

B. TECH. PROJECT REPORT

On

Verification of PCIe and UART Protocol Features

BY
Pitchika Dinesh



DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
Dec 2019

Verification of PCIe and UART Protocol Features

A PROJECT REPORT

*Submitted in partial fulfillment of the
requirements for the award of the degrees
of*
BACHELOR OF TECHNOLOGY
In
ELECTRICAL ENGINEERING

Submitted by:
Pitchika Dinesh

Guided by:
Dr. Saptarshi Ghosh
Assistant Professor
Discipline of Electrical Engineering



INDIAN INSTITUTE OF TECHNOLOGY INDORE
Dec 2019

CANDIDATE’S DECLARATION

We hereby declare that the project entitled “**Verification of PCIe and UART Protocol features**” submitted in partial fulfillment for the award of the degree of Bachelor of Technology in ‘Electrical Engineering’ completed under the supervision of **Dr. Saptarshi Ghosh, Assistant Professor, Discipline of Electrical Engineering, IIT Indore** is an authentic work.

Further, I/we declare that I/we have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student(s) with date

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Signature of BTP Guide(s) with dates and their designation

Table of Contents

Preface	V
Acknowledgements	VI
Abstract	VII
List of Figures	VIII
Chapter 1 - Introduction	1
1.1 Problem Statement.....	3
1.2 Motivation to Work	3
Chapter 2 - Background	5
2.1 Verilog.....	5
2.2 Systemverilog.....	6
2.3 Universal Verification Methodology(UVM).....	7
Chapter 3 - Protocols	11
3.1 Universal Asynchronous Receiver Transmitter(UART) Protocol.....	11
3.2 Peripheral Component Interconnect Express(PCIe) Protocol.....	13
Chapter 4 - My Contribution	21
4.1 Work done in Verilog.....	21
4.2 Work done in Systemverilog.....	25

4.3 Work done in UVM.....	27
4.4 Work done in PCI Express.....	28
4.5 Work done in UART.....	29
Chapter 5 - Conclusion	33
5.1 Future Works.....	33
References	35

Preface

This report on “**Verification of PCIe and UART protocol features**” is prepared under the guidance of **Dr. Saptarshi Ghosh, Assistant Professor, Discipline of Electrical Engineering, IIT Indore.**

In this report I am going to describe my experiences during my internship at Mentor Graphics Pvt Ltd. The report contains an overview of the activities and projects that I have done during my internship. Writing this report, I will also describe and reflect upon my learning's and objectives that I have set during the course of my internship period. I have tried to discover the relationship between theoretical and practical type of knowledge and to bridge the gap between theoretical assumptions and practical necessities. I made all possible efforts and collected necessary information to submit this paper in an enlightened form in a very short time. I have tried my level best to eliminate errors from the paper.

Pitchika Dinesh

B.Tech. IV Year

Discipline of Electrical Engineering

IIT Indore

Acknowledgements

This report has been prepared for the internship that has been done at Mentor Graphics Private Ltd. in order to study the practical aspect of the course and implementation of the theory in the real field with the purpose of fulfilling the requirements of the course of Electrical Engineering.

I would like to express my sincere gratitude to my supervisor Dr. Saptarshi Ghosh for encouraging me and allowing me to go for the internship and learn something throughout the course of the internship. I would also like to thank him for the suggestions provided which helped in completion and documentation of the internship.

Also I would like to thank Mr. Amit Gupta for providing me an opportunity to work at Mentor Graphics Pvt Ltd. Also I would like to express my gratitude to Mr. Pankaj Kumar Tripathi and the entire VTLQA team at Mentor Graphics Pvt Ltd. for encouraging me and for adding value into my work.

Pitchika Dinesh

B.Tech. IV Year

Discipline of Electrical Engineering

IIT Indore

Abstract

Verification of a Digital Design is one of the important aspects of ASIC/SoC Design Flow. To ensure that the end user of the product does not face any issues, 80% of the time required for a design specification to take shape into a product is consumed in Verification phase. In Verification the design undergoes testing under different scenarios and test cases including testing it in erroneous condition to cover all possible scenarios. With PCI Express and UART becoming one of the most widely used in SoC designs, verification of their features should be fast and efficient. With newer features being added each day verification has to be up to date. The more complex features the more complex task it is to verify those features. The work done here is a contribution to verify some of those features and built a robust testbench code without any errors, so that it could drive the design through all possible scenarios and check for bugs in the design if there are any. The Testbench environment is also designed in such a way that it is robust and can be reused with minimal modifications so that more and more test cases could be added for verifying any features expected in later versions.

List of Figures

Figure 1: UART Transmitter and Receiver.....	1
Figure 2: Data transmitted b/w UART devices.....	2
Figure 3: Overview of PCI Express.....	2
Figure 4: Typical Systemverilog TestBench.....	6
Figure 5: uvm base classes hierarchy.....	7
Figure 6: UVM Testbench Block Diagram.....	8
Figure 7: Phasing in UVM.....	9
Figure 8: Block diagram of UART.....	11
Figure 9: Data Format in UART.....	12
Figure 10: PCIe hierarchy.....	13
Figure 11. Link and lane.....	14
Figure 12: Layers in PCIe devices.....	15
Figure 13: Bus Enumeration.....	16
Figure 14: Phases of Link Training.....	17
Figure 15: General TLP format.....	19
Figure 16: Memory Model.....	25
Figure 17: Testbench for Memory Model.....	25
Figure 18: Full Adder Model.....	27
Figure 19: Testbench for Full Adder Model.....	27

Chapter 1

Introduction

A system on chip (SoC) is an integrated circuit that contains almost all components of an electronic system like a computer etc.. The components normally present are, a central processing unit, a memory unit, input/output ports and a secondary storage etc, all on a single substrate or microchip, the size of a coin. Each individual component of a SoC is called an Intellectual Property (IP). IP's are reusable digital logic circuits which can be used in other SoCs as well.

