3.47** - PCIe Transmission State Machine*

Figure 3.47 shows the interfaces of this state machine. The main purpose of this State Machine is to generate the timing diagram depicted in figure 3.49. This figure shows a TLP with a header of 3 DWs and a payload of 1 DW. This TLP can be an example of a 32-bit addressable Memory Write request, or a CPLD TLP.

Table 3.13 illustrates a simplified transition table of this state machine. The corresponding state diagram is shown in figure 3.48. As mentioned before, these states are required for sending Memory Write, Memory Read, and CPLD TLPs.

In this state machine and in case of having the example of a Memory Write request as shown in figure 3.49, the following sequence of events has to be performed on the PCIe Transmit Transaction interfaces:

Table 3.13 *- PCIe Transmission State Machine Transition Table**

Current State	**Inputs**						**Input Vector**	**Outputs**							**Output Vector**	**Next State**
	trn_tdst_rdy_n	trn_tdst_dsc_n	master_enable	mem_rd_gen	mem_wr_gen	compl_gen		cpld_transmitted	mem_wr_transmitted	mem_rd_transmitted	trn_tsof_n	trn_teof_n	trn_tsrc_rdy_n	trn_tsrc_dsc_n		
S_0	X	0	X	X	X	X	A	0	0	0	1	1	1	0	a	S_0
S_0	0	1	1	0	1	0	B	0	0	0	0	1	0	1	b	S_1
S_0	0	1	1	1	0	0	C	0	0	0	0	1	0	1	c	S_4
S_0	0	1	1	0	0	1	D	0	0	0	0	1	0	1	d	S_6
S_6	X	0	X	X	X	X	E	0	0	0	1	1	1	0	e	S_0
S_6	0	1	X	X	X	X	F	0	0	0	1	1	0	1	f	S_7
S_6	1	1	X	X	X	X	G	0	0	0	1	1	0	1	g	S_6
S_7	X	0	X	X	X	X	H	0	0	0	1	1	1	0	h	S_0
S_7	0	1	X	X	X	X	I	0	0	0	1	1	0	1	i	S_8
S_7	1	1	X	X	X	X	J	0	0	0	1	1	0	1	j	S_7
S_8	X	0	X	X	X	X	K	0	0	0	1	1	1	0	k	S_0
S_8	0	1	X	X	X	X	L	1	0	0	1	0	0	1	l	S_0
S_8	1	1	X	X	X	X	M	0	0	0	1	1	0	1	m	S_8
S_1	X	0	X	X	X	X	N	0	0	0	1	1	1	0	n	S_0
S_1	0	1	X	X	X	X	O	0	0	0	1	1	0	1	o	S_2
S_1	1	1	X	X	X	X	P	0	0	0	1	1	0	1	p	S_1
S_2	X	0	X	X	X	X	Q	0	0	0	1	1	1	0	q	S_0
S_2	0	1	X	X	X	X	R	0	0	0	1	1	0	1	r	S_3
S_2	1	1	X	X	X	X	S	0	0	0	1	1	0	1	s	S_2
S_3	X	0	X	X	X	X	T	0	0	0	1	1	1	0	t	S_0
S_3	0	1	X	X	X	X	U	0	1	0	1	0	0	1	u	S_0
S_3	1	1	X	X	X	X	V	0	0	0	1	1	0	1	v	S_3
S_4	X	0	X	X	X	X	W	0	0	0	1	1	1	0	w	S_0
S_4	0	1	X	X	X	X	X	0	0	0	1	1	0	1	x	S_5
S_4	1	1	X	X	X	X	Y	0	0	0	1	1	0	1	y	S_4
S_5	X	0	X	X	X	X	Z	0	0	0	1	1	1	0	z	S_0
S_5	0	1	X	X	X	X	γ	0	0	1	1	0	0	1	η	S_0
S_5	1	1	X	X	X	X	σ	0	0	0	1	1	0	1	μ	S_5

* For the purpose of simplification, not all the inputs and outputs are specified in this table. For example, the assignment of trn_td [31:0] is not included here.

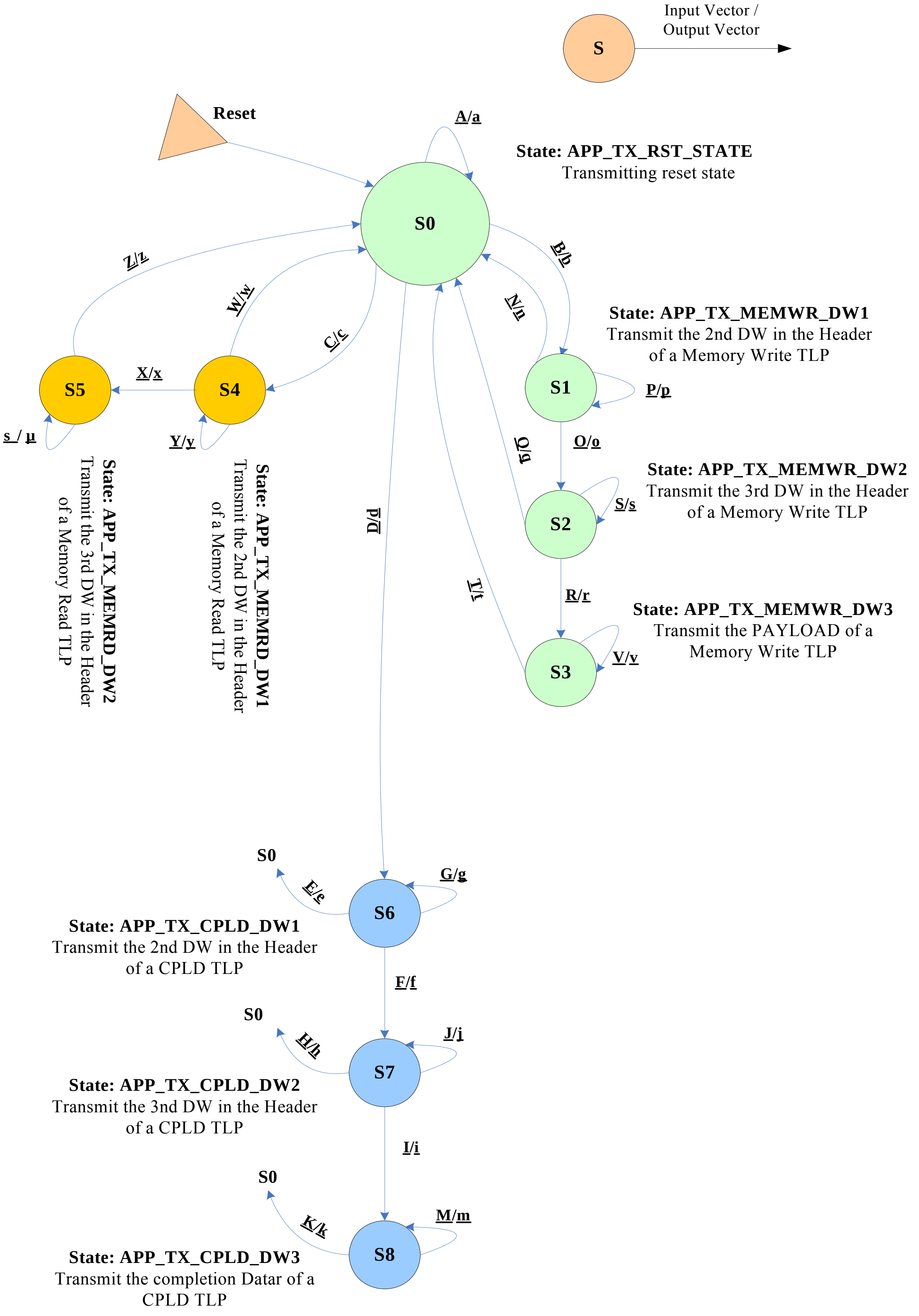


Figure 3.48 - PCIe Transmission State Machine State Diagram

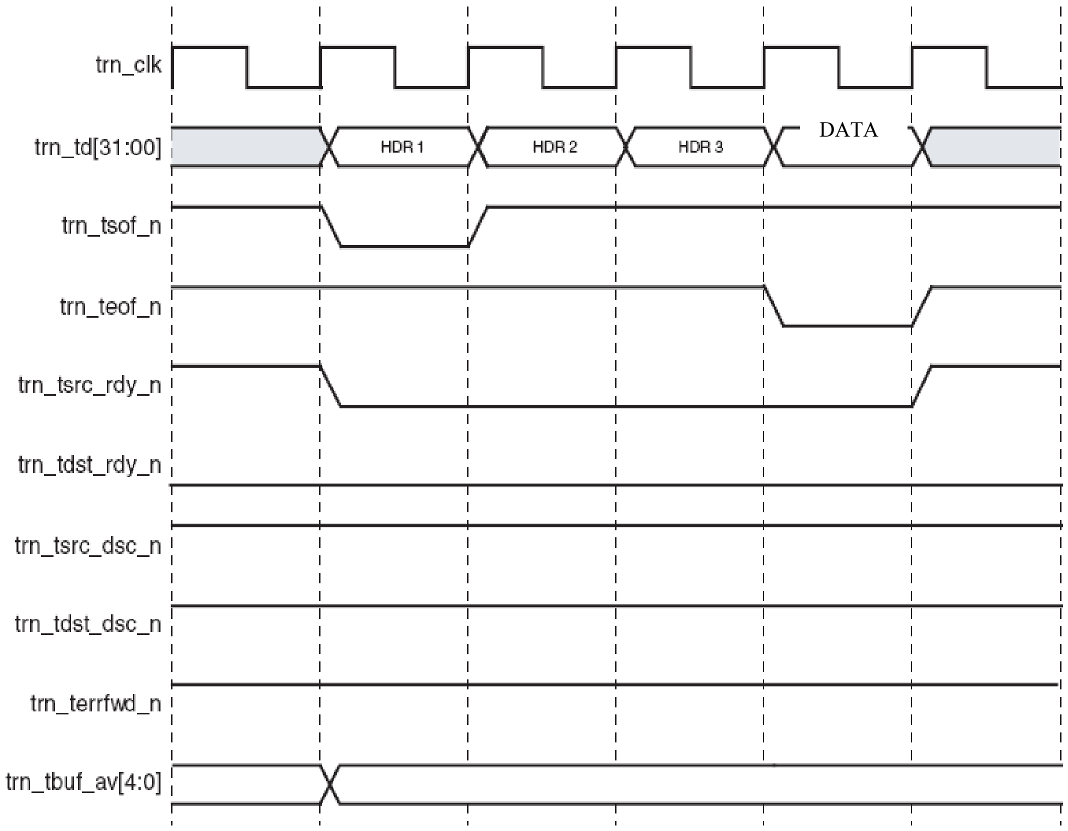


***Figure 3.49** - Memory Write TLP with a 3 DW Header and Payload [XUG167]*

Firstly, after receiving a control signal (mem_wr_gen) from the processor indicating the availability of all DWs of the packet, the machine asserts trn_tsof_n, trn_tsrc_rdy_n and presents the first TLP's DW on trn_td [31:0], as long as the PCIe core is indicating that it is ready to accept data on trn_rd [31:0] by asserting trn_tdst_rdy_n.

Secondly, at the next clock cycle, the state machine deasserts trn_tsof_n and presents the rest of the TLP's DWs on trn_td [31:0]. The PCIe core keeps the assertion of trn_tdst_rdy_n.

Thirdly, this state machine asserts trn_tsrc_rdy_n and trn_teof_n together with the last DW of data.

Finally, at the next clock cycle, the state machine deasserts trn_tsrc_rdy_n to indicate the end of valid transfer of data on trn_td [31:0].

In figure 3.50, a 3-DW TLP Header without data payload is shown. A 32-bit addressable Memory Read request is an example of such TLP.

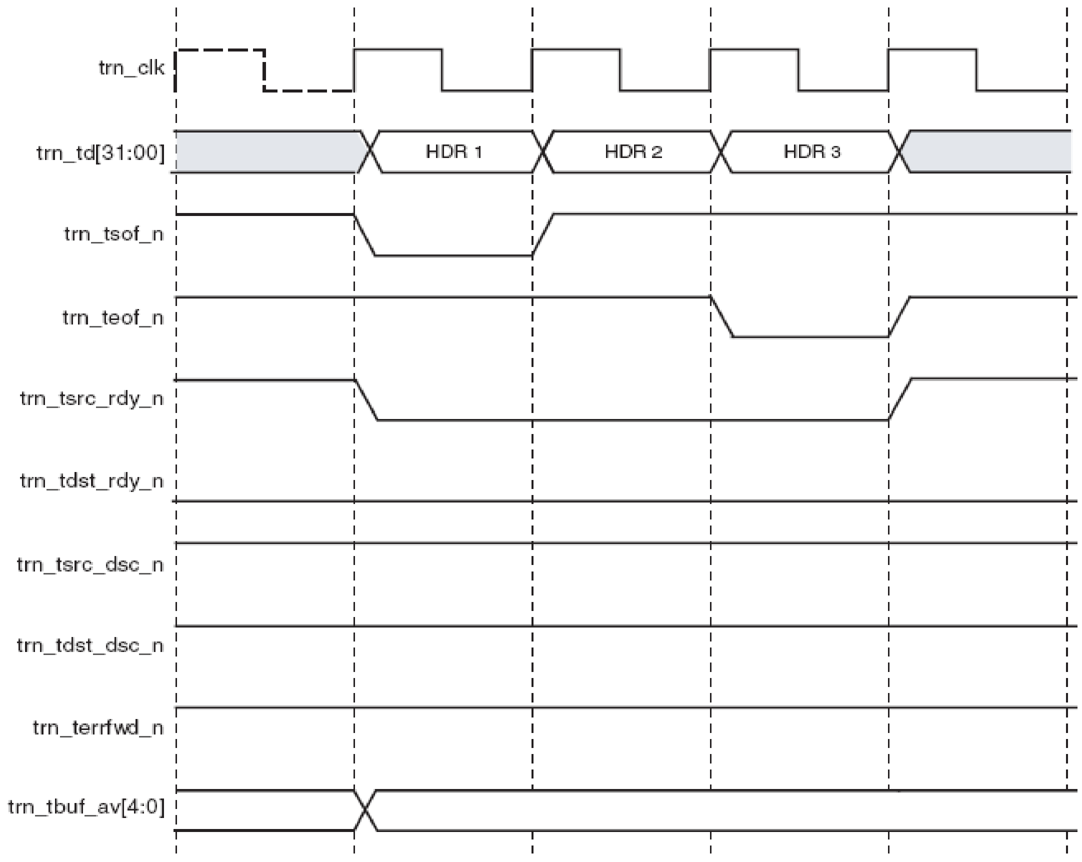


Figure 3.50 *- Memory Read TLP with a 3 DW Header without Payload [XUG167]*

3.5.3.5 PCIe Receiving State Machine

This PCIe Receiving State Machine is responsible for receiving TLPs from the PCIe core. The received TLPs are stored in the internal register of the USER LOGIC before being transferred to the Microblaze over the OPB.

Figure 3.51 shows the interfaces of this machine. The main purpose of this PCIe Transmission State Machine is to enable the reception of TLPs coming from the PCIe core by generating the timing diagram shown in figure 5.53. This figure shows a received TLP with a header of 3 DWs and a payload of 1 DW. This TLP might represent an example of a received 32-bit addressable Memory Write request, or a received CPLD TLP.

Table 3.14 illustrates a simplified transition table of this state machine. Figure 3.52 shows the corresponding state diagram. As mentioned before, these states are required for receiving Memory Write, Memory Read, and CPLD TLPs.

In this state machine and for the purpose of receiving a Memory Write TLP as shown in figure 3.53, the following sequence of events has to be performed on the PCIe Receive Transaction interfaces:

Firstly, this state machine asserts trn_rdst_rdy_n whenever it is ready to receive data.

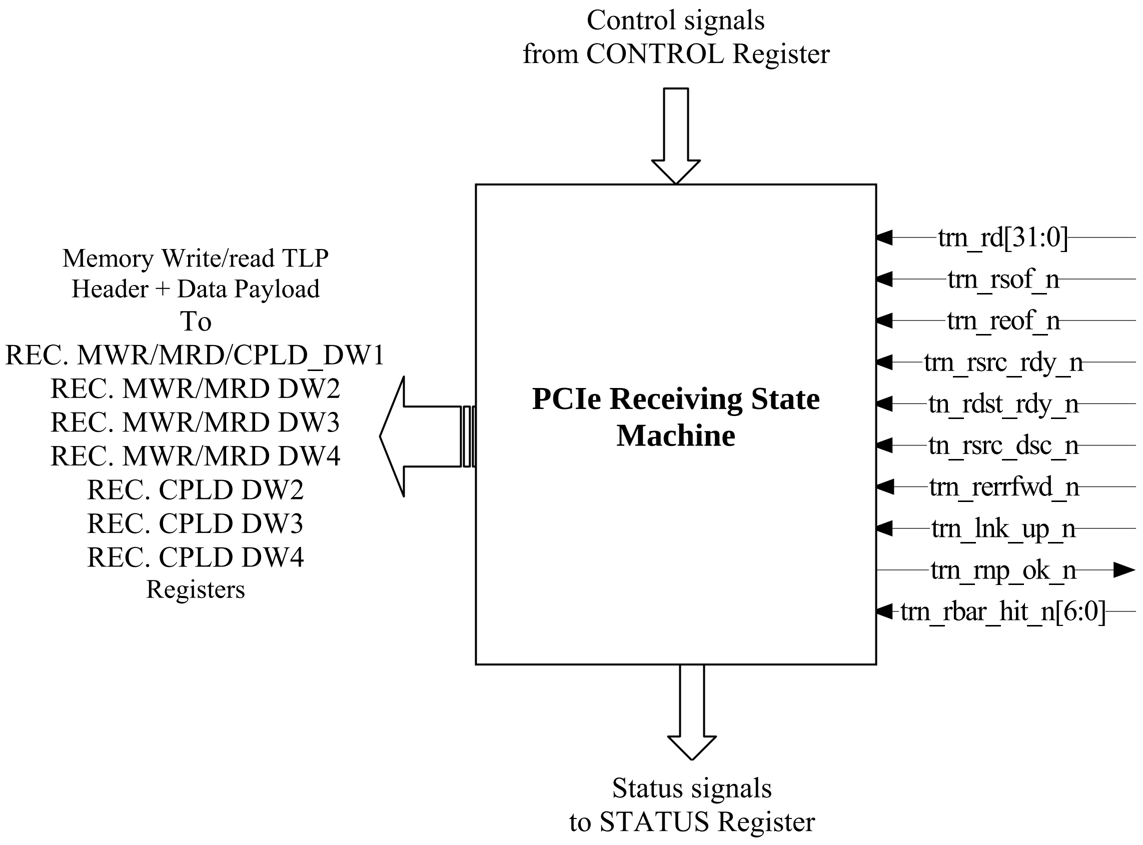


Figure 3.51 *- PCIe Receiving State Machine*

Secondly, the PCIe core asserts trn_rsrc_rdy_n when it is ready to transfer the data. At the same time, it asserts trn_rsof_n and presents the first DW of the TLP on trn_rd [31:0].

Thirdly, at the next clock cycle, the PCIe core deasserts trn_rsof_n, asserts trn_rsrc_rdy_n, and presents the rest of the TLP DWs on trn_rd [31:0] for the successive clock cycles. The state machine keeps the assertion of trn_rdst_rdy_n.

Fourthly, the PCIe core asserts trn_reof_n with the simultaneous presentation of the last DW of the TLP.

Fifthly, at the next clock cycle, the PCIe core deasserts trn_rsrc_rdy_n to indicate the end of valid transfer of data on trn_rd [31:0].

Table 3.14 - *PCIe Receiving State Machine Transition Table**

Current State	**Inputs**			**Input Vector**	**Outputs**			**Output Vector**	**Next State**
	trn_rsof_n	trn_rsrc_rdy_n	trn_rd [30:24]		cpld_received	mem_wr_received	mem_rd_received		
S_0	0	0	"1001010"	A	0	0	0	a	S_6
S_0	0	0	"1000000"	B	0	0	0	b	S_1
S_0	0	0	"0000000"	C	0	0	0	c	S_4
. S_0 .	. 0 .	. 0 .	. Other combinations than above .	. D .	. 0 ..	. 0 .	. 0 .	. d .	. S_0 .
S_0	0	0	X	E	0	0	0	e	S_0
S_0	0	1	X	F	0	0	0	f	S_0
S_0	1	0	X	G	0	0	0	g	S_0
S_0	1	1	X	H	0	0	0	h	S_0
S_4	X	0	X	I	0	0	0	i	S_5
S_4	X	1	X	J	0	0	0	j	S_4
S_5	X	0	X	K	0	0	1	k	S_0
S_5	X	1	X	L	0	0	0	l	S_5
S_1	X	0	X	M	0	0	0	m	S_2
S_1	X	1	X	N	0	0	0	n	S_1
S_2	X	0	X	O	0	0	0	o	S_3
S_2	X	1	X	P	0	0	0	p	S_2
S_3	X	0	X	Q	0	1	0	q	S_0
S_3	X	1	X	R	0	0	0	r	S_3
S_6	X	0	X	S	0	0	0	s	S_7
S_6	X	1	X	T	0	0	0	t	S_6
S_7	X	0	X	U	0	0	0	u	S_8
S_7	X	1	X	V	0	0	0	v	S_7
S_8	X	0	X	W	1	0	0	w	S_0
S_8	X	1	X	X	0	0	0	x	S_8

* For the purpose of simplification, not all the inputs and outputs are specified in this table.

In figure 3.54, a 3-DW TLP Header without data payload is shown. A 32-bit addressable Memory Read request is an example of such TLP.

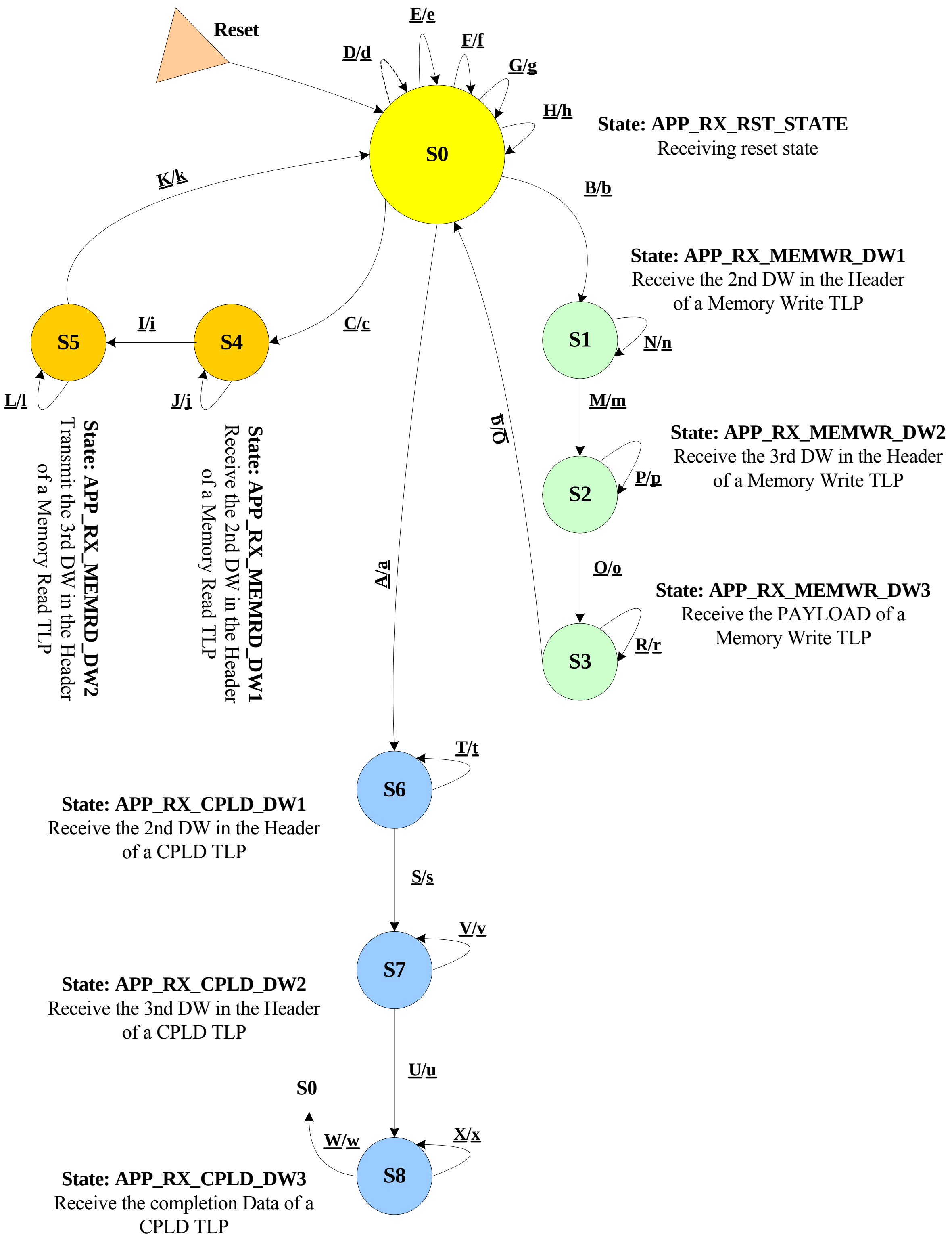

Reset
S0
D/d
E/e
F/f
G/g
H/h
State: APP_RX_RST_STATE
Receiving reset state
K/k
B/b
State: APP_RX_MEMWR_DW1
Receive the 2nd DW in the Header of a Memory Write TLP
S5
I/i
S4
C/c
S1
N/n
L/l
J/j
M/m
Q/q
State: APP_RX_MEMRD_DW2
Transmit the 3rd DW in the Header of a Memory Read TLP
State: APP_RX_MEMRD_DW1
Receive the 2nd DW in the Header of a Memory Read TLP
State: APP_RX_MEMWR_DW2
Receive the 3rd DW in the Header of a Memory Write TLP
S2
P/p
A/a
O/o
State: APP_RX_MEMWR_DW3
Receive the PAYLOAD of a Memory Write TLP
S3
R/r
T/t
S6
State: APP_RX_CPLD_DW1
Receive the 2nd DW in the Header of a CPLD TLP
S/s
V/v
S7
State: APP_RX_CPLD_DW2
Receive the 3nd DW in the Header of a CPLD TLP
U/u
S0
W/w
X/x
S8
State: APP_RX_CPLD_DW3
Receive the completion Data of a CPLD TLP

Figure 3.52 - PCIe Receiving State Machine State Diagram

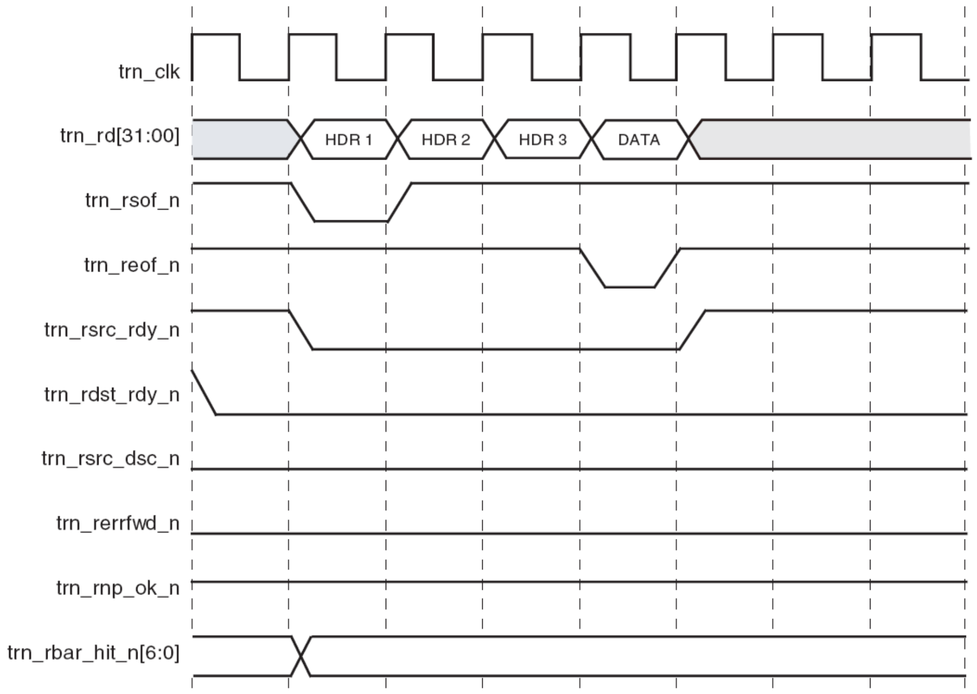


Figure 3.53 **-** *Received 32-bit Addressable Memory Write TLP [XUG167]*

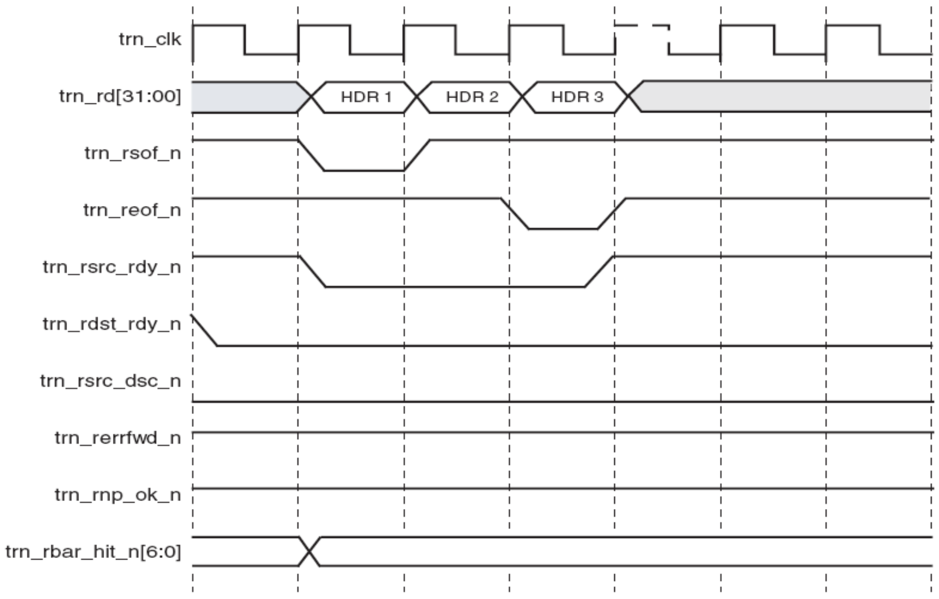


Figure 3.54 **-** *Received 32-bit addressable Memory Read TLP [XUG167]*

3.5.3.6 PCIe Configuration space Access READ/WRITE State Machine

Some of the registers within the PCIe configuration space can be accessed directly through the interfaces provided by the PCIe core. The contents of these registers can only be modified by Configuration Writes issued by the Root Complex. Changing the contents of these register from the user side is not possible.

Table 3.15 lists the Command and Status registers mapped directly to the configuration ports of the PCIe core.

Table 3.15 *- Registers mapped directly onto the configuration Interface of the core*

Register Name	Description
cfg_bus_number[7:0]	Configuration Bus Number: This register provides the assigned bus number to the core. Default value is 00h and over written whenever a Type0 configuration packet is received.
cfg_device_number[4:0]	Configuration Device Number: This register provides the assigned device number to the core. Default value is 00000b and over written whenever a Type0 configuration packet is received.
cfg_function_number[2:0]	Configuration Function Number: This register provides the function number of the core. This is hard wired to 000b.
cfg_status[15:0]	Configuration Status: PCI status register from the configuration space header.
cfg_command[15:0]	Configuration Command: PCI command register of the configuration space header.
cfg_dstatus[15:0]	Configuration Device Status: PCI Express PIPE device status register output.
cfg_dcommand[15:0]	Configuration Device Command: PCI Express PIPE device command register output.
cfg_lstatus[15:0]	Configuration Link Status: PCI Express PIPE link status register output.
cfg_lcommand[15:0]	Configuration Link Command: PCI Express PIPE link command register output.

A combination of cfg_bus_number [7:0], cfg_device_number [4:0] and cfg_function_number [2:0] forms the PCIe core identification. This ID is written by the Root Complex through the generation of Type0 Configuration Write. The designed Endpoint uses this number as a Requester ID for all the TLPs it generates, or as a Completer ID for all the TLPs it completes. The used PCIe core supports only one function. Therefore, the function number is hard wired to 000b.

cfg_status [15:0] advertises the Status Register of the PCI configuration space header. cfg_command [15:0] allows the user to see the value stored in the Command Register of the PCI configuration space, where cfg_command [0] indicates whether the IO Address Space Decoder is enabled or not, while cfg_command[1] specifies whether the memory address space decoder is activated or not. cfg_command [2] reflects enabling the PCIe core as a bus master.

cfg_dcommand [15:0] contains the information exists in the Device Control Register of the PCI Express Extended Capabilities. For example, cfg_dcommand [7:5] determines the maximum payload size allowed by this PCIe core.[1]

In order to access the other registers, within this configuration space, the PCIe configuration space Access State Machine was developed. In this state machine, the required events to generate Read and Write cycles are implemented. Writing access implemented in this machine does not work properly, because the PCIe core specifications do not allow writing to its configuration space.

When user wants to write the configuration space, the write cycle does not finish correctly, because the PCIe core does not provide the required reaction on the interfaces.

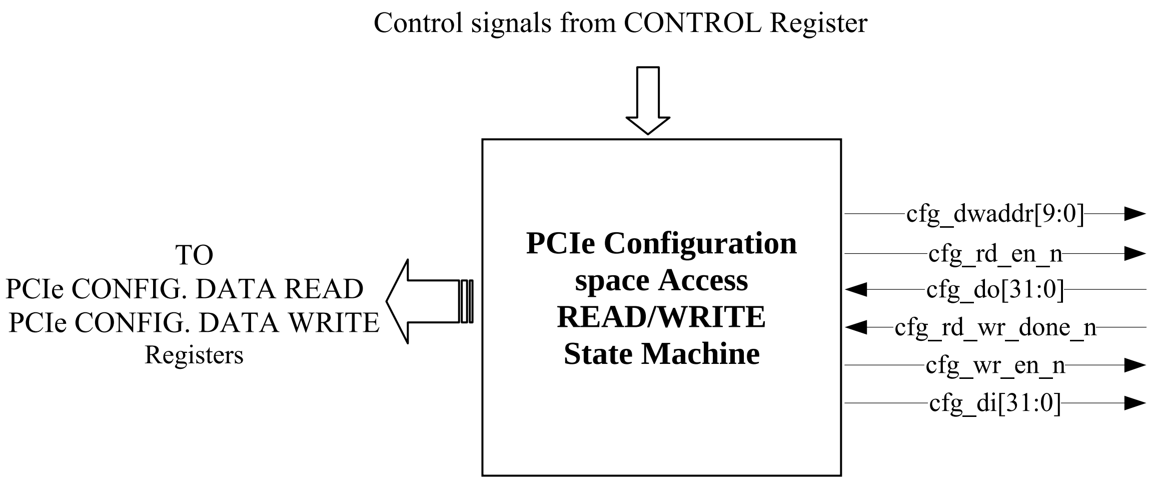


***Figure 3.55** - PCIe Configuration Space Access READ/WRITE State Machine*

Figure 3.55 shows the interfaces of the PCIe Configuration space Access Read/Write State Machine. Table 3.16 illustrates a simplified transition table of this state machine. Its corresponding state diagram is depicted in figure 3.56.

The main functionality of this state machine is to enable the accessing of the PCIe configuration space for the purpose of reading from this space by generating the subsequent events shown in figure 3.57.

In order to read the content of any register in the configuration space of the PCIe core, the state machine places the DWORD address of the required register on cfg_dwaddr [9:0]. This address points to two registers within this space. The required register can then be separated in the application program. The state machine simultaneously asserts cfg_rd_en_n. Once the PCIe core receives this signal, it loads the content of the addressed register on cfg_do [31:0].

[1] Refer to the PCI Express Base Specification [PXS05] for a detailed description of these Registers.

The state machine waits until cfg_rd_wr_done_n is asserted by the PCIe core. After the assertion of this signal, it reads the configuration data from cfg_do [31:0] as shown in figure 3.57. This figure shows an example of two consecutive reads from the Configuration Space.

Table 3.16 *- PCIe Configuration Space Access READ/WRITE State Machine Transition Table**

Current State	**Inputs**		**Input Vector**	**Outputs**		**Output Vector**	**Next State**
	control [4:5]	cfg_rd_wr_done_n		cfg_rd_en_n	cfg_wr_en_n		
S_0	"00"	X	A	1	1	a	S_0
S_0	"11"	X	B	1	1	b	S_0
S_0	"10"	X	C	1	1	c	S_1
S_0	"01"	X	D	1	1	d	S_{16}
S_1	X	1	E	0	1	e	S_1
S_1	X	0	F	1	1	f	S_2
S_2	X	X	G	1	1	g	S_3
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
S_{15}	X	X	G	1	1	g	S_0
S_{16}	X	1	H	1	0	h	S_{16}
S_{16}	X	0	I	1	1	i	S_{17}
S_{17}	X	X	J	1	1	j	S_{18}
S_{18}	X	X	J	1	1	j	S_{19}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
S_{23}	X	X	J	1	1	j	S_0

* For the purpose of simplification, not all the inputs and outputs are specified in this table.