In an SoC there are buses which connect different IP's so that data could be shared among them. In most of the cases, the IP cores are designed with many interfaces and communication among them can be a problem while integrating into an SoC. To solve these types of problems, standard on-chip bus structures and protocols were developed.

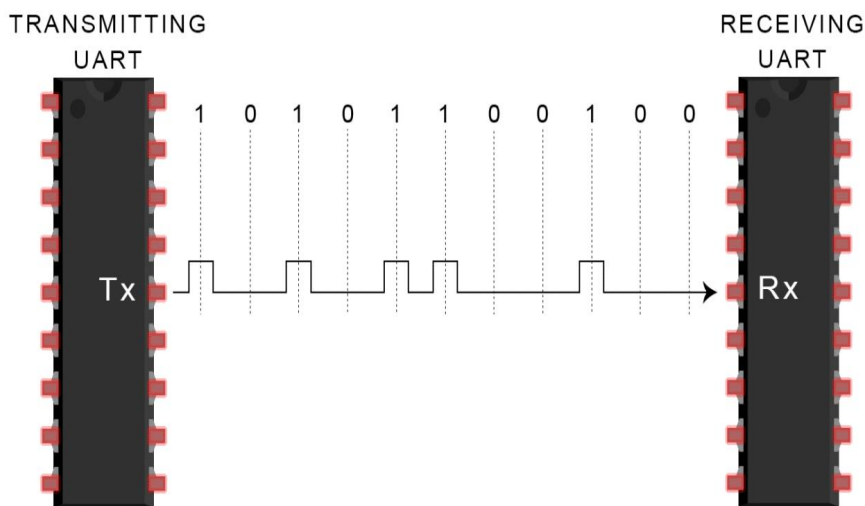


Figure 1: UART Transmitter and Receiver

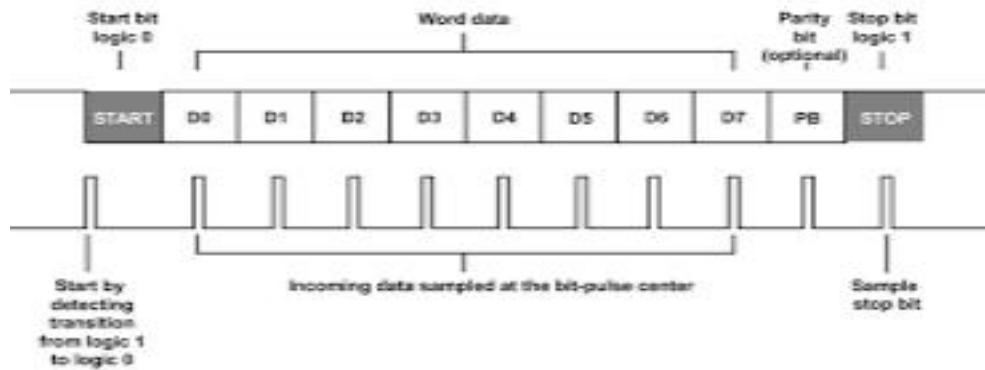


Figure 2: Data transmitted b/w UART devices

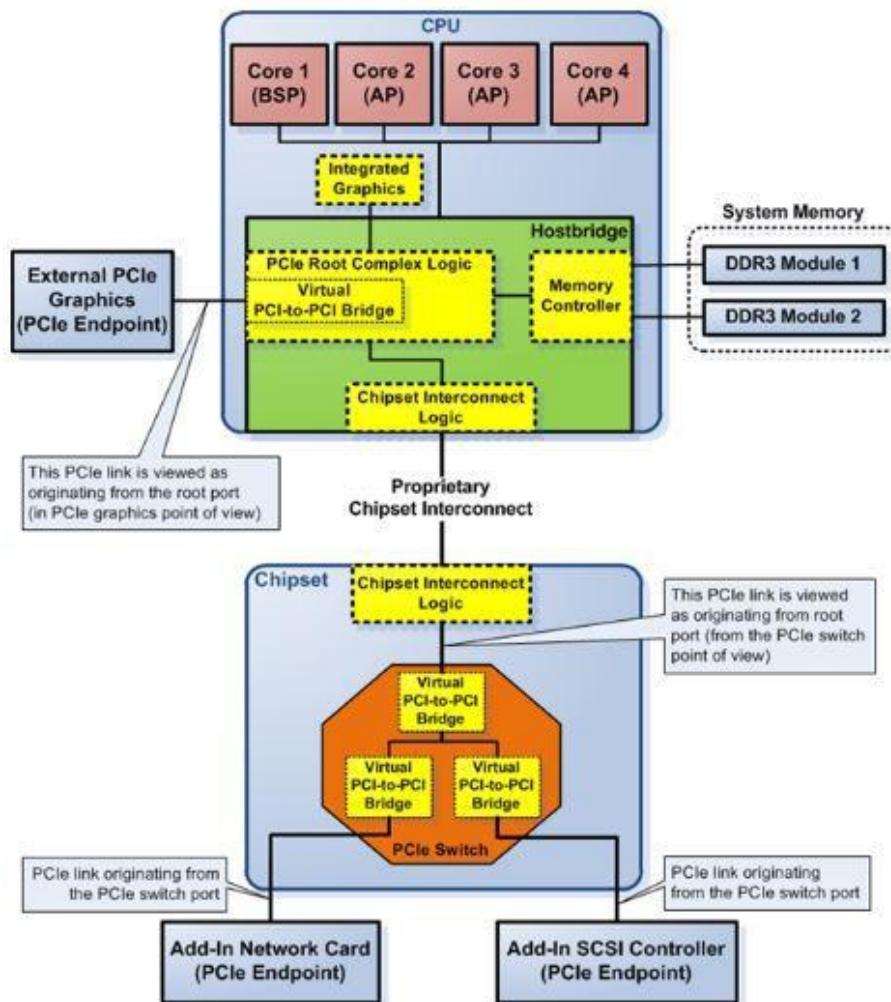


Figure 3: Overview of PCI Express

1.1 Problem Statement

Main Task as a part of this internship is to verify different features provided for Root Complex(RC) and EndPoint(EP) which are devices part of PCIe protocol and to verify xterm enable and pure receive mode feature provided for UART Receiver and Transmitter devices.

1.2 Motivation to Work:

Universal Asynchronous Receiver Transmitter(UART) and Peripheral Component Interconnect Express(PCIe) are one of those protocols used for communication among IP cores. Performance of the chip(SoC) is highly dependent on how effectively data is transferred between IP cores through these protocols. Each protocol whether it's PCIe or UART have some features which ensure errorless and effective transmission among IP cores. So, 'Verification' of these features provided by PCIe and UART becomes very important as performance of the SoC is highly dependent on them. And as new features get added to these protocols(especially PCIe) new verification methodologies and test plans have to be prepared to verify its behaviour in all possible scenarios, more complex the features are more complex it is to verify those features.

Chapter 2

Background

2.1 Verilog:

Verilog is a Hardware description language generally used for designing different combinational and sequential digital circuits like logic gates(AND,OR etc), Full Adders, Multiplexers, Demultiplexers, Encoders, Decoders, FlipFlops, Registers, Counters, Memories etc. All components stated above form the basis of designing IP cores which are designed using Verilog.