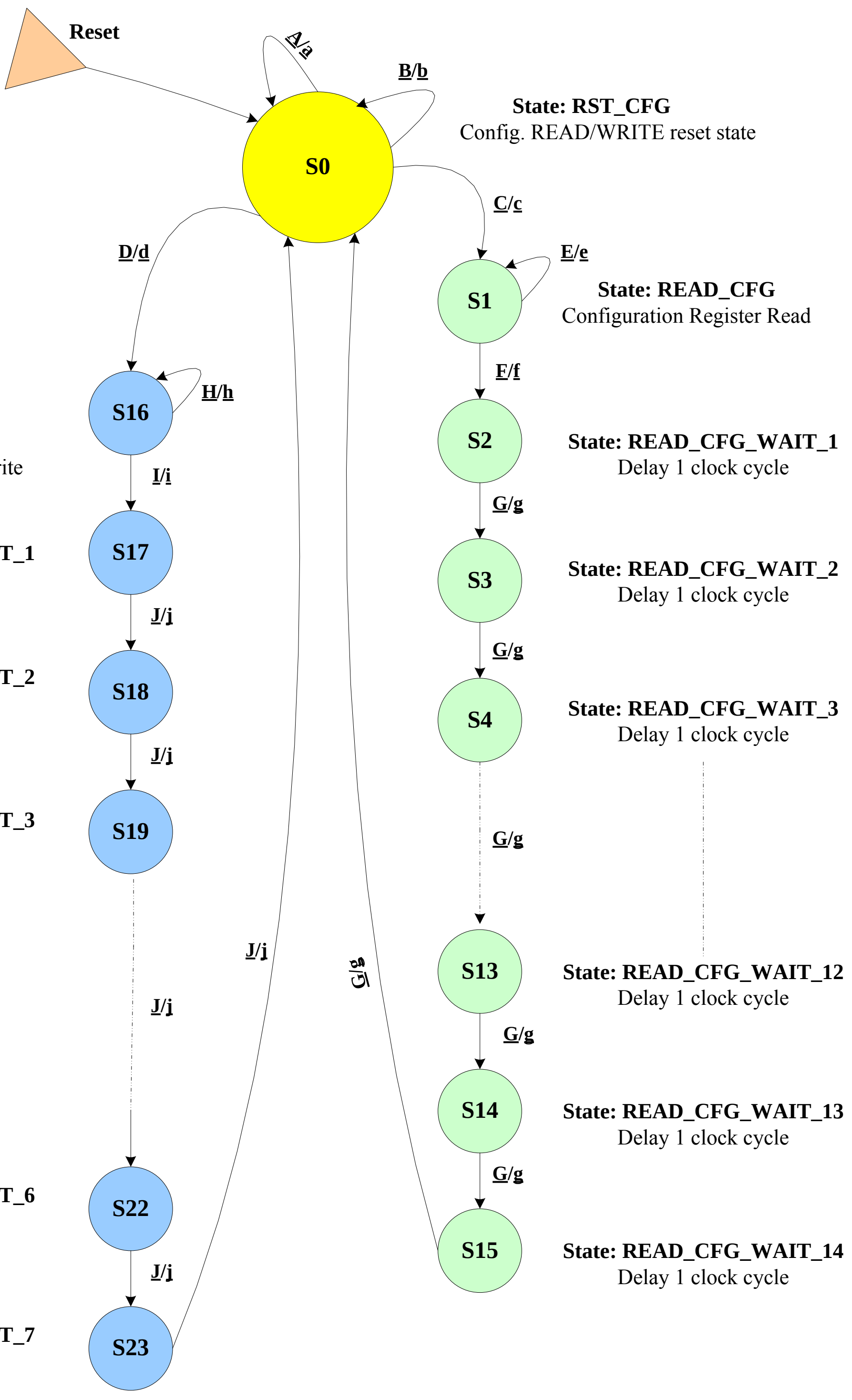


***Figure 3.56** - PCIe Configuration space Access READ/WRITE State Machine Bubble Diagram*

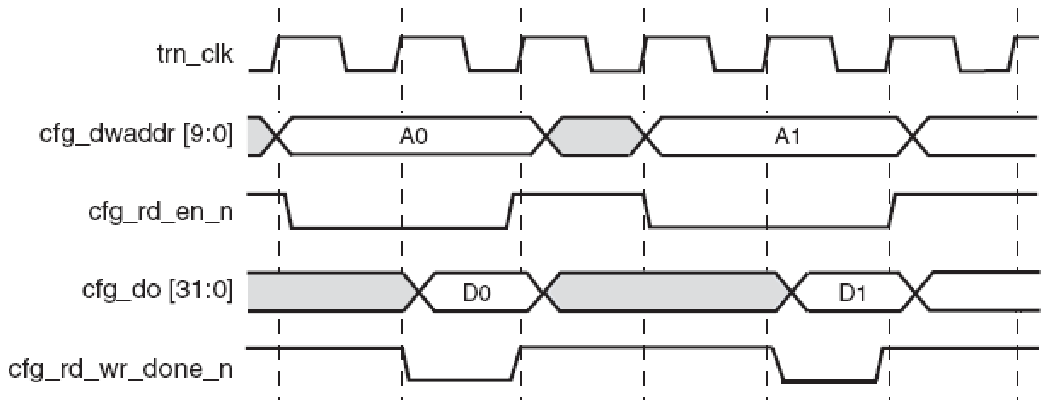


Figure 3.57 *- Reading of PCIe Configuration Space [XUG167]*

4 PCIe Endpoint Simulation

4.1 PCIe Testbench

The designed PCIe Endpoint was integrated in a top level Testbench to simulate its functionality. Figure 4.1 shows the top level of this Testbench (which is written in Verilog HDL). The figure depicts the hierarchy of this Testbench. In the top level named boardx01 (indicates a x1 PCIe design), the PCIe Downstream Port model, the Philips PHY and the Design Under Test (DUT) are instantiated.

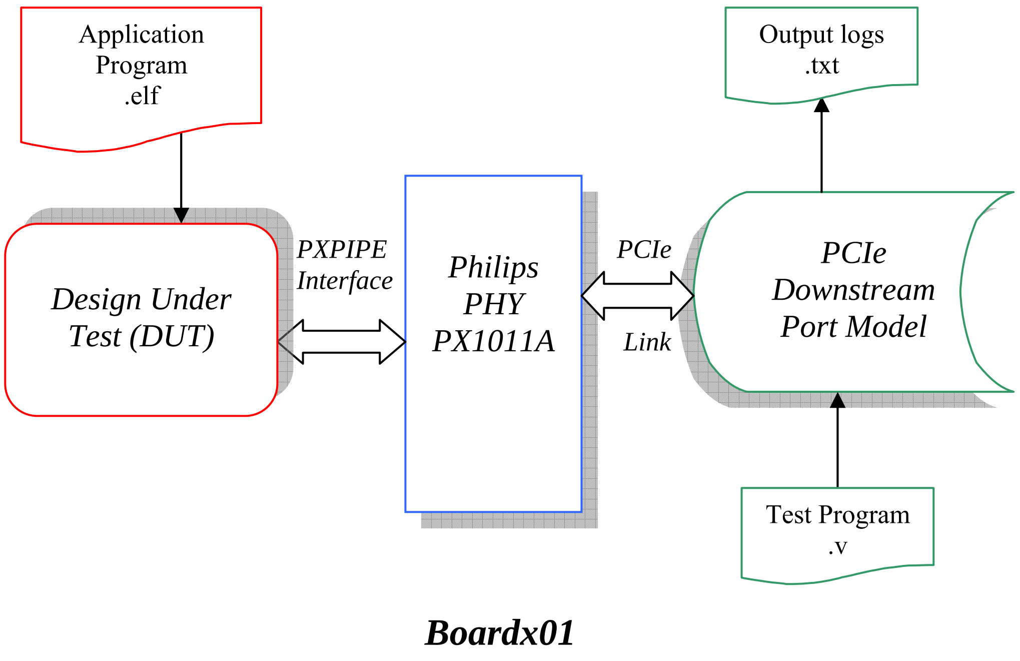


Figure 4.1 *- PCIe Testbench Top-level*

The following subsections explain each of these simulation models in details.

4.1.1 Philips PHY Simulation Model

The PX1011A behavioral model is a packaged model, which can be simulated in ModelSim or other standard Hardware Description Language (HDL) simulators. The IP Model Packager from Cadence was used to generate this model.

This model can be integrated in any simulator that supports either the IEEE standard 1499 – the open Model Interface, or the IEEE standard 1364 – the Verilog PLI 1.0 (Programming Language Interface).

The ModelSim simulator supports the PLI. PLI is a kind of an interface that defines a way for implementing tasks and functions that communicates with the used simulator through a defined C procedural interface.

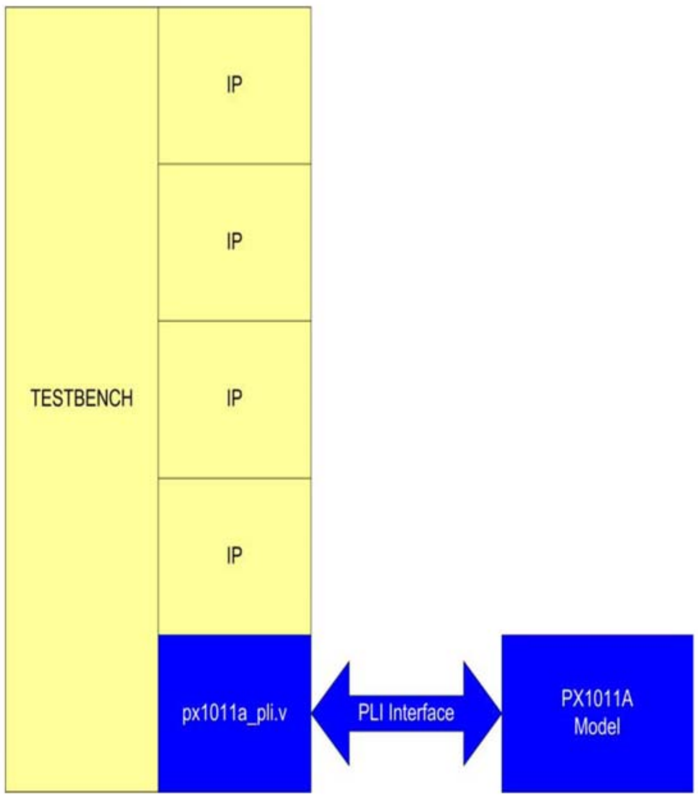


Figure 4.2 *- PX1011A Packaged Model [PUG05]*

For the model usage in ModelSim, one can either use the precompiled version "libpli.dll" provided with the package, or by compiling and linking of the adapter delivered with the package. For the purpose of this diploma work, the precompiled version, provided by NXP Semiconductors, was used as shown in figure 4.2.

4.1.2 Xilinx PCIe Downstream Port Simulation Model

In a PCIe Testbench, a simulation model is needed to implement the functionality of the Root Complex and the PCIe switch in the PCIe topology shown in figure 4.3.

The Xilinx PCIe Downstream Port simulation model, offered by Xilinx when generating the core, was used for the purpose of simulation in the PCIe based system.

The main functionality of this model is to generate downstream TLPs from the CPU to the PCIe Endpoint and to receive upstream TLPs from the PCIe Endpoint to the CPU.

In addition to the main functionality, this model does the initialization of the PCIe core's configuration registers, verifies the transmission and reception of TLPs by generating TLP logs, and provides a kind of Test Programming Interface (TPI), which enables the simulation of PCIe Endpoint device.

This model is written in Verilog HDL, all source codes are provided to give the designer the possibility to customize the test cases for the best usage and to save time in the creation of PCIe testbench.

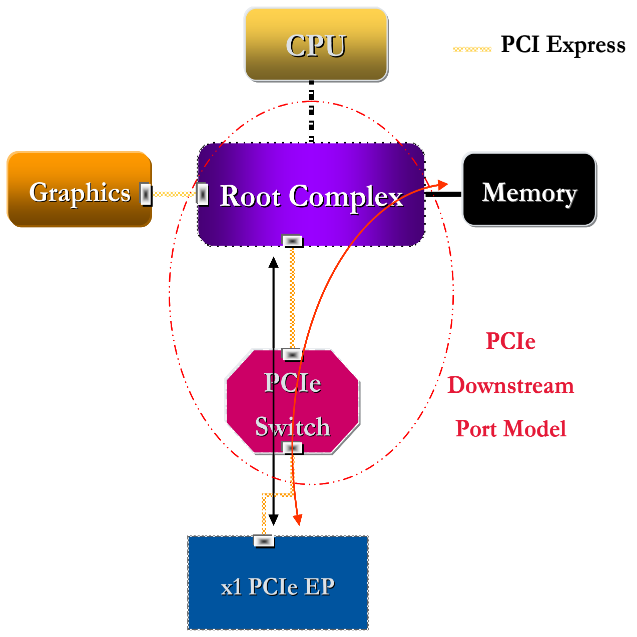


Figure 4.3 *- PCIe Downstream Port Model*

Figure 4.4 depicts the different components of the PCIe Downstream Port model. DSPORT implements the functionality of the physical and the Data Link Layers of the PCIe protocol, which are responsible for the electrical signalling interfaces to the PCIe link and the reliable transport of TLPs across the PCIe link, respectively.

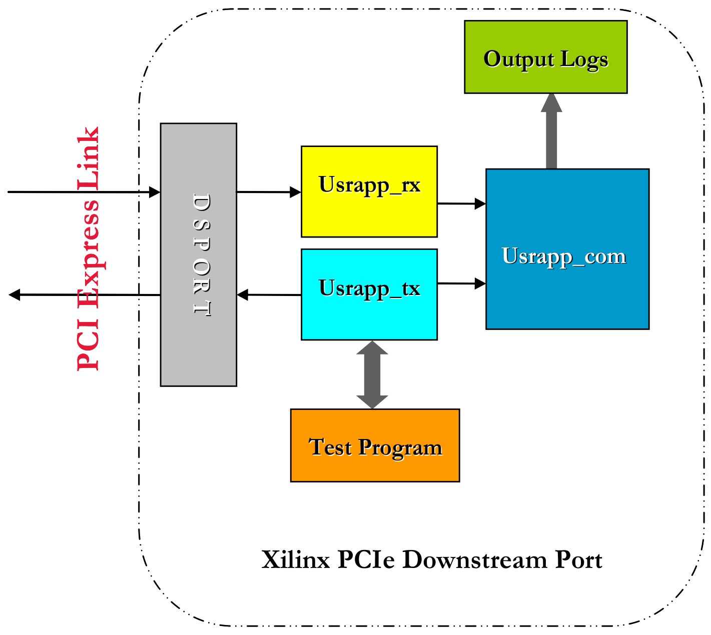


Figure 4.4 *- Functional Block Diagram of the PCIe Downstream Port Model [XUG341]*

The *Usrapp_tx* demonstrates a transmission engine, which is responsible for the generation of downstream TLPs to simulate the functionality of the PCIe Endpoint. The *Usrapp_rx* implements all the functions needed to receive upstream TLPs generated by the PCIe Endpoint.

Both Usrapp_tx and Usrapp_rx models use common tasks, which are implemented by the *Usrapp_com* model.

Customized tests can be included by the mean of Test Program Interface (TPI). These tests are written in Verilog HDL. The user can indicate the test case to be carried out, when invoking the simulator.

For the purpose of functional verification, the model implements an output logging mechanism. Three different text files are generated, when running a defined task. One of the files summarizes the received TLPs, another shows the transmitted TLPs, and the third file includes error messages, in case any errors are detected.

4.1.3 Design Under Test (DUT)

Figure 4.5 shows a top level of the DUT, which consists of two sub-models: the Microblaze based system and the PCIe core simulation models.

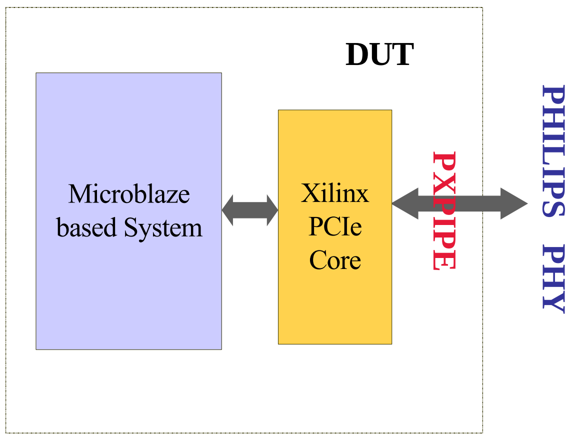


Figure 4.5 *- Top level of DUT Model*

4.1.3.1 Xilinx PCIe Core Simulation Model

The generation of the core using Xilinx CORE Generator resulted in several models, one of these model was the PCIe core simulation model. This simulation model is a VHDL structural verification model that uses simulation primitives, which may not truly implement the device. Such a model is not synthesizable.

4.1.3.2 Microblaze based system Simulation Model

The Microblaze based simulation model is illustrated in figure 4.6. Shown are the different components building up this system. This simulation model was generated using the Xilinx Platform Studio (XPS). This tool allows the initialization of the on-chip BRAM with the compiled application program in the Executable Link Format (ELF). This application program is executed by the Microblaze.

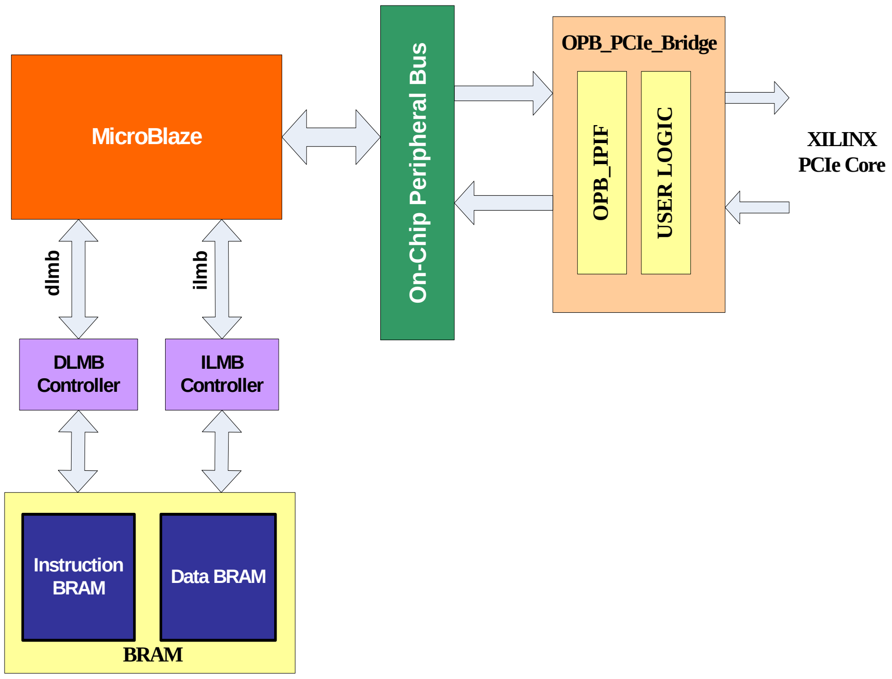


Figure 4.6 - *Top level of Microblaze based System Simulation Model*

4.2 C Application Program

The BRAM is initialized with the application program described next. This program is written in C, using special C-functions provided by Xilinx. It is compiled into an executable link format and loaded onto the on-chip BRAM.

The application program is divided into several segments. Theses segments are executed sequentially by the embedded processor. The following is an explanation of each segment in this program, a flowchart is provided for each of these segments.

Segment 1: Initialization and configuration of the PCIe Core

Figure 4.7 shows the flowchart of this segment. In this segment, the assigned address to the PCIe peripheral is obtained. A time delay is required to allow the plug and play software to configure the PCIe core. During this time delay, the following actions take place:

- System Reset deassertion.
- Transaction Reset deassertion
- Transaction Link Up activation.
- Configuration of the PCIe core.
- Endpoint configuration as Bus master.

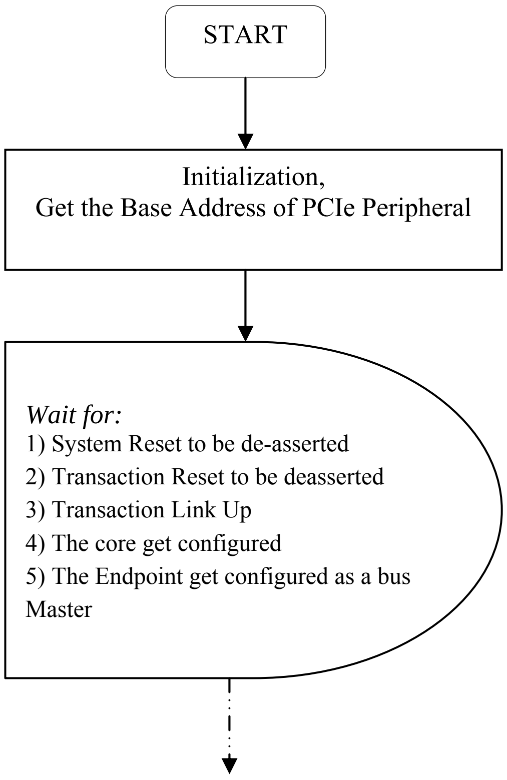

START
Initialization,
Get the Base Address of PCIe Peripheral
Wait for:
1) System Reset to be de-asserted
2) Transaction Reset to be deasserted
3) Transaction Link Up
4) The core get configured
5) The Endpoint get configured as a bus Master

Figure 4.7 *- Segment 1: Initialization and configuration of the PCIe Core*

Segment 2: PCIe Core Configuration Space Read

In this segment, shown in figure 4.8, the Microblaze generates a PCIe configuration space read cycle to read one of the configuration registers within the PCIe configuration space. In order to generate such a read cycle, the Microblaze does the following actions:

- Firstly, it generates a write cycle to access the CONTROL register. It set cfg_read to "1" within this register to enable the PCIe configuration space read process and writes the DWORD address of the targeted configuration register.
- Secondly, it reads the required data by generating a read cycle to access the PCIe CONFIG DATA READ register.
- Finally, it generates a write cycle to access the CONTROL register. It sets cfg_read to "0" to disable the PCIe configuration space read process.