Verilog can also be used for verification of digital design to check whether the logic is right or not and for simulation and timing analysis of a digital logic design. In Verilog, the digital circuit design is described in terms of modules. Each module has inputs and outputs. To verify the functionality of a module, we create another module which is called 'TestBench' in which we instantiate the module whose functionality is to be verified. A testbench does not have any inputs and outputs, it's only purpose is to drive the inputs to the module which is being tested and collect the response from the outputs. QuestaSim Software was used to simulate the testbench module and to visualize the input and output waveforms. Using these waveforms we can verify the functionality of the module. But using Verilog, we can only make primitive testbenches which can only drive only known inputs values. But for complete verification of a Design we should be able to write a testbench which can drive almost all types to inputs to the design which it can encounter in real time usage.

2.2 Systemverilog:

So, to build a robust testbench which can drive all kinds of inputs into the design and check the output for each and every input, we use an improvised version of Verilog called SystemVerilog.

The key features of Systemverilog are Object Oriented Programming in which we can send inputs in terms of transactions or packet and receive output in terms of packets. Systemverilog also provides a constraint block which helps use to drive random input values into the design under some constraints in a much easier manner compared to verilog. It also has a feature called Functional Coverage using which we can track how many possible scenarios have been covered while testing the functionality of the design.

A Typical Systemverilog testbench has the following components: Generator, Driver, Monitor, Scoreboard, Interface, Design Under Test(DUT).

‘DUT’ is the Device(Module) Under Test whose functionality is being tested. Signals are exchanged between the Test ‘Environment’ and DUT in terms of ‘transactions’. ‘Generator’ generates those transactions and ‘Driver’ drives them towards the DUT through the ‘Interface’ and ‘Monitor’ collects the response of DUT through the interface and sends them to the ‘Scoreboard’ which checks whether the output is correct or not.