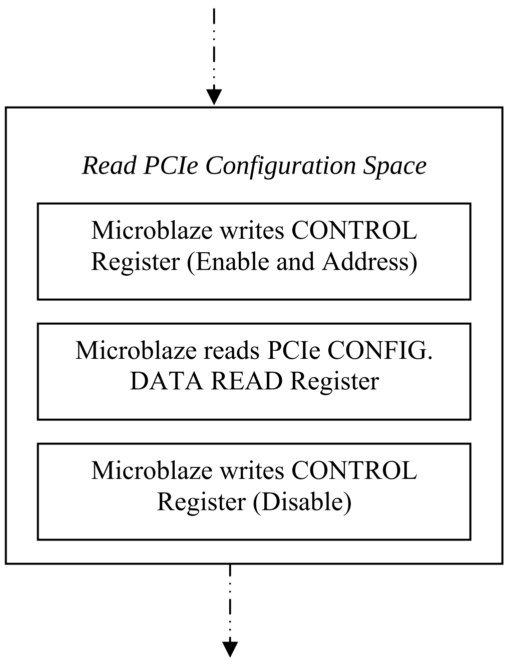

Read PCIe Configuration Space
Microblaze writes CONTROL Register (Enable and Address)
Microblaze reads PCIe CONFIG. DATA READ Register
Microblaze writes CONTROL Register (Disable)

Figure 4.8 *- Segment 2: PCIe Core Configuration Space Read*

Segment 3: PCIe Core Configuration Space Write

Figure 4.9 shows the flow diagram of this segment. In this segment, the Microblaze generates a PCIe configuration space write cycle to write onto one of the configuration registers within the PCIe configuration space. In order to generate such a write cycle, the Microblaze does the following actions:

- Firstly, it generates a write cycle to access the CONTROL register. It sets cfg_write to "1", within this register, to enable the PCIe configuration space write process and writes the DWORD address of the targeted configuration register.
- Secondly, it writes the required data by generating a write cycle to write the data onto the PCIe CONFIG WRITE READ register.
- Finally, it generates a write cycle to access the CONTROL register. It sets cfg_write to "0" to disable the PCIe configuration space write process.

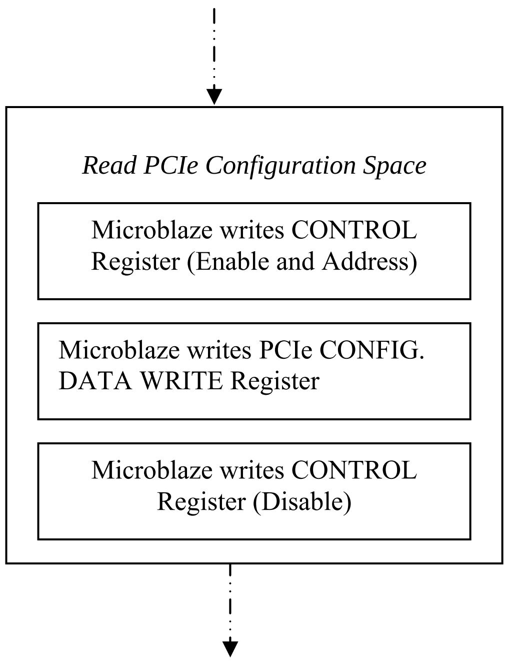

Read PCIe Configuration Space
Microblaze writes CONTROL Register (Enable and Address)
Microblaze writes PCIe CONFIG. DATA WRITE Register
Microblaze writes CONTROL Register (Disable)

Figure 4.9 *- Segment 3: PCIe Core Configuration Space Write*

Segment 4: Bus Master Enable

Figure 4.10 shows the flow diagram of this segment. In this segment, the Microblaze checks whether the core is configured as a bus master or not. It reads the STATUS register and then checks cfg_command (2) for bus mastering.

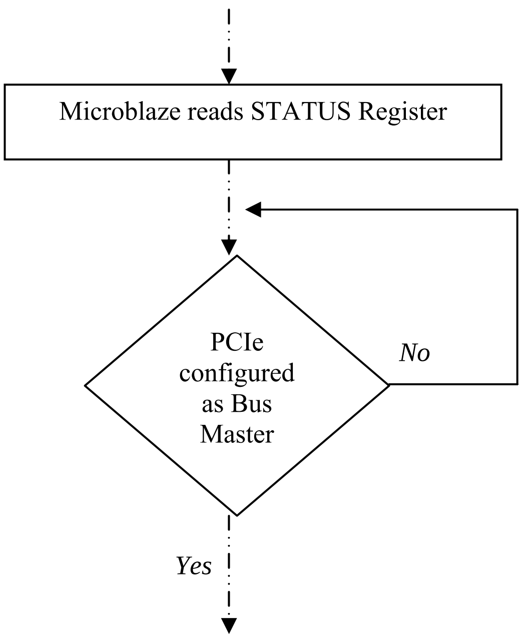

Microblaze reads STATUS Register
PCIe configured as Bus Master
No
Yes

Figure 4.10 *- Segment 4: Bus Master Enable*

Segment 5: Generating of a Memory Write TLP

Figure 4.11 shows the flow diagram of this segment. In this segment, the Microblaze sends the Header and the data Payload to the USER LOGIC model, in order to generate a Memory Write TLP. In this segment, the Microblaze does the following actions:

- Firstly, if the core is enabled as a bus master, it starts sending the Header and Payload by carrying out the following events:
  1. It writes the first DW in the Header onto the MWR/MRD HDR DW1 register.
  2. It writes the second DW in the Header onto the MWR/MRD HDR DW2 register.
  3. It writes the third DW in the Header onto the MWR/MRD HDR DW3 register.
  4. It writes the data Payload onto the MWR PAYLOAD register.
- Secondly, it generates a write cycle to access the CONTROL register. It sets both *mem_wr_gen* and *master_enable* to "1" in order to activate the Memory Write TLP generation process and to confirm the enabling of the core as bus master.
- Thirdly, it reads the STATUS register to check whether the TLP was sent or not. It checks *mem_wr_transmitted*. If this signal is = 1, it continues to the next segment. Otherwise, it keeps reading this register and controlling this signal.

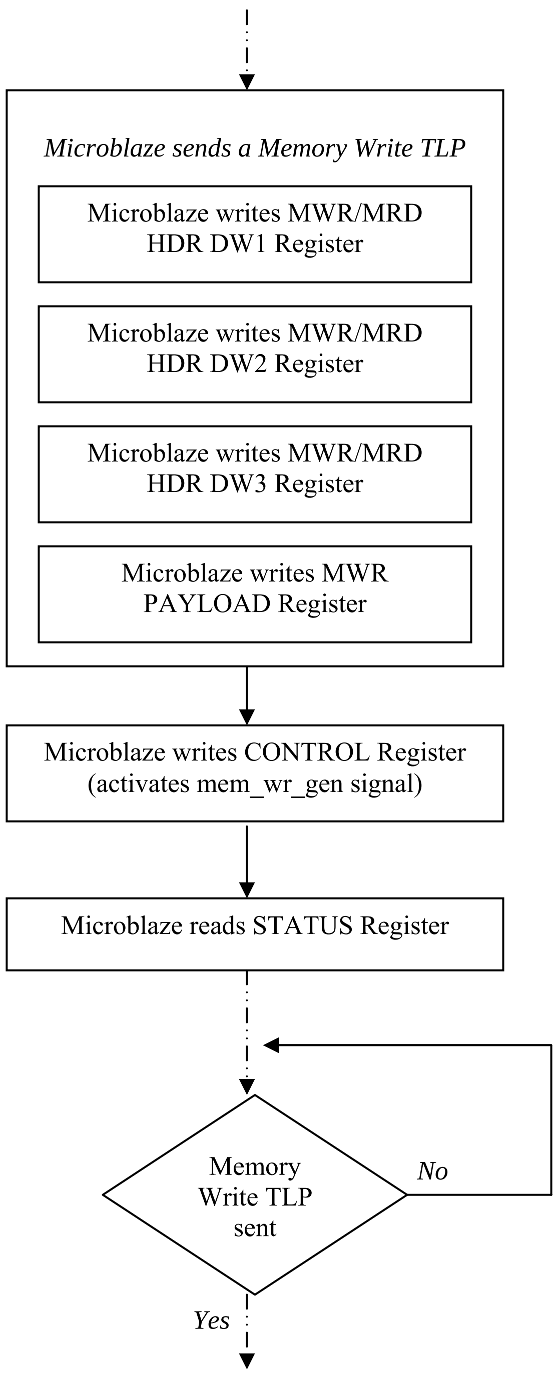

Microblaze sends a Memory Write TLP
Microblaze writes MWR/MRD HDR DW1 Register
Microblaze writes MWR/MRD HDR DW2 Register
Microblaze writes MWR/MRD HDR DW3 Register
Microblaze writes MWR PAYLOAD Register
Microblaze writes CONTROL Register (activates mem_wr_gen signal)
Microblaze reads STATUS Register
Memory Write TLP sent
No
Yes

Figure 4.11 *- Segment 5: Generating of a Memory Write TLP*

Segment 6: Generating of a Memory Read TLP

Figure 4.12 shows the flow diagram of this segment. In this segment, the Microblaze sends the Header to the USER LOGIC model in order to generate a Memory Read TLP. In this segment the Microblaze does the following actions:

- Firstly, if a Memory Write TLP was sent, it waits for a while then starts sending the first DW in the Header of the Memory Read TLP to the MWR/MRD HDR DW1 register. The same information stored in the registers MWR/MRD HDR_DW2 and MWR/MRD HDR_DW3, are used as the second and third DWs of the Header.
- Secondly, it generates a write cycle to access the CONTROL register. It sets *mem_wr_gen* to "0" and *mem_rd_gen* to "1" in order to deactivate the Memory Write TLP generation process and to activate the Memory Read TLP generation process. It also confirms the enabling of the core as bus master.
- Thirdly, it reads the STATUS register to check whether the Memory Read TLP was sent or not. It checks *mem_rd_transmitted*. If this signal is = 1, it continues to the next segment. Otherwise, it keeps reading this register and controlling this signal.

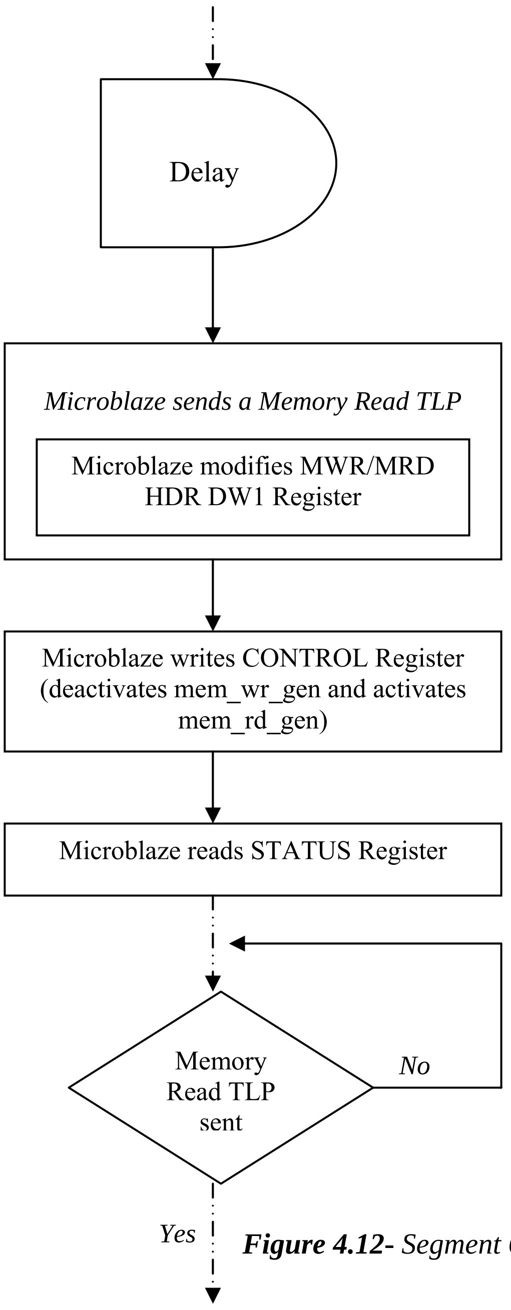

Delay
Microblaze sends a Memory Read TLP
Microblaze modifies MWR/MRD HDR DW1 Register
Microblaze writes CONTROL Register (deactivates mem_wr_gen and activates mem_rd_gen)
Microblaze reads STATUS Register
Memory Read TLP sent
No
Yes

Figure 4.12- *Segment 6: Generating of a Memory Read TLP*

Segment 7: Receiving of a CPLD TLP

The Memory Read TLP is a non-posted transaction, which requires a completion TLP either with Data (CPLD) or without data (CPL). In this segment, the Microblaze receives a CPLD as a consequence of its Memory Read request. In this segment, the Microblaze does the following actions:

- Firstly, it reads the STATUS register to check whether a CPLD TLP was received or not. It checks *cpld_received*. If this signal is = 1, it starts receiving the CPLD TLP. Otherwise, it keeps reading this register and controlling this signal.
- Secondly, if CPLD TLP was received, it reads the four registers, REC. MWR/MRD/CPLD DW1, REC. CPLD DW2, REC. CPLD DW3, and REC. CPLD DW4 Registers, successively, as shown in figure 4.13.

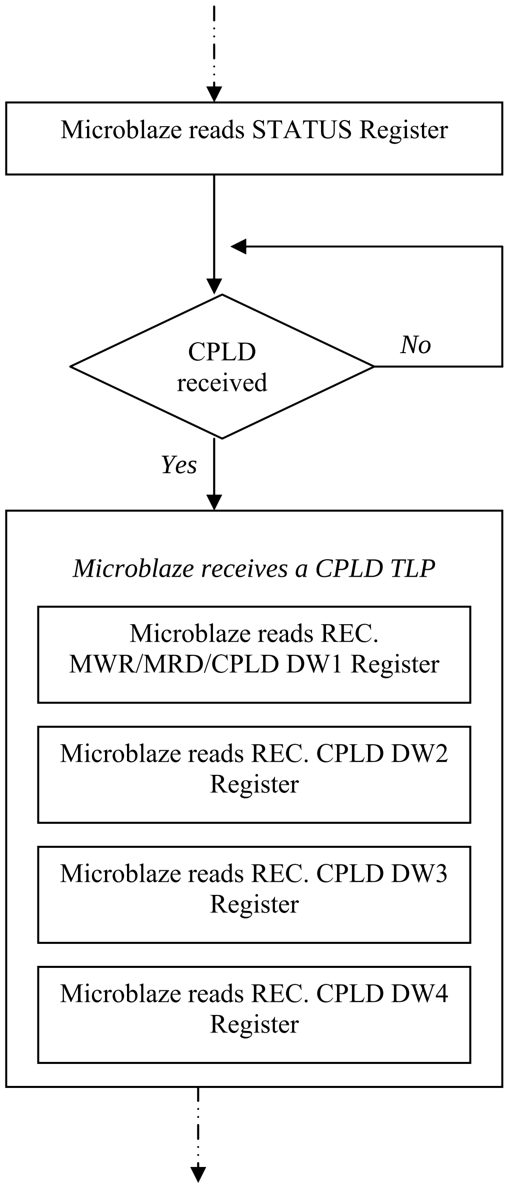

Microblaze reads STATUS Register
CPLD received
No
Yes
Microblaze receives a CPLD TLP
Microblaze reads REC. MWR/MRD/CPLD DW1 Register
Microblaze reads REC. CPLD DW2 Register
Microblaze reads REC. CPLD DW3 Register
Microblaze reads REC. CPLD DW4 Register

Figure 4.13 *- Segment 7: Receiving of a CPLD*

Segment 8: Receiving of a Memory Write TLP

In this segment, the Microblaze receives a Memory Write TLP. It does the following actions:

- Firstly, it reads the STATUS register to check whether a Memory Write TLP was received or not. It checks *mem_wr_received*. If this signal is = 1, it starts receiving the TLP. Otherwise, it keeps reading this register and controlling this signal.
- Secondly, if a Memory Write TLP was received, it reads the four registers, REC. MWR/MRD/CPLD DW1, REC. MWR/MRD DW2, REC. MWR/MRD DW3, and REC. MWR/MRD DW4 Registers, successively, as shown in figure 4.14.

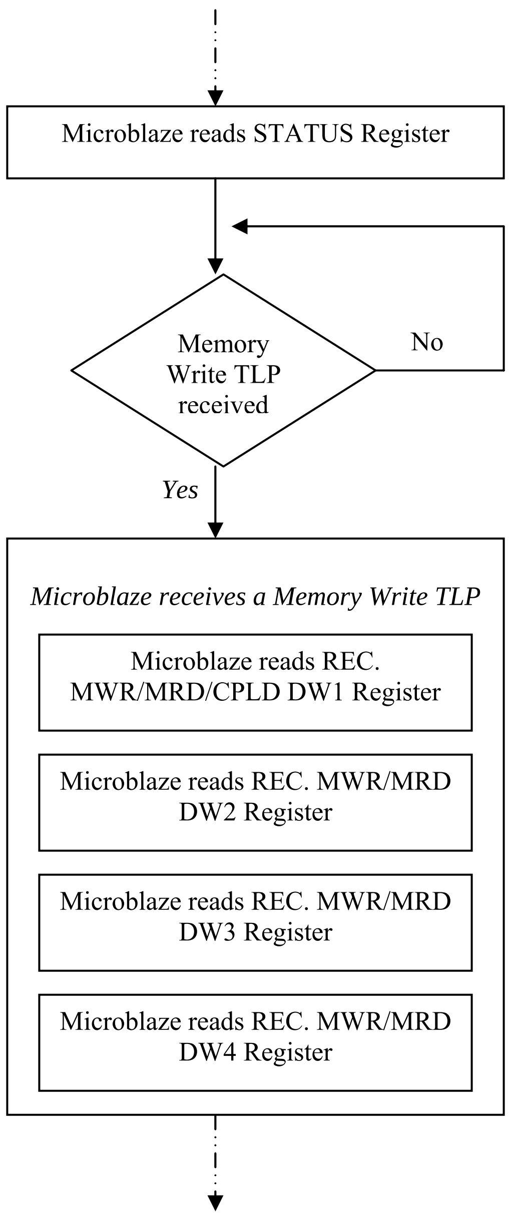

Microblaze reads STATUS Register
Memory Write TLP received
No
Yes
Microblaze receives a Memory Write TLP
Microblaze reads REC. MWR/MRD/CPLD DW1 Register
Microblaze reads REC. MWR/MRD DW2 Register
Microblaze reads REC. MWR/MRD DW3 Register
Microblaze reads REC. MWR/MRD DW4 Register

Figure 4.14 *- Segment 8: Receiving of a Memory Write TLP*

Segment 9: Receiving of a Memory Read TLP

In this segment, the Microblaze receives a Memory Read TLP. It does the following actions:

- Firstly, it reads the STATUS register to check whether a Memory Read TLP was received or not. It checks *mem_rd_received*. If this signal is = 1, it starts receiving the TLP. Otherwise, it keeps reading this register and controlling this signal.
- Secondly, if a Memory Read TLP was received, it reads the four registers, REC. MWR/MRD/CPLD DW1, REC. MWR/MRD DW2, REC. MWR/MRD DW3, and REC. MWR/MRD DW4 Registers, successively, as shown in figure 4.15.

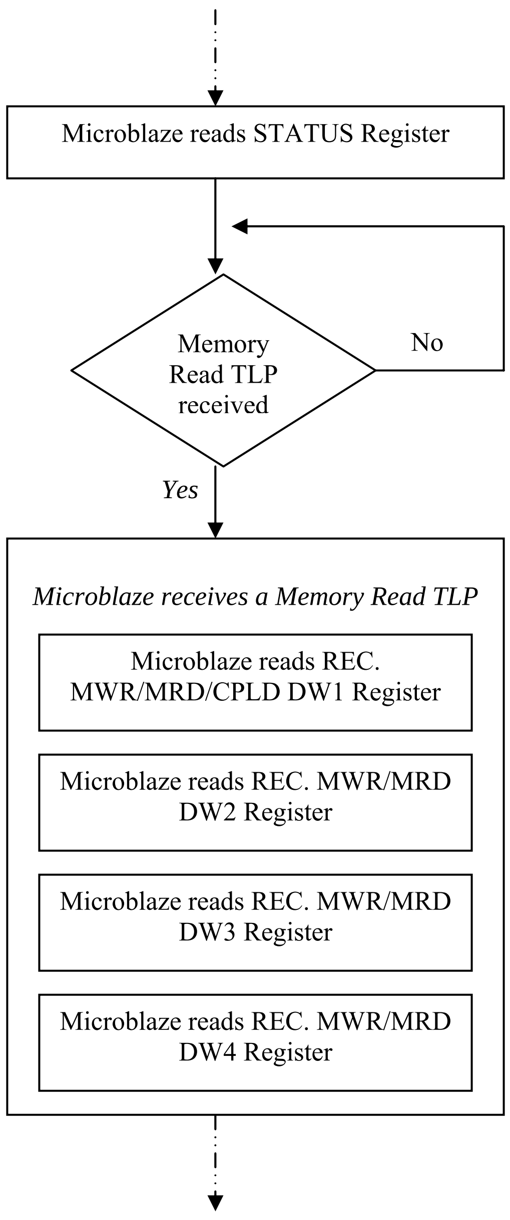

Microblaze reads STATUS Register
Memory Read TLP received
No
Yes
Microblaze receives a Memory Read TLP
Microblaze reads REC. MWR/MRD/CPLD DW1 Register
Microblaze reads REC. MWR/MRD DW2 Register
Microblaze reads REC. MWR/MRD DW3 Register
Microblaze reads REC. MWR/MRD DW4 Register

Figure 4.15 *- Segment 9: Receiving of a Memory Read TLP*

Segment 10: Generation of a CPLD TLP

Figure 4.16 shows the flow diagram of this segment. In this segment, the Microblaze enables the generation of a Completion with Data TLP. This CPLD is generated in the USER LOGIC model. In this segment, the Microblaze does the following actions:

- Firstly, it generates a write cycle to access the CONTROL register. It sets *compl_gen* to “1”, *mem_rd_gen* and mem_wr_gen to “0” in order to deactivate both the Memory Write and Read TLP generation processes and to activate the CPLD TLP generation process. It also confirms the enabling of the core as bus master.
- Secondly, it reads the STATUS register to check whether the TLP was sent or not. It checks *cpld_transmitted*. If this signal is = 1, it finishes. Otherwise, it keeps reading this register and controlling this signal.

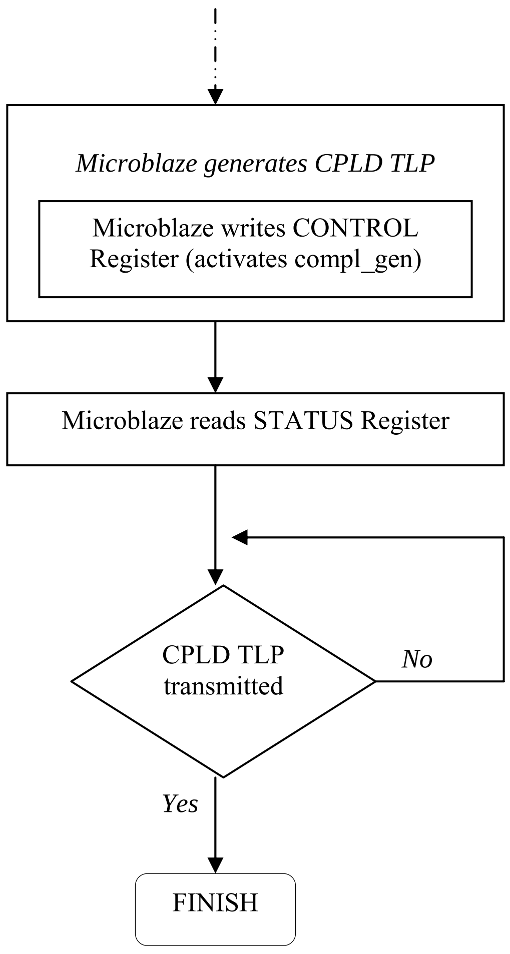

Microblaze generates CPLD TLP
Microblaze writes CONTROL Register (activates compl_gen)
Microblaze reads STATUS Register
CPLD TLP transmitted
No
Yes
FINISH

Figure 4.16 - *Segment 10: Generation of a CPLD TLP*

4.3 Simulation Flow

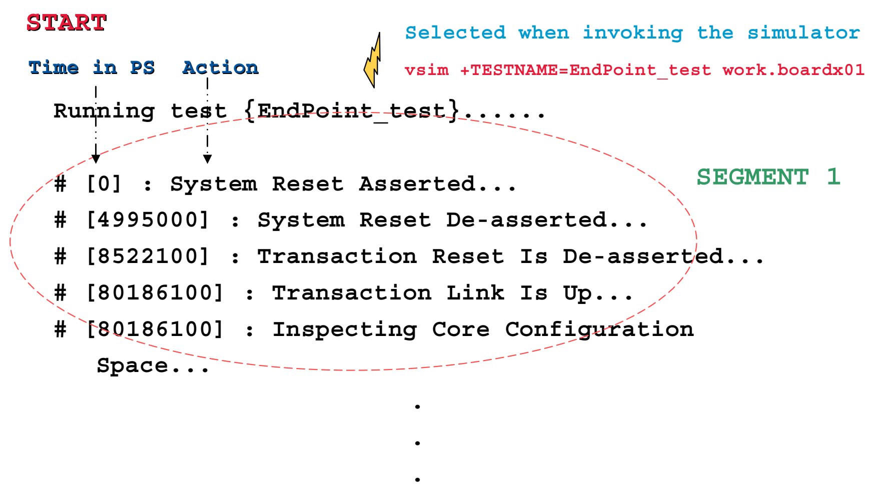
Figures 4.17, 4.20, 4.26, 4.28, and 4.31 show the simulation flow carried out to verify the functionality of the designed Endpoint. In these figures the transcript window of the ModelSim Simulator is shown. Each of these figures is related to one or more of the C application program segments executed by the Microblaze. One should differentiate between this C application program executed by the Microblaze and the test program executed by the PCIe Downstream Port simulation model. The simulation flow is divided into the following stages:

Stage 1: Initialization and configuration of the PCIe Core

The simulation starts by selecting the required test when invoking the simulator as shown in figure 4.17. This stage of the simulation flow is related to segment 1 of the C application program. In this stage, the test program of the PCIe Downstream Port Simulation model waits the system reset to deassert as well as the endpoint's trn_lnk_up signal to assert, before it starts configuring the Endpoint.

The waveforms shown in figure 4.18 depict the PXPIPE Interfaces. In this figure, shown are the Physical Layer Packets (PLPs) which issued by the physical layer of the Downstream port and terminated at the physical layer of the PCIe Endpoint. Such PLPs are used during the *Link Training and Initialization*.

The cursor in this figure indicates the moment when RXVALID changes from “0” to “1”, at this moment, symbol lock takes place and valid data are available on RXDATA and RXDATAK.



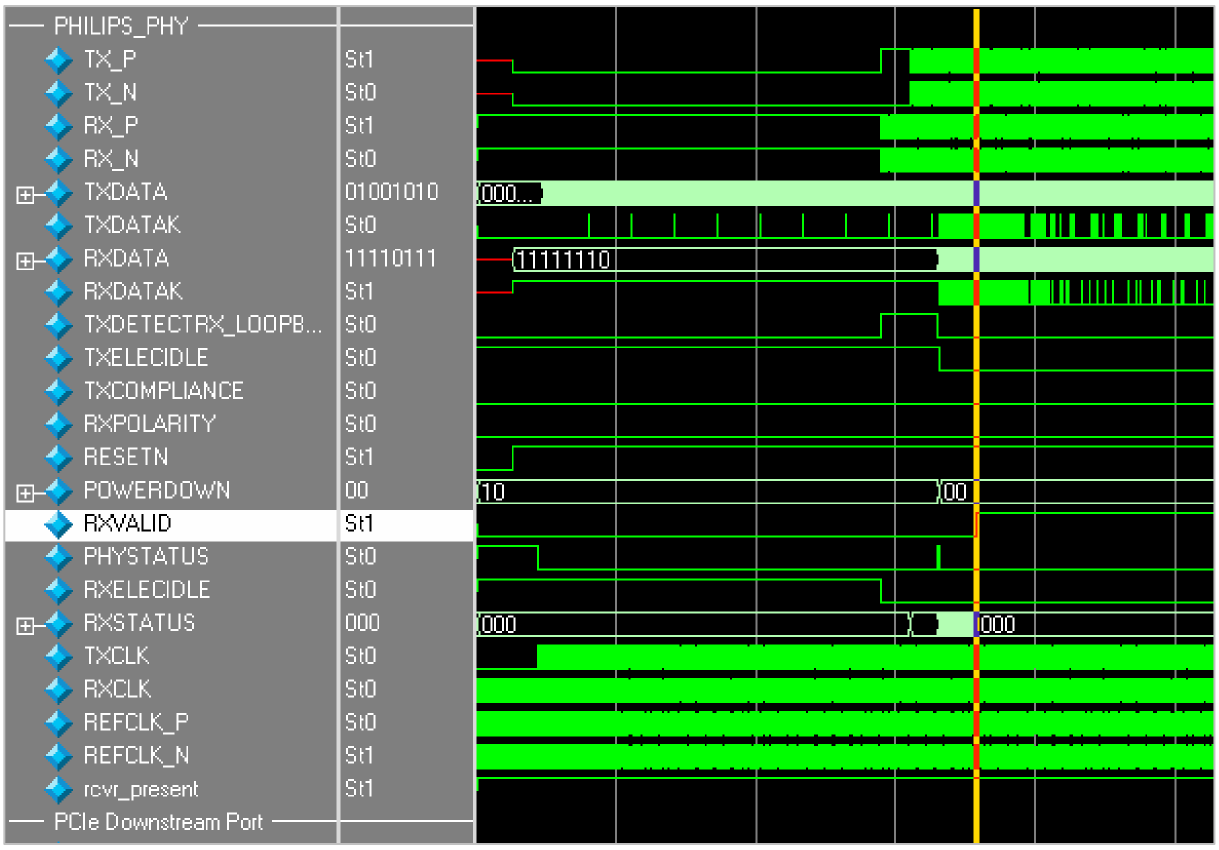
```
vsim +TESTNAME=EndPoint_test work.boardx01
```

```
  Running test {EndPoint_test}......

# [0] : System Reset Asserted...
# [4995000] : System Reset De-asserted...
# [8522100] : Transaction Reset Is De-asserted...
# [80186100] : Transaction Link Is Up...
# [80186100] : Inspecting Core Configuration
   Space...
                        .
                        .
                        .
```

Figure 4.17 *- Simulation Flow Stage 1*

Figure 4.19 shows the interfaces of the Downstream Port model. In this figure, the system reset is indicated by the first cursor on the most left. The second cursor shows the moment when trn_lnk_up is activated.



PHILIPS_PHY	
TX_P	St1
TX_N	St0
RX_P	St1
RX_N	St0
TXDATA	01001010
TXDATAK	St0
RXDATA	11110111
RXDATAK	St1
TXDETECTRX_LOOPB...	St0
TXELECIDLE	St0
TXCOMPLIANCE	St0
RXPOLARITY	St0
RESETN	St1
POWERDOWN	00
RXVALID	St1
PHYSTATUS	St0
RXELECIDLE	St0
RXSTATUS	000
TXCLK	St0
RXCLK	St0
REFCLK_P	St0
REFCLK_N	St1
rcvr_present	St1
PCIe Downstream Port	

***Figure 4.18** - Symbol Lock/PXPIPE Waveforms*

The test program then carries out a series of Type 0 Configuration Writes and Reads to the Endpoint's PCI configuration space shown in figure 2.6. It determines the memory and IO requirements of the Endpoint, and then programs the Endpoint's Base Address Registers in order to make the Endpoint device ready to receive TLPs from the PCIe Downstream Port model. These Configuration Write and Read TLPs are indicated between the second and the third cursors shown in figure 4.19.

The test program cycles through all the Endpoint's BARs and determines whether they are enabled or disabled. If they are enabled, it determines their type, whether they are 32-bit memory, 64-bit memory or IO spaces as shown in figure 4.20.

Referring to figure 3.15, the PCIe core was configured to support only 32-bit memory space of 64 Kbytes and a starting address of ffff0000h, by configuring BAR0. Figure 4.20 emphasizes that the test program found the same configuration. After this inspection the test program starts setting the core configuration space. The procedure of setting this space is illustrated in figure 4.21.

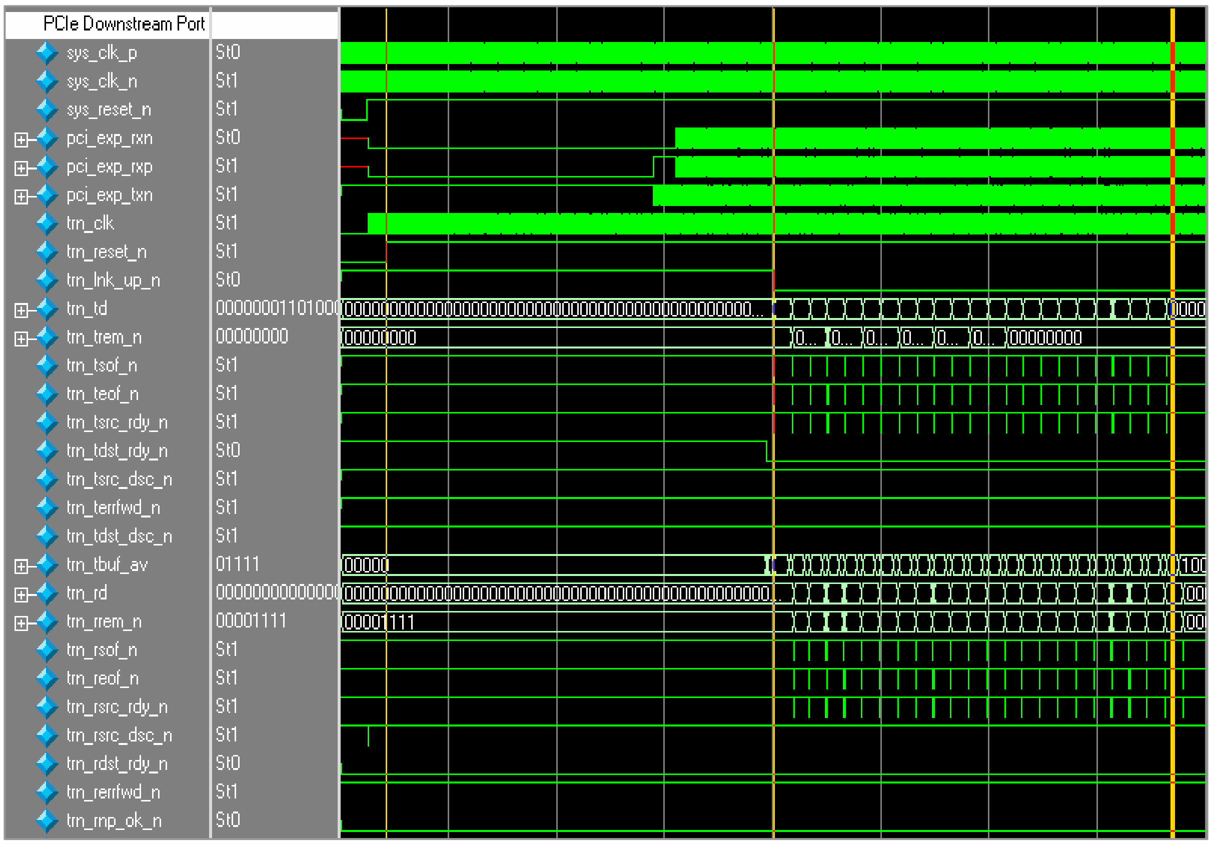


Figure 4.19 - *PCIe Downstream Port Waveforms*

```
                                  .
                                  .
                                  .     64 KB
# [126330000] PCI EXPRESS BAR MEMORY/IO MAPPING PROCESS BEGUN...
#  BAR 0: VALUE = 00000000 RANGE = ffff0000 TYPE =  MEM32 MAPPED
#  BAR 1: VALUE = 00000000 RANGE = 00000000 TYPE =  DISABLED
#  BAR 2: VALUE = 00000000 RANGE = 00000000 TYPE =  DISABLED
#  BAR 3: VALUE = 00000000 RANGE = 00000000 TYPE =  DISABLED
#  BAR 4: VALUE = 00000000 RANGE = 00000000 TYPE =  DISABLED
#  BAR 5: VALUE = 00000000 RANGE = 00000000 TYPE =  DISABLED
#  EROM : VALUE = 00000000 RANGE = 00000000 TYPE =  DISABLED
# [126330000] : Setting Core Configuration Space...
                                  .
                                  .
# [184890000] : PCIe core is configured as Bus Master
                                  .
                                  .
```

Figure 4.20 - *Simulation Flow Stage 1 (Continued)*

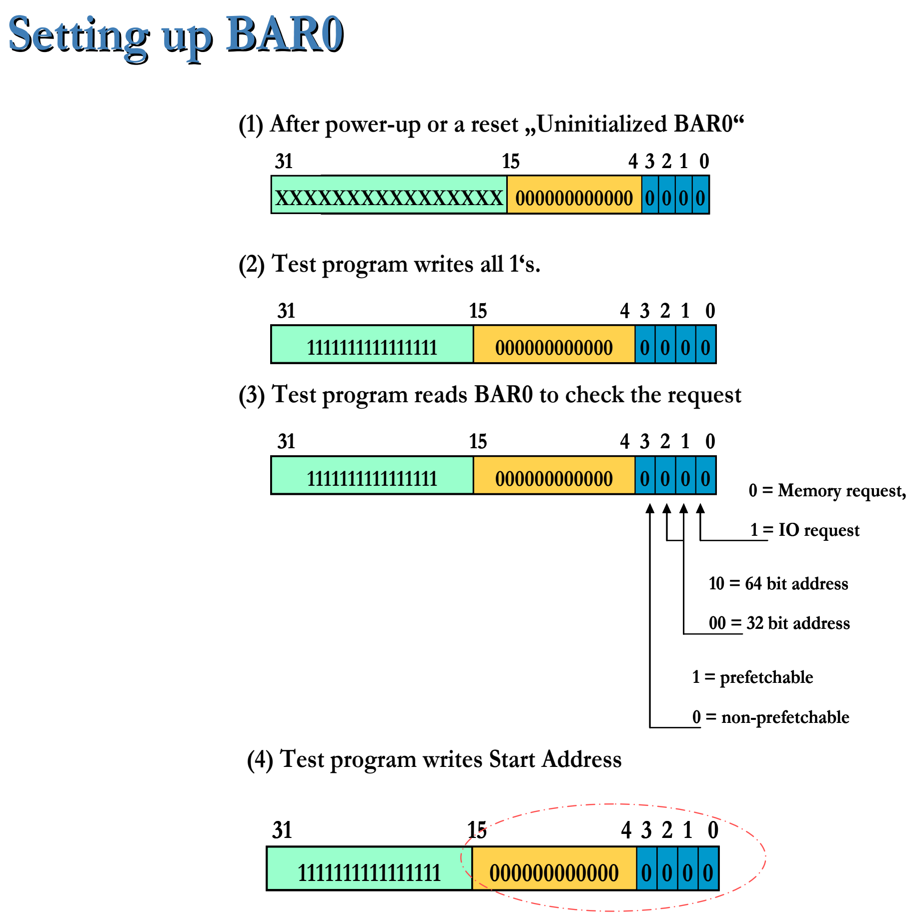


Figure 4.21 *- Setting of PCIe Configuration Space*

The Type0 Configuration Write to and Read from Endpoint's PCI Base Address Register 0 (BAR0) TLPs are logged out in the transmitting text file generated when running a defined task as shown in figure 4.22. The PCIe Endpoint completes the Read request with a completion without data TLP (CPL), which is received by the Downstream Port model. Figure 4.23 depicts this TLP which is logged out in the receiving text file. The waveform of this TLP is depicted in figure 4.25.

The final step in setting up the BAR0 (as illustrated in figure 4.21) is to write the starting address for BAR0. This Configuration Write TLP is depicted in figure 4.24 which again shows the transmitting output logging.