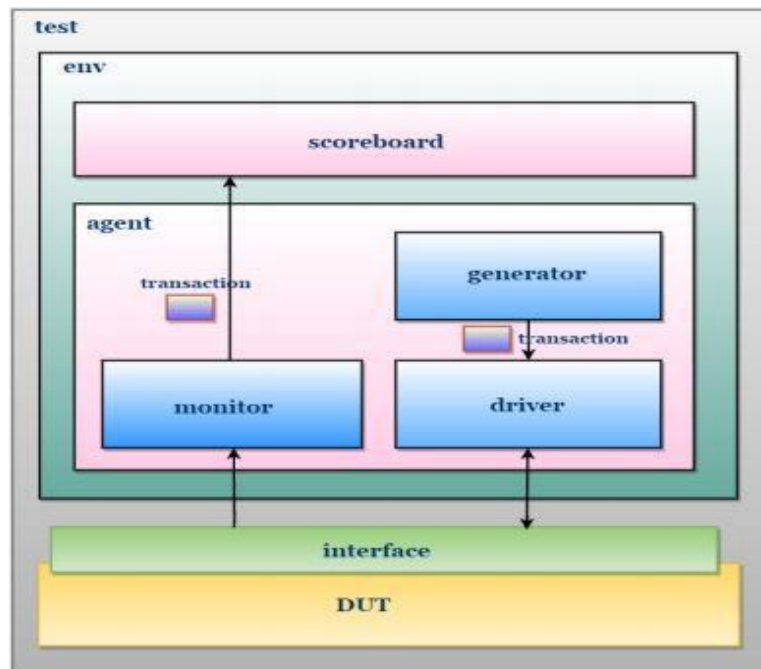


Figure 4: Typical Systemverilog TestBench

2.3 Universal Verification Methodology(UVM):

The Universal Verification Methodology (UVM) is a framework built using Systemverilog which has classes and libraries required for developing a well constructed, reusable Systemverilog based Verification Testbench environment. In simple words, UVM has a set of base classes for each component of a Testbench with methods defined in it, code for writing each and every testbench can be done easily by extending those base classes and simulation can be easily done by the methods provided within each base class. Some Base classes in UVM are: uvm_object, uvm_component, uvm_sequence_item, uvm_sequence, uvm_sequencer, uvm_driver, uvm_monitor, uvm_scoreboard, uvm_agent, uvm_environment, uvm_test.

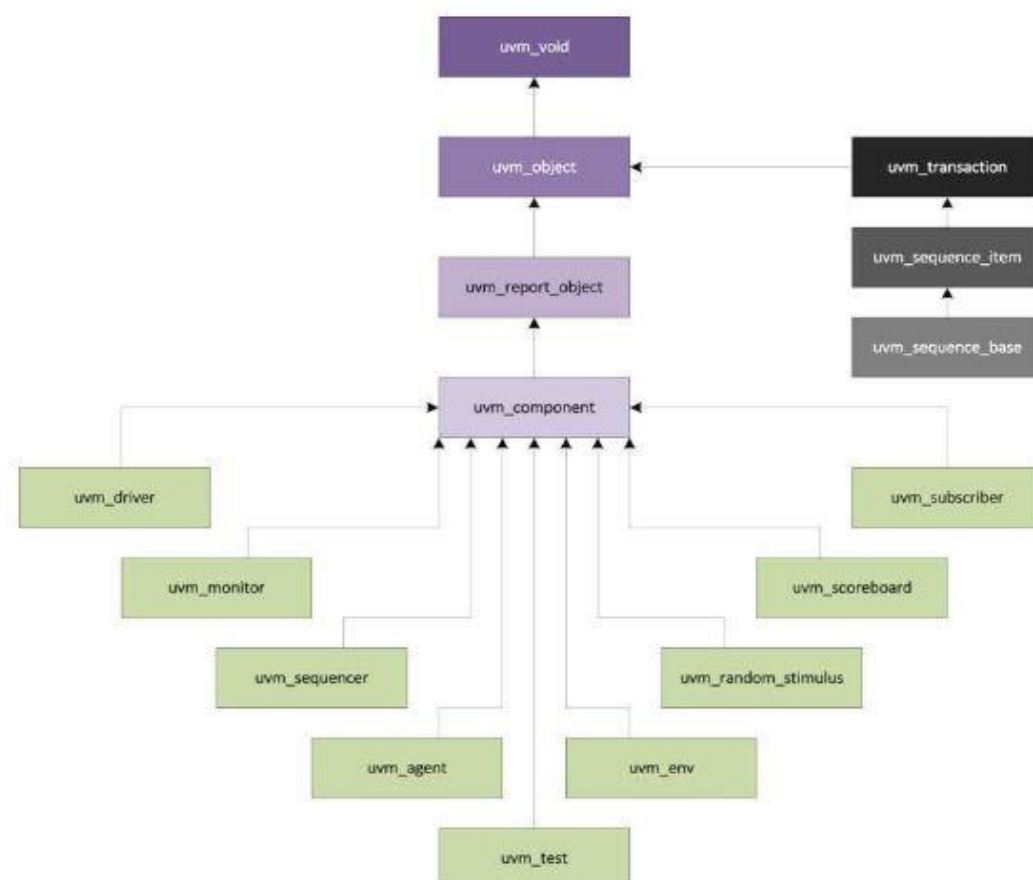


Figure 5: uvm base classes hierarchy

Uvm_sequence_item and uvm_sequence come under uvm_object. Object of these types are dynamic in nature in the sense that they can be created and sent at any time during the simulation. Uvm_sequencer, uvm_driver, uvm_monitor, uvm_scoreboard, uvm_agent, uvm_environment, uvm_test extend from uvm_component. Objects of this kind are static in the sense that multiple number of uvm_components are not created during simulation.

In the above mentioned uvm_components, uvm_agent is a combination of sequencer, driver and monitor. Uvm_agent and uvm_scoreboard are connected using analysis ports. Agent and scoreboard combined together is called uvm_environment. Environment and Sequence are together called uvm_test.