The procedure of setting BAR0 is illustrated here. The same steps are carried out when setting BAR1 to BAR5 and the expansion ROM Base Address.

```
[80282000]: Config Write Type 0 Frame

Traffic Class: 0x0
TD: 0
EP: 0
Attributes: 0x0
Length: 0x001
Requester Id: 0x01a0
Tag: 0x00
Last and First Byte Enables: 0x0f
Completer Id: 0x01a0
Register Address: 0x010

0xff
0xff
0xff
0xff


[83578000]: Config Read Type 0 Frame

Traffic Class: 0x0
TD: 0
EP: 0
Attributes: 0x0
Length: 0x001
Requester Id: 0x01a0
Tag: 0x01
Last and First Byte Enables: 0x0f
Completer Id: 0x01a0
Register Address: 0x010
```

Figure 4.22 - *Transmission Output Logging*

```
[83866000]: Completion Without Data Frame

Traffic Class: 0x0

TD: 0
EP: 0
Attributes: 0x0
Length: 0x000
Completer Id: 0x01a0
Completion Status: 0x0
Requester Id: 0x01a0
Tag: 0x00
```

Figure 4.23 - *Reception Output Logging*

```
[126426000]: Config Write Type 0 Frame

      Traffic Class: 0x0
      TD: 0
      EP: 0
      Attributes: 0x0
      Length: 0x001
      Requester Id: 0x01a0
      Tag: 0x00
      Last and First Byte Enables: 0x0f
      Completer Id: 0x01a0
      Register Address: 0x010

      0x00
      0x00
      0x00
      0x00
```

Figure 4.24 **-** *Transmitting Output Logging/Writing BAR0 Starting Address*

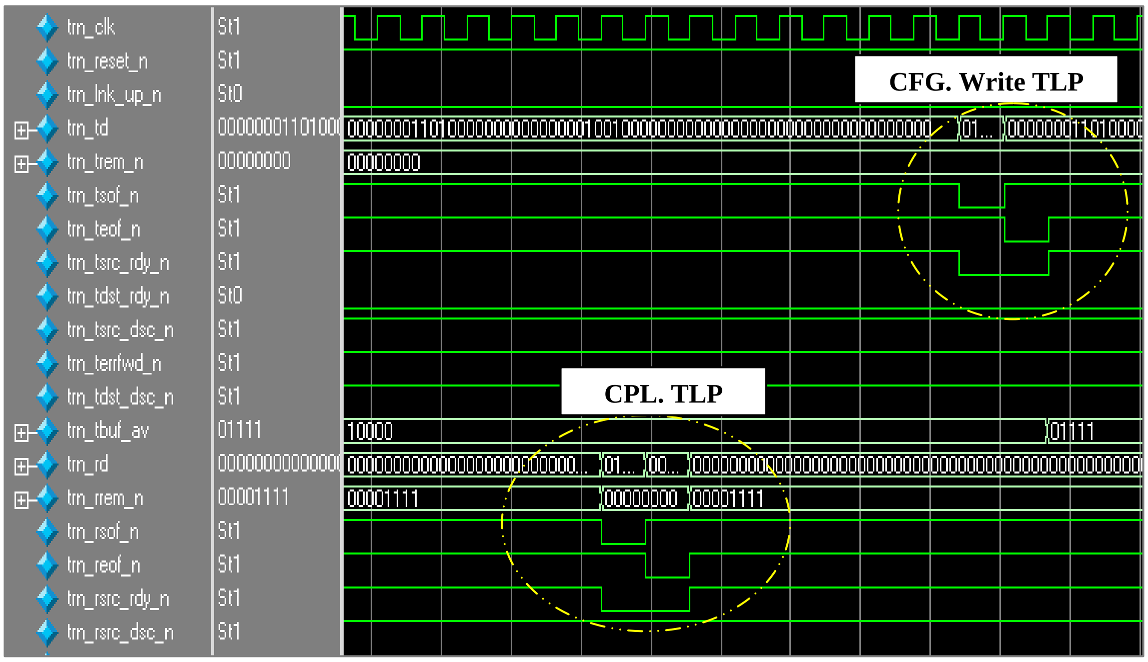


Figure 4.25 **-** *Waveforms of Configuration Write and CPL TLPs*

In addition to assigning the starting address of BAR0, the test program writes to both the PCIe command and device control registers in order to configure the core as bus master (as illustrated in figure 4.20) and to indicate the maximum payload size.

Stage 2: PCIe Core Configuration Space Access

This stage is related to segments 1 and 2 of the C application program. In this stage the Microblaze accesses the PCIe configuration space, reading from and writing to this space as shown in figure 2.26. The waveforms of this stage are illustrated in figure 4.27. As mentioned before, the write cycle does not finish properly due to the fact of having the PCIe core does not allow such a write access to it's configuration space.

```
                              .
                              .
# ** Note: Microblaze reads PCIe Core Configuration Space
# Time: 187018568 ps  Iteration: 4  Instance:
/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/pcie_i
_0/pcie_ip_0/user_logic_i
# ** Note: Microblaze writes PCIe Core Configuration Space
# Time: 187514568 ps  Iteration: 4  Instance:
/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/pcie_i
_0/pcie_ip_0/user_logic_i
# [187962000] : TSK_PARSE_FRAME on Receive
                              .
                              .
```

Figure 4.26 - *Simulation Flow Stage 2*

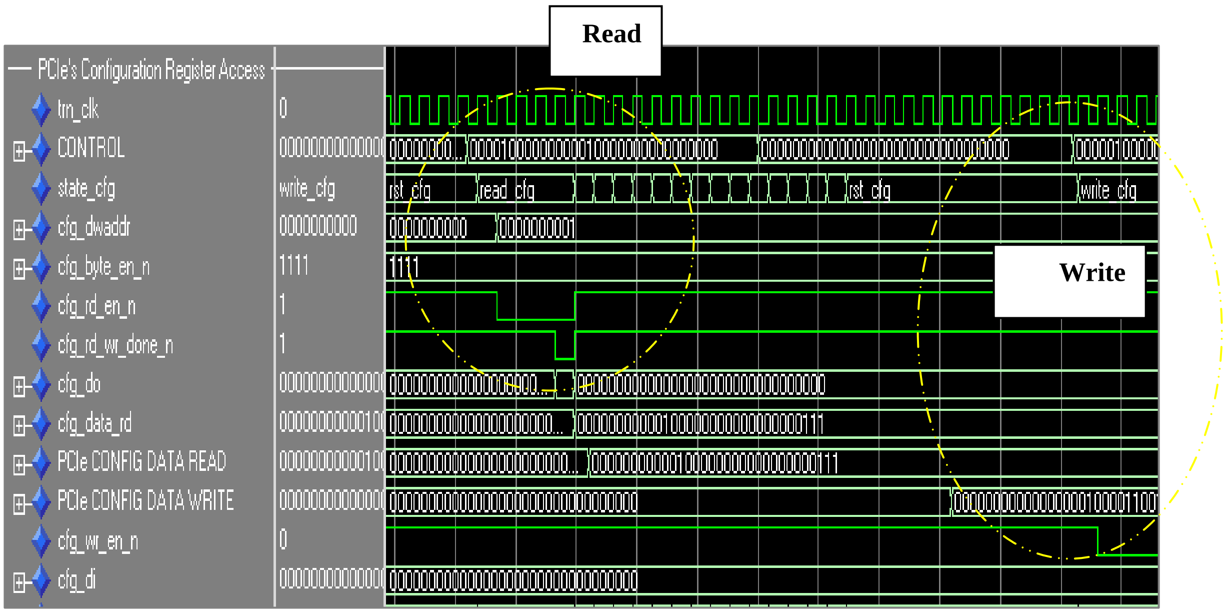
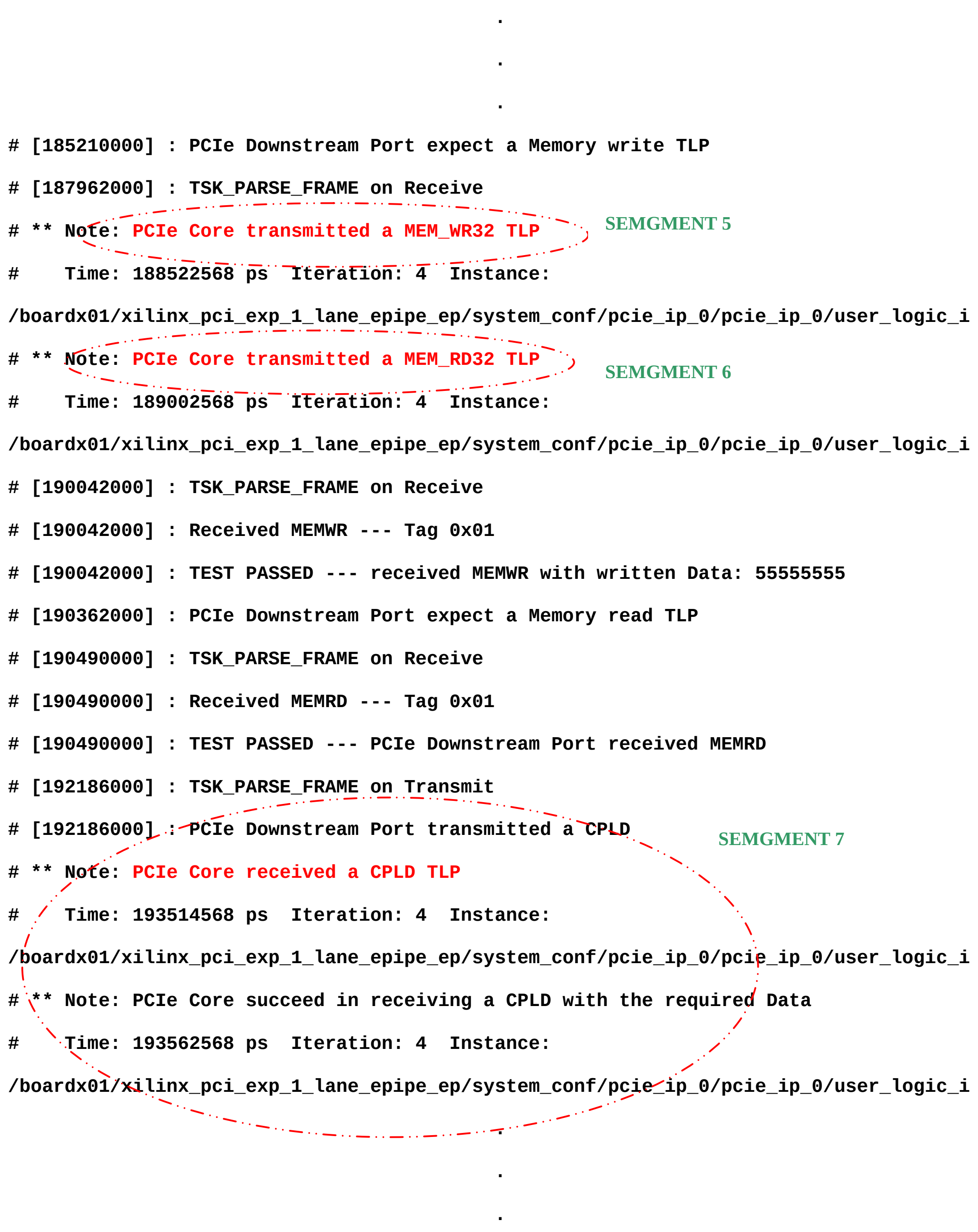


Figure 4.27 - *PCIe Core Configuration Space Access Waveforms*

Stage 3: Endpoint generates Memory Write/Read TLPs

This stage is related to segments 5 to 7 of the C application program. Figure 4.28 depicts this stage, where the Endpoint transmits a Memory Write TLP followed by a Memory Read TLP. The PCIe Downstream Port Model receives these TLPs and responds with a CPLD TLP as shown in the figure.



```
                                        .
                                        .
                                        .
# [185210000] : PCIe Downstream Port expect a Memory write TLP
# [187962000] : TSK_PARSE_FRAME on Receive
# ** Note: PCIe Core transmitted a MEM_WR32 TLP
#    Time: 188522568 ps  Iteration: 4  Instance:
/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/pcie_ip_0/pcie_ip_0/user_logic_i
# ** Note: PCIe Core transmitted a MEM_RD32 TLP
#    Time: 189002568 ps  Iteration: 4  Instance:
/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/pcie_ip_0/pcie_ip_0/user_logic_i
# [190042000] : TSK_PARSE_FRAME on Receive
# [190042000] : Received MEMWR --- Tag 0x01
# [190042000] : TEST PASSED --- received MEMWR with written Data: 55555555
# [190362000] : PCIe Downstream Port expect a Memory read TLP
# [190490000] : TSK_PARSE_FRAME on Receive
# [190490000] : Received MEMRD --- Tag 0x01
# [190490000] : TEST PASSED --- PCIe Downstream Port received MEMRD
# [192186000] : TSK_PARSE_FRAME on Transmit
# [192186000] : PCIe Downstream Port transmitted a CPLD
# ** Note: PCIe Core received a CPLD TLP
#    Time: 193514568 ps  Iteration: 4  Instance:
/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/pcie_ip_0/pcie_ip_0/user_logic_i
# ** Note: PCIe Core succeed in receiving a CPLD with the required Data
#    Time: 193562568 ps  Iteration: 4  Instance:
/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/pcie_ip_0/pcie_ip_0/user_logic_i
                                        .
                                        .
                                        .
```

Figure 4.28 *- Simulation Flow Stage 3*

The waveforms of the transmitted Write and Read TLPs from the PCIe Endpoint are illustrated in figure 4.29 which also shows the PCIe transmitting state machine.

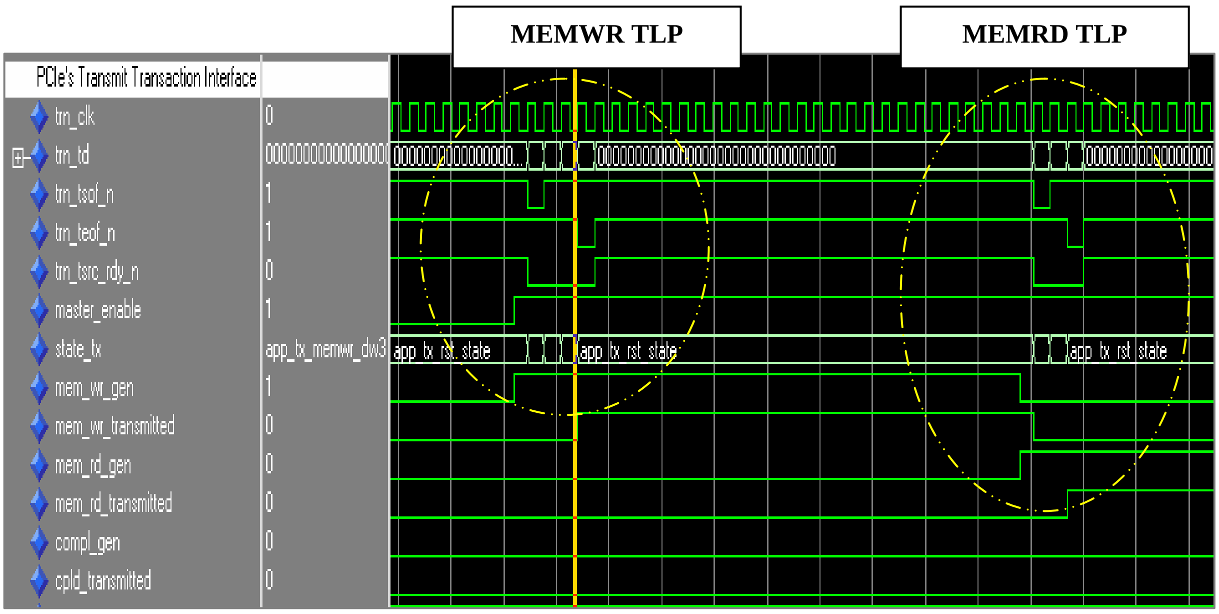


Figure 4.29 - *Simulation Flow Stage 3 Waveforms*

Stage 4: PCIe Downstream Port Model generates Memory Writes/Reads TLPs

This stage is related to segments 8 to 10 of the C application program. Figure 4.31 shows this stage, where the PCIe Downstream Port Model transmits a Memory Write TLP followed by a Memory Read TLP. The Endpoint receives these TLPs and responds with a CPLD TLP as depicted by the waveform shown in the figure 4.30.

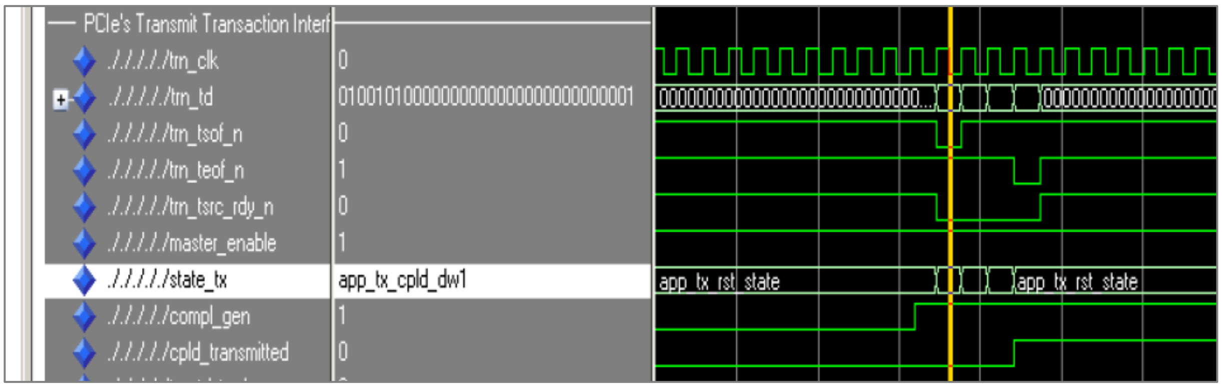
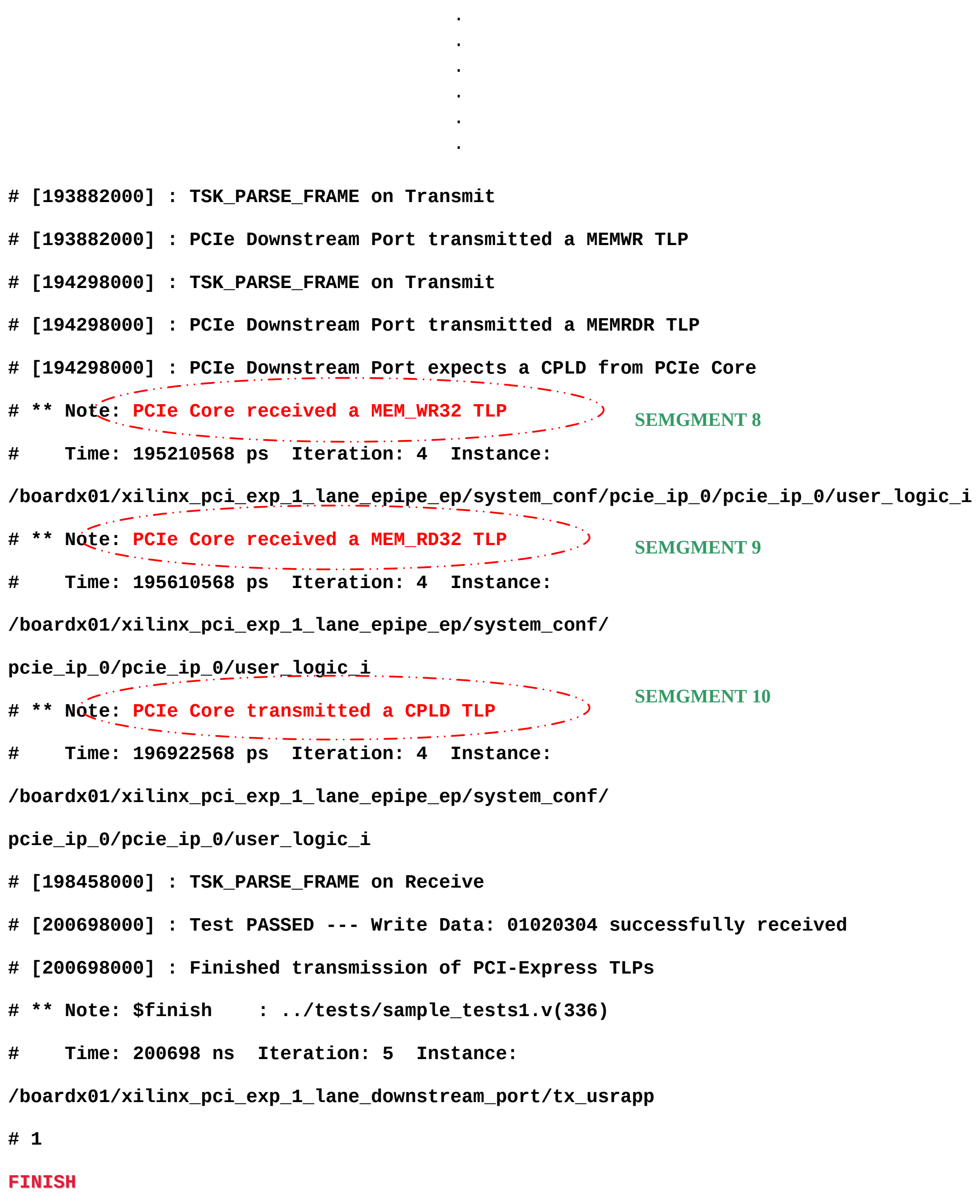

PCIe's Transmit Transaction Interf
./././././trn_clk 0
./././././trn_td 0100101000000000000000000000001
./././././trn_tsof_n 0
./././././trn_teof_n 1
./././././trn_tsrc_rdy_n 0
./././././master_enable 1
./././././state_tx app_tx_cpld_dw1
./././././compl_gen 1
./././././cpld_transmitted 0
app_tx_rst_state

Figure 4.30 - *Simulation Flow Stage 4 Waveforms*



```
.
.
.
.
.
.

# [193882000] : TSK_PARSE_FRAME on Transmit

# [193882000] : PCIe Downstream Port transmitted a MEMWR TLP

# [194298000] : TSK_PARSE_FRAME on Transmit

# [194298000] : PCIe Downstream Port transmitted a MEMRDR TLP

# [194298000] : PCIe Downstream Port expects a CPLD from PCIe Core

# ** Note: PCIe Core received a MEM_WR32 TLP

#    Time: 195210568 ps  Iteration: 4  Instance:

/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/pcie_ip_0/pcie_ip_0/user_logic_i

# ** Note: PCIe Core received a MEM_RD32 TLP

#    Time: 195610568 ps  Iteration: 4  Instance:

/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/

pcie_ip_0/pcie_ip_0/user_logic_i

# ** Note: PCIe Core transmitted a CPLD TLP

#    Time: 196922568 ps  Iteration: 4  Instance:

/boardx01/xilinx_pci_exp_1_lane_epipe_ep/system_conf/

pcie_ip_0/pcie_ip_0/user_logic_i

# [198458000] : TSK_PARSE_FRAME on Receive

# [200698000] : Test PASSED --- Write Data: 01020304 successfully received

# [200698000] : Finished transmission of PCI-Express TLPs

# ** Note: $finish    : ../tests/sample_tests1.v(336)

#    Time: 200698 ns  Iteration: 5  Instance:

/boardx01/xilinx_pci_exp_1_lane_downstream_port/tx_usrapp

# 1

FINISH
```

Figure 4.31 **-** *Simulation Flow Stage 4*

4.4 Test Cases Summary

Several test cases were conducted to verify the functionality of this system. The following is a summary of these test cases:

Test case 1:

In this test case (shown in figure 4.32), the CPU generates a Memory Write TLP to write data of 1 DW payload size to a memory mapped location within the Endpoint. It then generates a Memory Read TLP to read this data.

The Endpoint responds to this Memory Read request by generating a Completion with the required Data CPLD TLP. The CPU receives this TLP and terminates the transaction.

Test case 2:

In this test case, the Endpoint, which is configured as a bus master, generates a Memory Write TLP to write data of 1 DW payload size to a location within the System Memory. The Root Complex receives this TLP and writes the data onto the specified location. The Endpoint then generates a Memory Read TLP to read the same data, it has written. The Root Complex receives this TLP and in turn accesses the System Memory to get the required data, once it has the data, it generates a Completion with this Data CPLD TLP. This TLP is downstream steered to the Endpoint, which in turn receives this TLP and terminates the transaction.

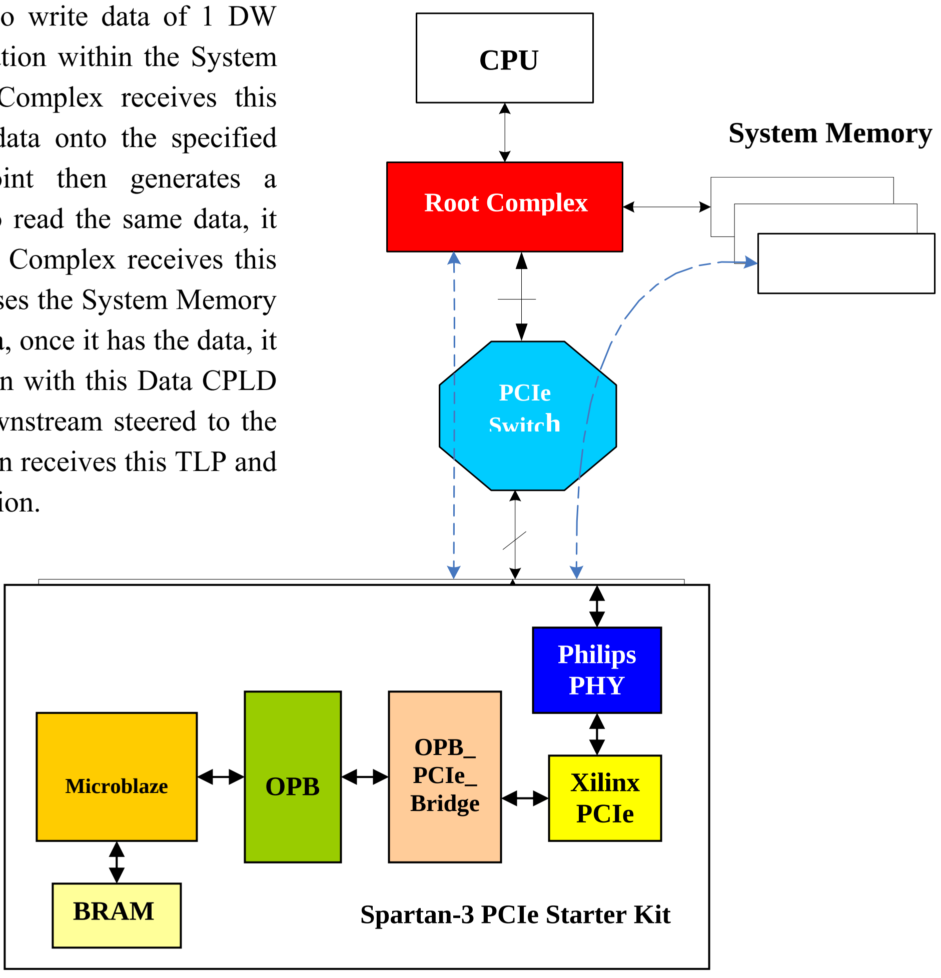


Figure 4.32 *- Test Cases 1 and 2*

Test case 3:

The purpose of this test case is to verify the ability of the Microblaze to read the PCIe configuration space. In this test case, the Microblaze reads one of the registers within this space.

Test case 4:

In this test case, the Microblaze tries to write to one of the PCIe configuration registers. In this case, the Microblaze fails to write because the version of the PCIe core used to implement the PCIe protocol layers does not allow such an access. This feature might be implemented in newer version of this core.

5 PCIe Endpoint Implementation

The designed PCIe Endpoint was synthesized using the Xilinx Integrated Software Environment (ISE). The different phases of the design implementation (Translation, Mapping, and Place & Route) were also performed.

The generation of a programming file to configure the targeted FPGA device (which is located on the Xilinx PCIe Spartan-3 Starter Kit) was not performed. Therefore, a board level functional verification was not carried out. The reason behind this was the unavailability of a windows software driver for the regarded kit.

Table 5.1 summarizes some of the resulted device utilization (FPGA family: Spartan-3, Target Device: xc3s1000, Target Package: fg676, and Target Speed: -4). These results are obtained from the generated Map and Place & Route reports. All the time requirements and constraints were met.

Table 5.1 - Device Utilization

Name	**Nr. out of total resources Nr.**	**Percentage (%)**	**Description**
BUFGMUXs	5 out of 8	62	Global clock buffer Multiplexers associated with the clock distribution tree.
DCMs	1 out of 4	25	Data Clock Managers
External IOBs	35 out of 391	8	Input/Output Blocks
LOCed IOBs	35 out of 35	100	Located Input/Output Blocks
MULT18X18s	3 out of 24	12	18X18 Multipliers
Block RAMs	16 out of 24	66	Block Random Access Memory
GCLKs	5 out of 8	62	Global Clocks
4 input LUTs	7668 out of 15360	49	4 input Look Up Tables
Slice Flip Flops	5796 out of 15360	37	Flip Flops
Slices	6209 out of 7680	80	Area utilization

6 Conclusion and Future Work

Within this diploma work, the various capabilities of the PCIe bus protocol were demonstrated. The theory of PCIe was summarized and presented in monthly-based presentations (PCIe tutorials). Some of the available PCIe IP solutions in the market were studied and compared.

In a platform based on PCIe topology, an *Endpoint* device was designed. This Endpoint embeds the *Microblaze* soft core of Xilinx, which is bridged to the PCIe protocol layers implemented by the *PCIe core*, to serve the data communication between this intelligent Endpoint and the CPU/system memory through the Root Complex.

The Xilinx Platform Studio (XPS), a part of the Xilinx Embedded Development Kit (EDK), was used to specify and design the Microblaze based system. A basic and simplified *OPB to PCIe Bridge* was developed to bridge the Microblaze and the PCIe protocol layers. The PCIe core was generated, configured and customized using the Xilinx CORE generator. A packaged simulation model, provided by NXP Semiconductors, was used to simulate the functionality of the PCIe physical layer. This model interfaces the simulation tool using the Verilog HDL Programming Language Interface (PLI).

In a modified version of a PCIe Testbench (provided by Xilinx) and with the help of the simulation tool ModelSim, the functionality of the designed Endpoint was simulated and verified.

In addition to that, the designed Endpoint was synthesized using the Xilinx Integrated Software Environment (ISE). The synthesized system was prepared to be implemented in the Xilinx Spartan-3 FPGA, located on the Xilinx PCIe Spartan-3 Starter Kit. The implementation itself was not carried out. The reason behind this was the unavailability of a windows software driver for this Kit. Therefore, the system functionality was not verified on the board level. This motivates the developing of a software driver to enable the future implementation (with the available results) and the board level verification of the designed PCIe Endpoint.

It can also be concluded that working with PCIe requires the knowledge of the PCIe protocol, because most of the available PCIe IP cores don’t provide a compatible interfaces, which allow them to be directly connected to the regarded processor. Therefore, in most cases, an effort must be made to develop a bridge that allows an easy connection of the PCIe peripheral to the processor.

Furthermore, the functionality of this designed Endpoint can be more complicated than this simple data transfer task. One can further extend the capabilities of this Endpoint by reconfiguring the PCIe core to include IO mapped space, as well as to allow some of the advanced features of this PCIe Endpoint.

Appendix A: PCI Express IP Providers

Many FPGA vendors and third party companies provide tested and optimized PCIe IPs as shown in figure A.1. Provided features for each layer of the PCIe structure differ from one provider to another.

Within this diploma work, some of the PCIe IPs available in the market were studied and compared.

***Figure A.1** - PCIe IP Providers*

The comparison was held in three different tables. In the first table, the various features of the physical and data link layers were compared. Features such as the number of the implemented Lanes, the line speed, the link Initialization and Training, the configuration of the physical layer (whether being built-in or external), the type of the interface to the PCIe IP, and the inclusion of a retry (replay) buffer and its size were considered in this table.

The second table compares the features implemented for the transaction layer. The features considered here are: the transmitter buffer and its size, the data bus width, the availability of virtual channel buffers, whether quality of services protocol is implemented or not, the data payload (Max. of bytes), flow control, TLP order rules, and the type of interface the core has when attached to a microprocessor.

In the third table, the general key features were compared. Features such as the functionality of the IP cores, the PCIe base specification version the cores meet, the targeted devices, the implemented features for data integrity, Compatibility with PCI-Special Interest Group (PCI-SIG), the power management features, and the configuration possibilities were considered in this table.

Table A.1 *- Features of the Physical Layer and Data Link Layer* [1]

IP Provider	Core Name	Nr. of Lanes	Line speed (Gbps)	Link initialization and Training	Built-in PHY	External PHY	PIPE Interface	Replay (Retry) Buffer
XILINX	Pci_exp_1_Lane_64b_ep	X1	2.5	Link width & Link data rate	1 MGTs[2]	NA	NA	NA
	Pci_exp_4_Lane_64b_ep	X4	10		4 MGTs			
	Pci_exp_8_Lane_64b_ep	X8	20		8 MGTs			
	Pci_exp_1_Lane_32b_ep	X1	2.5		1 MGTs			
	Pci_exp_4_Lane_32b_ep	X4	10		4 MGTs			
	PCI Express PIPE Endpoint 1-Lane	X1	2.5	Polarity Inversion	NA	Philips PX1011A	8-bit	
	PCI Express Endpoint Block	x1,x2, x4,x8	2.5,10,20	NA	Rocket IO GTP	NA	NA	1 up to 8 (32kbit RAM Block)
Perftrends	PCI-Express CTLR	x1, x2, x4,x8	2.5,10,20	Lane and polarity Reversal LTSSM	NA	Support for 8 or 16 bit PIPE interface for SerDes	8-bit or 16	Supported
Eureka Technology	PCI-Express Bus Controller EC310	x1,x4	2.5,10	Full LTSSM	Rambus and Xilinx PHY	Philips' PX1011A, PHY from Genesys Logic	Standard PIPE Interface	Supported
tallika CORPORATION	PCI Express Endpoint Core	x1,x2,x4 ,x8,x12, x16	2.5,5,10,20,30,40	Full LTSSM	NA	Any PIPE 1.0 compliant PHY.	8-bit or 16	Single

[1] NA stands for Not Available. The feature is either not available, or no information are provided
[2] MGTs: Multi-Gigabit Tranceivers

Table A.1 *- Features of the Physical Layer and Data Link Layer (Cont.)* [1]

IP Provider	Core Name	Nr. of Lanes	Line speed (Gbps)	Link initialization and Training	Built-in PHY	External PHY	PIPE Interface	Replay (Retry) Buffer
ASIC ARCHITECT	Endpoint	x1, x2, x4,x8 or x16	2.5,5,10,20 ,40	NA	NA	PIPE or Non-PIPE PHY logic	8-bit and 16-bit PIPE	Configurable
	Root port							
	Dual Mode(EP/RC)							
	Switch Port (Up-/Downstream)							
CAST	PCIe-EP	x1,x4	2.5,10	Link width, lane order, Lane Reversal and polarity Inversion	NA	Any 16-bit PIPE-compliant PHY	16-bit	NA
NORTHWEST LOGIC	PCI Express Core	x1,x4, x8	2.5,10,20	NA	Integrated PHY FPGAs	Discrete PHY (Genesys, Philips, TI) and PIPE-compliant ASIC PHY	Standard PIPE	NA
Rambus	GPEX-EP	x1, x2, x4,x8 or x16	2.5,5,10,20 ,40	Flexible Lane ordering and Lane Reversal	NA	PIPE spec v1.0 compliant	8-bit or 16-bit	Configurable

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.1 *- Features of the Physical Layer and Data Link Layer (Cont.)* [1]

IP Provider	Core Name	Nr. of Lanes	Line speed (Gbps)	Link initialization and Training	Built-in PHY	External PHY	PIPE Interface	Replay (Retry) Buffer
SYNOPSYS Predictable Success	PCI Express Endpoint Core PCI Express 2.0 Endpoint Core	x1, x2, x4,x8 or x16	2.5,5,10,20 ,40	Complete Link Training (LTSSM) Automatic Lane Reversal and polarity Inversion	NA	Rocket I/O for example.	8-bit or 16-bit	Configurable
ingot The Electronic Design Solutions Company	PCI Express IP core IP7001	x1, x2, x4, x8 future	2.5,5,10,20	NA	FPGA On-Chip PHY through wrapper	Intel compatible PIPE PCIe PHY	NA	NA
denali	Databahn™ PCI Express IP	x1, x2, x4, x8	2.5,5,10,20	NA	NA	Compliant with Intel PIPE Specification v1.86	8-bit or 16-bit PIPE	NA
GDA Technologies, Inc. accelerate your innovation	PCI Express End Point (GPEX-EP) PCI Express Root Complex GPEX-RC PCI Express Switch Controller (GPEX-SW)	x1, x2, x4, x8 or x16	2.5,5,10,20 ,40	Flexible lane ordering and support for lane reversal	NA	PIPE based PHY	PIPE based PHY	Efficient Flexible and configurable

[1] NA stands for Not Available. The feature is either not available, or no information are provided

[2] LTSSM: Link Training and Status State Machine

Table A.2 *- Features of the Transaction Layer* [1]

IP Provider	Core Name	Tx Buffers Width/bit	Data Bus Width	Virtual Channel Buffers	Quality of Services Protocol	Data Payload (Max. of bytes)	Flow Control	TLP Ordering Rules	Standard Bus Interface
XILINX®	Pci_exp_1_Lane_64b_ep	16 (5)	64	NA	NA	512	Receive and Transmit	Fully compliant	NA
	Pci_exp_4_Lane_64b_ep	16 (5)	64						
	Pci_exp_8_Lane_64b_ep	32 (6)	64						
	Pci_exp_1_Lane_32b_ep	8 (5)	32						
	Pci_exp_4_Lane_32b_ep	16 (5)	32						
	PCI Express PIPE Endpoint 1-Lane	6	32						
	PCI Express Endpoint Block	Min. 1 and max. 16 (36K-bit block RAM)	32	UP to 2 VCs	NA	From 128 to 4000			
Perftrends	PCI-Express CTLR	Configurable	32,64 or 128	Default TC0/VC0 VC capability	NA	NA	Receive and Transmit	Compliant	NA
Eureka Technology	PCI-Express Bus Controller EC310	NA	NA	NA	NA	NA	Receive and Transmit	Compliant	NA
tallika CORPORATION	PCI Express Endpoint Core	Flexible and configurable	64 or 128	Up to 8 VCs and 8 TCs	provided	NA	Flow control in both direction	Compliant	NA

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.2 *- Features of the Transaction Layer (Cont.)* [1]

IP Provider	Core Name	Tx Buffers Width/bit	Data Bus Width	Virtual Channel Buffers	Quality of Services Protocol	Data Payload (Max. of bytes)	Flow Control	TLP Ordering Rules	Standard Bus Interface
ASIC ARCHITECT	Endpoint	NA	32/64/ 128 bit	Up to 8 VC/8 TC	Provided	NA	NA	NA	NA
	Root port								
	Dual Mode(EP/RC)								
	Switch Port								
CAST	PCIe-EP	Configurable	64	Up to 8	NA	From 128 to 4000	Receive and Transmit	Supported	Wishbone AMBA
NORTHWEST LOGIC	PCI Express Core	Flexible sizing	NA	Multiple VCs	NA	NA	NA	NA	NA
ingot	PCI Express IP core IP7001	NA	NA	NA	NA	From 64 to 4000	NA	NA	NA

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.2 *- Features of the Transaction Layer (Cont.)* [1]

IP Provider	Core Name	Tx Buffers Width/bit	Data Bus Width	Virtual Channel Buffers	Quality of Services Protocol	Data Payload (Max. of bytes)	Flow Control	TLP Ordering Rules	Standard Bus Interface
Rambus	GPEX-EP	Configurable	32,64 or 128	Configurable up to 8	Provided	From 128 to 4000	Flow control logic for both directions	Compliant	NA
	GPEX-RC								
	GPEX-SW								
	GPEX-EP/RC								
denali	Databahn™ PCI Express IP	NA	NA	NA	NA	NA	NA	NA	NA
GDA Technologies, Inc. accelerate your innovation	PCI Express End Point (GPEX-EP)	NA	32,64 or 128	configurable	NA	From 64 to 4000	Flow control in both direction	Compliant	NA
	PCI Express Root Complex GPEX-RC			Configurable up to 8		From 128 to 4000	NA		
	PCI Express Switch Controller GPEX-SW			configurable			Flow control in both direction		