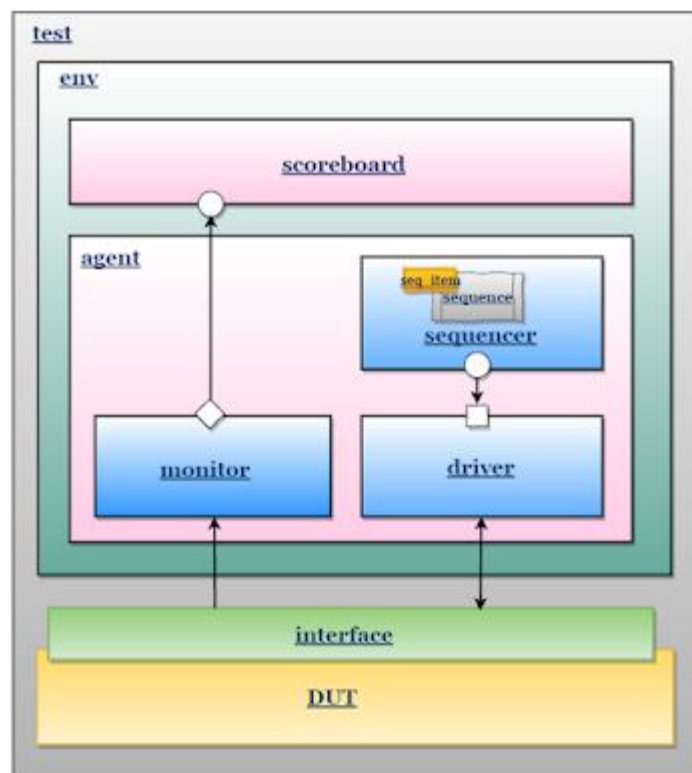


Figure 6: UVM Testbench Block Diagram

Each uvm_component has inbuilt methods called phases. When simulation starts, each component goes through a set of phases in a prescribed order. These phases act as a synchronization mechanism in the life cycle of simulation.

There are three types of phases:

1. Build phases
2. Run phases
3. Cleanup phases

When simulation of a test case starts build phases of each and every uvm_component is called in a top-down manner(from uvm_test to uvm_sequencer) where each and every uvm_components are created. After that, connect phase of each and every uvm_component is called in a bottom-up manner(from uvm_sequencer to uvm_test) where components are connected(Ex: driver and sequencer, monitor and scoreboard). After connect phase, Run phase of each and every component are called in parallel and this where the actual testing, i.e send inputs to DUT and receiving outputs from DUT takes place. The inputs and outputs are sent and received in the form of uvm_objects. After Run phases, clean up phases of each and every uvm_component is called which are used to collect summary of the test and the results of the simulation.

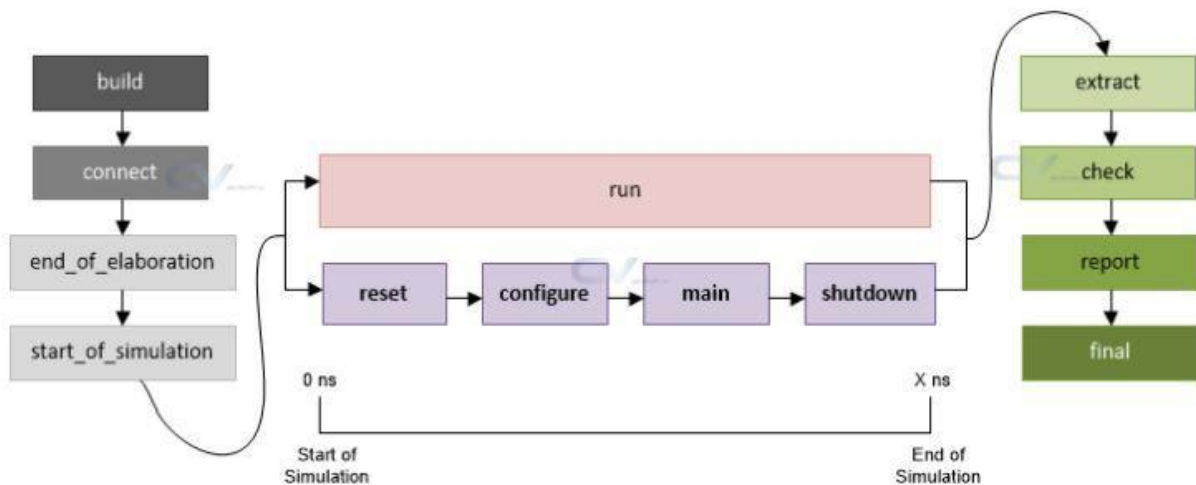


Figure 7: Phasing in UVM

Verilog, SystemVerilog and UVM are the basic requirements used for verifying functionality of a digital logic circuit(IP core) during simulation.

Chapter 3

Protocols

3.1 UART protocol

The **UART** stands for is “Universal Asynchronous Receiver/Transmitter”, and it is a serial communication protocol used to send data between two electronics devices. In UART communication takes place asynchronously, i.e. we do not have any clock signal which both devices can use to synchronize like in SPI and I2C protocols.

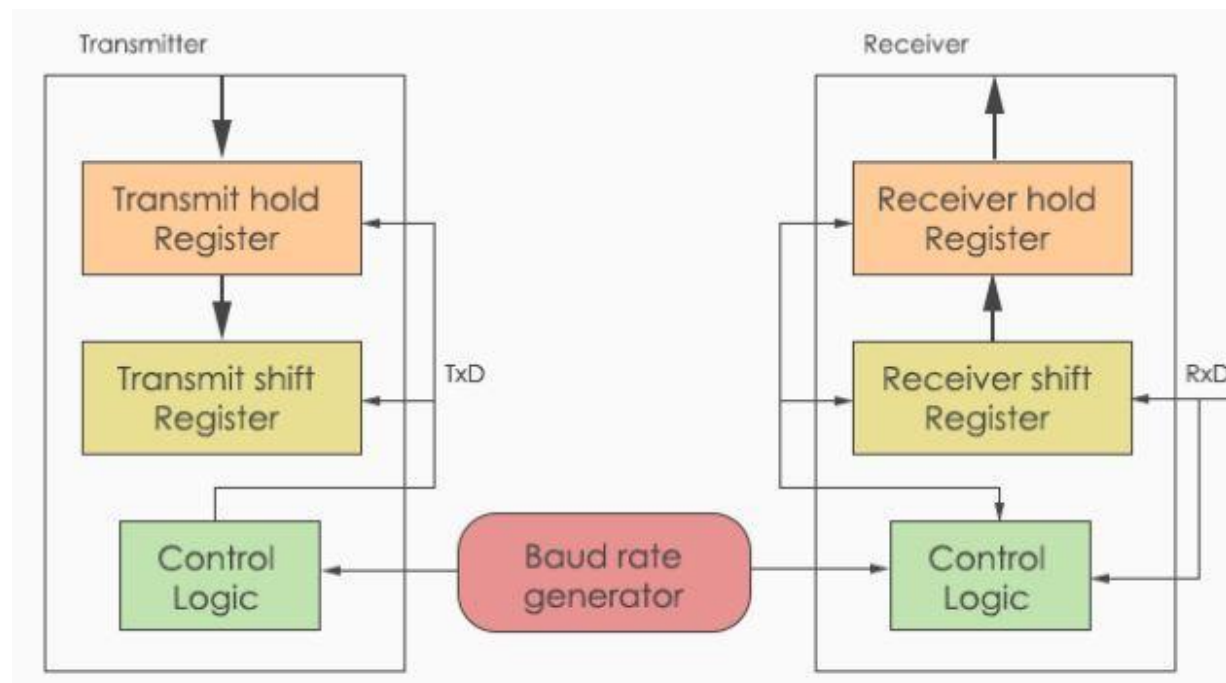


Figure 8: Block diagram of UART

Working of UART:

The Device or peripheral which wants to send data to some other device is connected to a Transmitting UART and the device which receives data is connected to a receiving UART. The Transmitting device sends whatever data it wants to the other device to the Transmitting UART through a Data Bus in parallel. The Transmitting UART adds a start bit, stop bit, and a parity bit to the data and sends them to the receiving UART in serial form. The Receiving UART receives the data in serial form, removes the start bit, stop bit and parity bit and sends data to the devices or peripheral to which it is connected in parallel form.

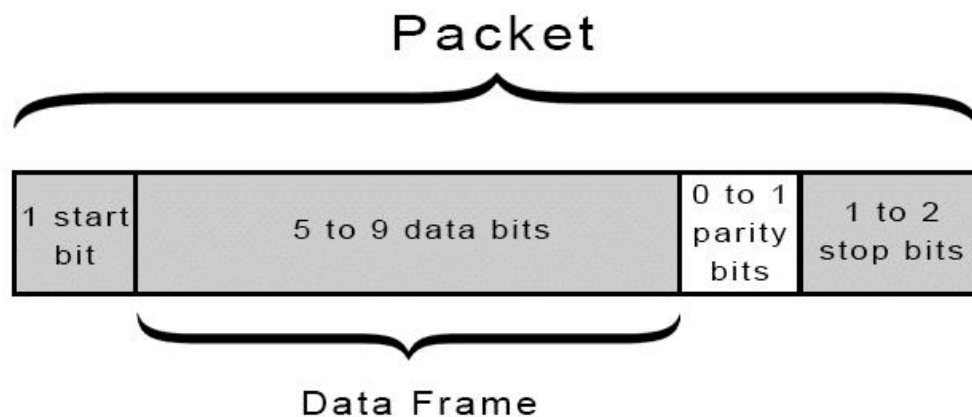


Figure 9: Data Format in UART

Since UART does not have a clock signal, it should have some sort of mechanism to transfer data synchronously. Start bit and Stop bit provide that synchronism by indicating the start and end to the data packet.

Start Bit:

When UART is not transmitting any data its Tx pin is held at a higher voltage level. So, to transmit data one has to bring that voltage to a lower level. When it is done the receiving UART detects that change and whatever is sent after this is considered as actual data. Generally only start bit is used.

Stop Bit:

After the actual data is transmitted the voltage at the Tx pin of transmitting UART is changed from low to high for at least two bit durations to indicate to the end of data to the receiving UART.

Data Frame:

It consists of actual data which is to be transmitted serially, the start bit, the stop bit and parity bit. Usually data frame is 5 to 8 bits long. If not parity bit is used data can be 9 bits long. The parity bit is used by the receiving UART to check whether data which is received is correct or not.

3.2 PCIe Protocol

PCIe(Peripheral Component Interconnect Express) is a standard bus protocol used to connect different peripherals devices in a PC like keyboard, mouse, Network Card, Ethernet etc to the CPU. It's a serial bus protocol where data is transmitted serially bit by bit between devices. It's a packet based protocol where instead of individual signals, all signals' data is packed and sent as a single entity. PCI Express is also backward compatible with its previous version like PCI-X.