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.2 *- Features of the Transaction Layer (Cont.)* [1]

IP Provider	Core Name	Tx Buffers Width/bit	Data Bus Width	Virtual Channel Buffers	Quality of Services Protocol	Data Payload (Max. of bytes)	Flow Control	TLP Ordering Rules	Standard Bus Interface
SYNOPSYS® Predictable Success	PCI Express Endpoint Core	Configurable	32,64 or 128	Configurable up to 8 And up to 8 TCs	NA	From 128 to 4000	Flow control in both direction	Compliant	AMBA™ 2.0 AHB™ and AMBA™ 3 AXI™
	PCI Express 2.0 Endpoint Core	NA	NA	NA	NA	NA	NA		NA

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.3 *- General Key Features* [1]

IP Provider	Core Name	Function	PCIe Base Spec.	Targeted Device	Data integrity, Message and Interrupt	PCI-SIG	Power Management and Configuration
XILINX	Pci_exp_1_Lane_64b_ep	Endpoint	v1.1	Virtex-4, Virtex-II Pro	Error detection, recovery and Reporting	Compliant	PCI/PCIe power management
	Pci_exp_4_Lane_64b_ep			Virtex-4, Virtex-II Pro			
	Pci_exp_8_Lane_64b_ep			Virtex-4			
	Pci_exp_1_Lane_32b_ep			Virtex-4			
	Pci_exp_4_Lane_32b_ep			Virtex-4			
	PCI Express PIPE Endpoint 1-Lane			Spartan-3™ Spartan-3E			Active State Power management Programmed Power management
	PCI Express Endpoint Block	PCIe Endpoint block		Virtex™-5 LXT			Up to 6 x 32-bit or 3 x 64-bit BARs (or a combination of 32 bit and 64 bit) and BARs configurable for memory or I/O
Perftrends	PCI-Express CTLR	Endpoint or Root complex	v1.1	ASIC FPGA	MSI or Legacy Interrupt Message Optional End-to-end CRC (ECRC) Optional parity protection	Compliant	NA
Eureka Technology	PCI-Express Bus Controller EC310	Endpoint	v1.1	ASIC,FPGA: Virtex II Pro Virtex 4	LCRC, error checking	Compliant	all PCI Express configuration and power management registers

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.3 *- General Key Features (Cont.)* [1]

IP Provider	Core Name	Function	PCIe Base Spec.	Targeted Device	Data integrity, Message and Interrupt	PCI-SIG	Power Management and Configuration
tallika CORPORATION	PCI Express Endpoint Core	Endpoint configuration. extensible to support root complex and switch solutions	*v*1.1	Virtex II Pro and Virtex 4 and 0.18u or below	Optional ECRC Optional Advanced Error Reporting support Complete message support for INTx, MSI,PME	Compliant	Optional Power Budgeting capability support. Supports Active State Power Management (ASPM) and Software compatible PCI-PM.
ASIC ARCHITECT	Endpoint	Endpoint	*v*1.1 and *v*2.0, Rev 0.7	ASIC FPGA	Selectable ECRC and Advanced Error Reporting Support	Compliant	Supports all power management states L0,L0s,L1,L2 & L3 Supports Beacon and Wake-Up mechanism Configurable Type-0 (Endpoint) or Type-1 (Root Port, Switch Port) Config Headers
	Root port	Root Complex					
	Dual Mode(EP/RC)	Dual (Endpoint/Root)					
	Switch Port (Up-/Downstream)	Switch					
CAST	PCIe-EP	Endpoint	1.0a	ASIC(TSMC 0.13 and 0.18 µm), FPGA (Virtex-II Pro, Startix GX)	Advanced error reporting ECRC	Compliant	PCI configuration space type 0 header MSI

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.3 *- General Key Features (Cont.)* [1]

IP Provider	Core Name	Function	PCIe Base Spec.	Targeted Device	Data integrity, Message and Interrupt	PCI-SIG	Power Management and Configuration
NORTHWEST LOGIC	PCI Express Core	Endpoint	v1.1	ASIC and FPGA	Complete error-handling (detection and reporting)	Compliant	User expansion of Config. space
Rambus	GPEX-EP	Endpoint Controller	v1.1 and v1.0a	ASIC and FPGA	Baseline and advanced error and reporting	Compliant	All configuration capabilities Memory/IO /Expansion ROM BARs
	GPEX-RC	Root Complex Port controller					
	GPEX-SW	Switch Port Controller					
	GPEX-EP/RC	Dual Mode Controller					
SYNOPSYS Predictable Success	PCI Express Endpoint Core	Endpoint, Root Complex, Dual mode, Switch/Bridge	v1.1	ASIC (0.18 micron or below) and FPGA	Optional advanced PCI Express error reporting, Optional ECC for RAM, Configurable ECRC generation and checking All in-band messages supported for Endpoint, Legacy PCI, MSI, and MSI-X interrupt support	Compliant	Configurable EP filtering rules for posted, non-posted and completion traffic Configurable BAR filtering (up to 6), IO filtering, configuration filtering and completion lookup/timeout for EP Supports expansion ROM Type 0 configuration space
	PCI Express 2.0 Endpoint Core		v1.1 and v2.0	ASIC (90nm or below) and FPGA			

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Table A.3 *- General Key Features (Cont.)* [1]

IP Provider	Core Name	Function	PCIe Base Spec.	Targeted Device	Data integrity, Message and Interrupt	PCI-SIG	Power Management and Configuration
ingot The Electronic Design Solutions Company	PCI Express IP core IP7001	Endpoint	NA	NA	NA	Compliant	NA
denali	Databahn™ PCI Express IP	Root Complex, Endpoint, Dual Mode (RC/EP)	v1.1 and preliminary v2.0	ASIC or FPGA	Advanced Error Reporting	Compliant	AN
GDA Technologies, Inc. accelerate your innovation	PCI Express End Point (GPEX-EP)	Endpoint, bridge, switch, Root Complex	v1.0a	0.18u ASIC or better, FPGA	NA	Compliant	X
	PCI Express Root Complex GPEX-RC	NA			Message manager to map error messages to local events Efficient error management scheme		Hardware assisted power management scheme
	PCI Express Switch Controller GPEX-SW	NA			NA		ASPM L1 / Wake support, Auxiliary power support Supports Type1 configuration space Supports Type0/1 configuration conversions

[1] NA stands for Not Available. The feature is either not available, or no information are provided

Appendix B: Xilinx WebCases

WebCase 668804

Table B.1: *WebCase Summary*

Title:	Working with PCI Express PIPE v1.5	Case Type:	Technical Support
Owner:	Mark Noble	Severity:	No Rush
Contact:	Faraj Nassar	Condition:	Closed
Phone:	6504394756	Status:	Closed
Site ID:	249680	Service Pack:	sp2
Site Name:	Technical University Of Vienna	Device Family:	Spartan-3
		Software Version:	8.2i
		Os Type:	WinXP
Attachments:	There are no attachments for this case.		

The following topics were discussed in this WebCase:

- Simulation without the Philips PHY
- Configuration of the PCIe Core
- Simulation of the reference design
- Link Training and Initialization
- Xilinx training courses on how to design and develop the PCIe interfaces with the Xilinx Core

For more details, refer to the file xilinx_webcase_history.doc, located in the documentation sub-directory of the project's directory (C: /pcie_based_system/doc/).

Appendix C: Project Directory Structure

Figures C.1 shows the PCIe based System directory structure.

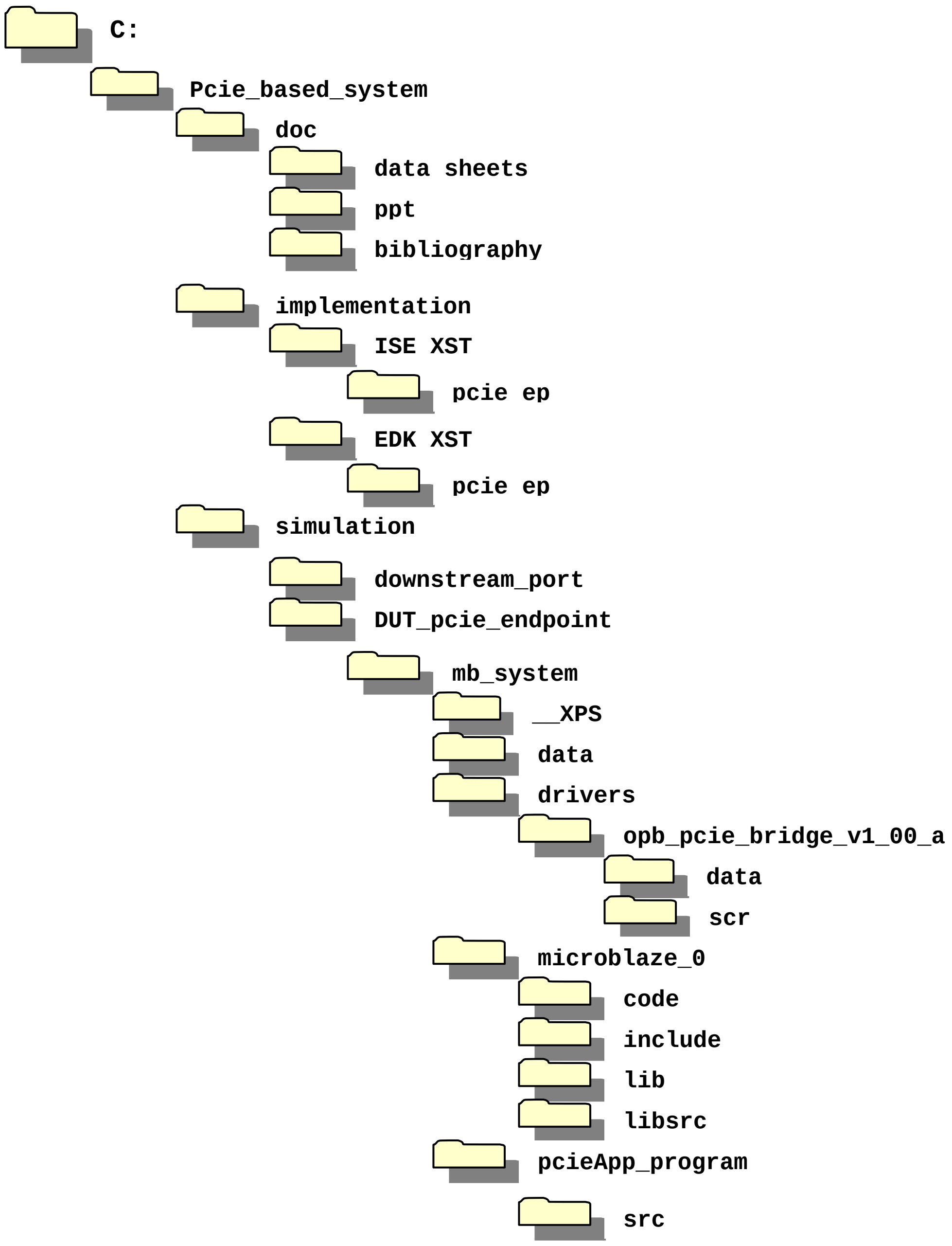


***Figure C.1** – Project Directory Structure*

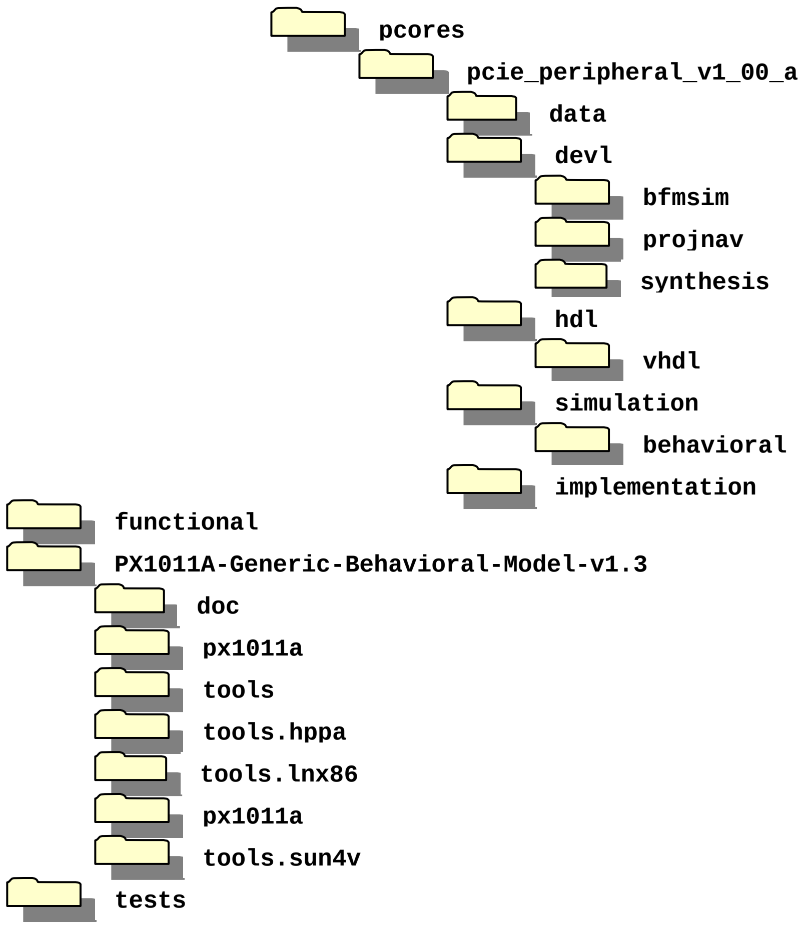


Figure C.1 *- Project Directory Structure (Cont.)*

Due to the license agreements with the companies Xilinx and NXP Semiconductors, the contents of the project directory, are kept for SIEMENS use only. Table C.1 lists the provided materials, to be found on a compact disc (CD) included with this thesis.

Table C.1*- Project Directory Structure*

Name	***Description***
`C:/pcie_based_system/`	**Main project's directory**
`C:/pcie_based_system/doc/`	**System Documentations**
`pcie_based_system.pdf`	PDF version of the Master's thesis.
`C:/…/…/data_sheets/`	Data sheets of PCIe IP solutions.
`C:/…/…/ppt/`	PCIe tutorial as PowerPoint presentations.
`C:/…/…/bibliography/`	References.

Bibliography

A

[AA05] **A Low-Cost PCI Express Solution**, Abhijit Athavale, Xcell Journal, Second Quarter 2005

[ABS04] **PCI Express System Architecture**, Don Anderson, Ravi Budruk and Tom Shanley, MINDSHARE, INC., 2004

[AS99] **PCI System Architecture**, Don Anderson and Tom Shanley, MINDSHARE, INC., 1999

B

[BTRL02] **Creating a PCI Express™ Interconnect**, Ajay V. Bhatt, Technology and Research Labs, Intel Corporation, 2002

C

[CUG03] **Cadence® IP Model Packager Guide for Model Users**, For Windows2000 and XP, Product Version 5.1w, September 2003

[GL05] **A Low-Cost Programmable PCI Express Solution**, David "Andrew" Brierley-Green and Ho Wai Wong-Lam, Xcell Journal, Third Quarter 2005

H

[HS04] **The Design of PCI Express for Future Communication Platform**, Eugin Hyun and Kwang-Su Seong, IEEE**,** 0-7803-8639-6/2004

[HS05] **Design and Verification for PCI Express Controller**, Eugin Hyun and Kwang-Su Seong, IEEE**,** 0-7695-2316-1/2005

J

[JW01] **Digital Design, Principles & Practices**, John F. Wakerly, Prentice Hall international, Inc., 2001

K

[KP03] **Advanced Switching Extends PCI Express**, Kiran S. Puranik, Xcell Journal, Fall 2003

[KPE05] **PXPIPE White Paper**, Application note, AN10372, Koninklijke Philips Electronics N.V, April 2006

[KPE06] **NXP x1 PHY single-lane transceiver PX1011A (I)**, Koninklijke Philips Electronics N.V, September 2006

L

[LB] **An Interface Methodology for Retargettable FPGA Peripherals**, Tien-Lung and Neil W. Bergmann, School of ITEE, The University of Queensland, Brisbane Australia

M

[MB99] "**Programmable Logic: What it to Ya?"**, Michael Barr, Embedded Systems Programming, pp. 75-84, June 1999

[MD06] **Debugging and Validating PCI Express I/O**, Richard Markley and Marco Davila, Agilent Technologies, I/O Magazine, January 2006

N

[NR06] **Lower System Cost with Spartan-3 based PCI Express Solutions,** Navneet Rao, Xilinx, Inc., September 26, 2006

P

[PM04] **Evaluating Xilinx MicroBlaze for Network SoC solutions**, *Master's Thesis in Computer Engineering*, Peter Magnusson, 10th January 2004

[PPHY] **Philips PCI Express PHY**, Philips

[PUG05] **PX1011A Behavioral Model User Guide**, PHILIPS, July 11th, 2005

[PXS02] **PCI Express Base Specification**, Revision 1.0, April 29, 2002

[PXS05] **PCI Express™ Base Specification**, Revision 1.1, March 28, 2005

S

[SD05] **Introduction to PCI Express –A New High Speed Serial Data Bus**, Satish K. Dhawan, IEEE, 0-7803-9221-3/2005

[SS06] **Achieve Performance Increases and Product Differentiation with OPB Mastering**, Steven M. Spano, Embedded Magazine, November 2006

U

[UG197] **Virtex-5 Integrated Endpoint Block for PCI Express Designs**, User Guide UG197 (v1.1) March 20, 2007

[UG2565] **Spartan-3 PCI Express Starter Kit Board**, User Guide v1.2, UG2565 July 21, 2006

X

[XAPP473] **Using the ISE Design Tools for Spartan-3 Generations FPGAs**, Xilinx Application Note, XAPP473 (v1.1) May 23, 2005

[XAPP516] **Bus Functional Model (BFM) Simulation of Processor Intellectual Property**, Lester Sanders, XAPP516 (v1.0) May 25, 2006

[XDS321] **PCI Express PIPE Endpoint 1-Lane Core v1.5**, Product Specification, DS321 September 21, 2006

[XDS401] **On-Chip Peripheral Bus V2.0 with OPB Arbiter (v1.10c),** Xilinx LogiCORE, DS401, Product Specification, August 31, 2006

[XDS414] **OPB IPIF Architecture**, Xilinx LogiCORE, DS414 (v1.3), Product Specification, January 13, 2003

[XDS444] **Block RAM (BRAM) Block (v1.00a),** Xilinx LogiCORE, DS444, Product Specification, August 21, 2006

[XDS445] **Local Memory Bus (LMB) V1.0 (v1.00a),** Xilinx LogiCORE, DS445, Product Specification, February 22, 2006

[XDS452] **LMB BRAM Interface Controller (v1.00b),** Xilinx LogiCORE, DS452, Product Specification, February 22, 2006

[Xilinx] Xilinx, www.xilinx.com

[XP05] **Low Cost Programmable PCI Express Solution**, Xilinx and Philips Semiconductors September 27, 2005

[XTU] **Custom Peripheral Design Guide**, Xilinx Tutorial

[XTU02] **Designing Custom OPB Slave Peripherals for MicroBlaze**, Xilinx Tutorial, February 8, 2002

[XTU06] **EDK 8.2 MicroBlaze Tutorial in Spartan 3**, WT001 (v4.0) August 30, 2006

[XUG05] **Processor IP Reference Guide**, Xilinx, February 2005

[XUG167] **LogiCORE™ PCI Express PIPE Endpoint 1-Lane v1.5**, User Guide, UG167 September 21, 2006

[XUG341] **LogiCORE™ PCI Express® Endpoint Block Plus v1.2,** Xilinx User Guide UG341 February 15, 2007

[XUG081] **MicroBlaze Processor Reference Guide**, Embedded Development Kit (EDK 8.2i), UG081 (v6.3), August 29, 2006