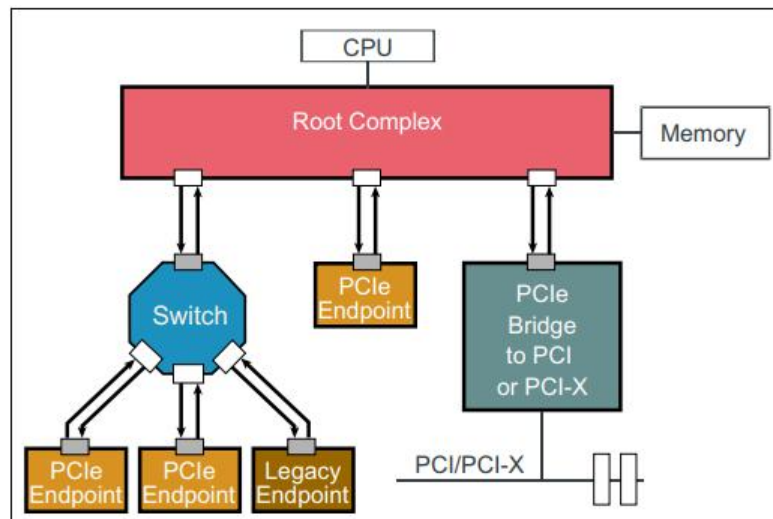


Figure 10: PCIe hierarchy

There are three main types of devices present in PCIe hierarchy:

1. **Root Complex(RC)** : The Root Complex is like the mind of PCI Express protocol. It monitors all the data transactions which are being sent to the CPU. It also configures the PCIe bus hierarchy when system is turned on. Now a days the CPU and Root Complex are present in the same chipset. The Root Complex has a downstream port through it connects other switches and end point devices.
2. **Switch**: A PCIe switch is a device that spawns buses which help in connecting more and more peripheral devices to the PCIe bus hierarchy. A Switch has both upstream and downstream through upstream ports it connects to all buses above its level all the upto Root Complex. Through downstream ports it connects to all buses below its level all the upto the end point devices.
3. **Endpoint Device(EP)**: This is the actual place which is connected to peripheral devices like Network card, Ethernet card etc. Endpoint device has only one upstream port through which it connect to buses above its level all the way upto the Root Complex

Each PCIe devices are connected to each through a set of wires called lanes. A link is a collection of such lanes.

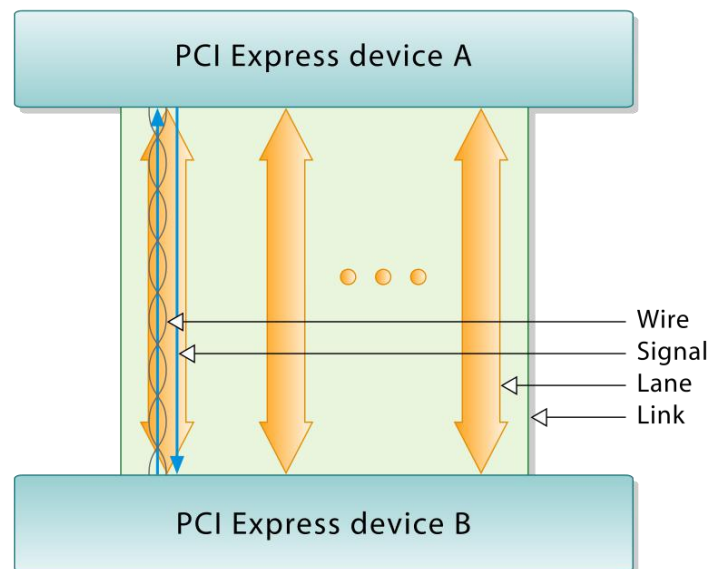


Figure 11. Link and lane

Each PCIe device whether it is a Root Complex, Switch or an endpoint device has three layers through data gets transferred. This is to ensure errorless and effective data transmission.

Three layers are:

- **Transaction Layer:** Generates packets in response to requests from Device core layer.
- **Data Link Layer:** Responsible for link management.
- **Physical Layer:** Physical layer is the where layer where data exchanged in the form of electrical signals.

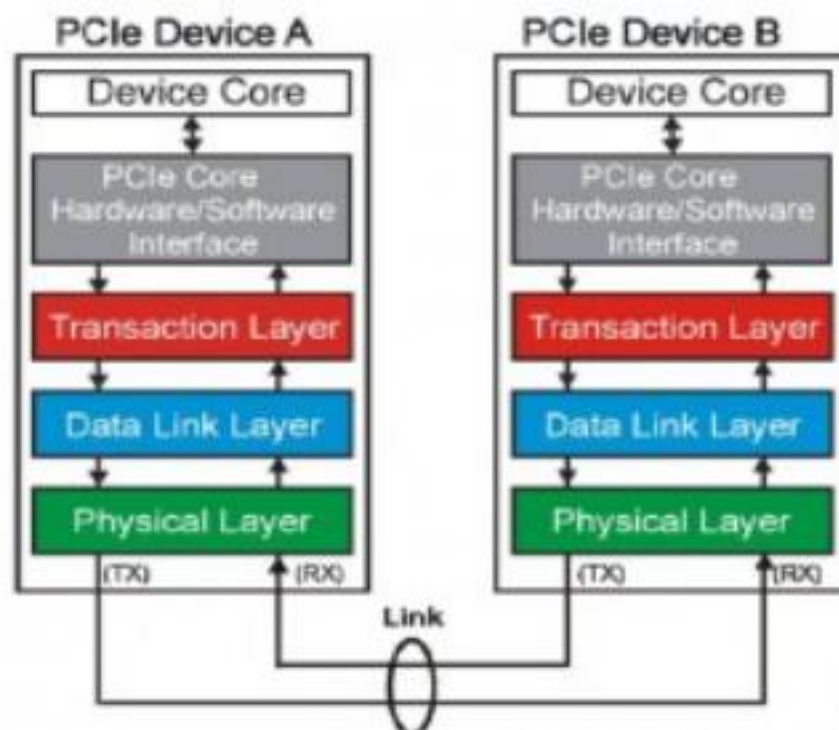


Figure 12: Layers in PCIe devices

Important Terms in PCIe:

1. **Configuration Space:** Each PCIe Device has a 4KB Configuration space which controls the behavior of that device. There are three types of registers in configuration space.
 - i. Standardized Header (Type 0/ Type 1)
 - ii. Capability Registers
 - iii. Extended Capability Registers

Every data packet or transaction starts at Root Complex travels through switches and reaches at Endpoint. Since Root Complex and Switches route the data packet towards the Endpoint they have Type 1 Header in their configuration space and Endpoint devices have Type

0 Header in their configuration space. Generally Type 1 header has information regarding routing of data and type 0 headers does not have routing information.

2.Bus Enumeration: Each device in PCIe hierarchy is assigned a 16 bit number called Bus/Device/Function(BDF) number which indicates the bus number in which that device resides and the device and function number also. The numbering starts from top left continues to right and then goes to bottom and so on.

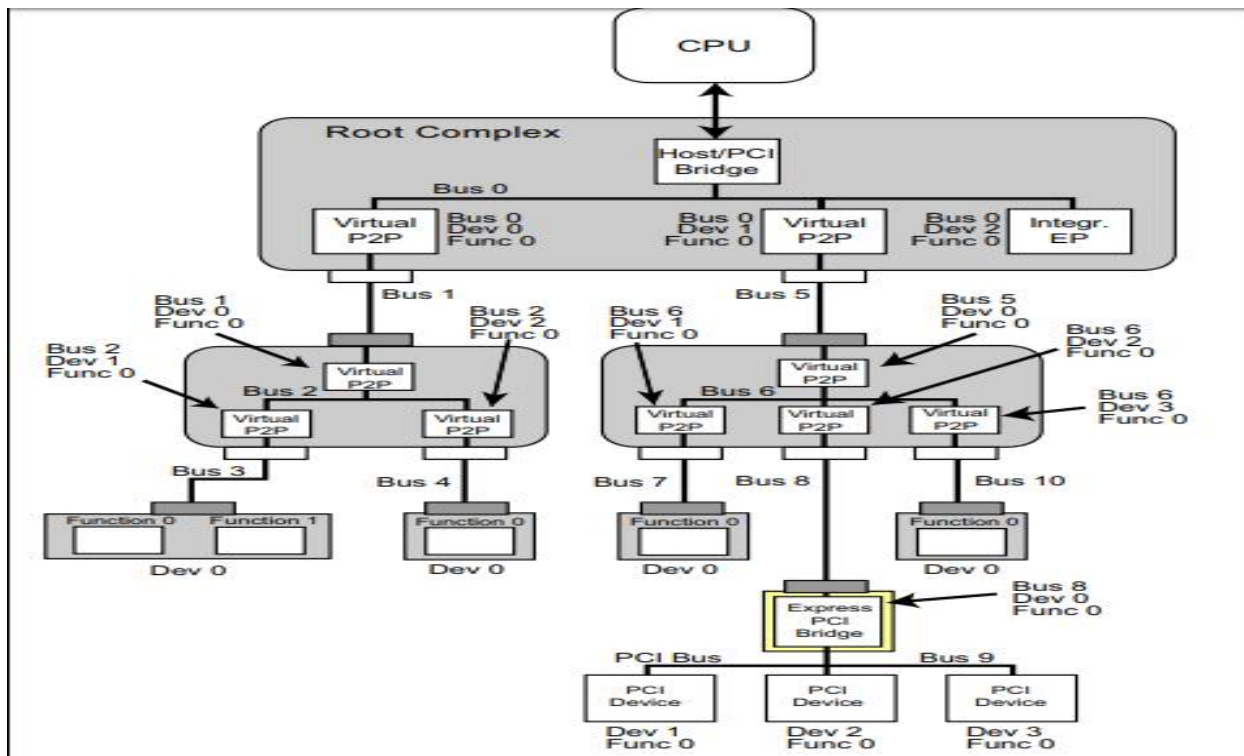


Figure 13: Bus Enumeration

3.Address Spaces(Memory/I/O) :

Each location in any device's memory has an address which indicates the location so that CPU could access it. A set of such addresses which can be accessed by CPU is called Memory Space. Here memory can be RAM, ROM or any memory which can be addressed by the CPU. An I/O **address** is a unique number assigned to a particular I/O device, used for **addressing** that device.

Important Registers in Configuration Space:

- **Primary/Secondary/Subordinate Registers:** (only in TYPE 1) Specifies buses present under that device.
- **Base/limit Registers:** (only in TYPE 1) Range of Addresses of devices which are present under that device.
- **Base Address Registers:** (both in TYPE 0/1) device's memory to CPU Memory or I/O space.
- **Capability Pointer:** (both in TYPE 0/1) Contains position of first Capability Register.

Physical Layer(PL) in PCIe:

This is the layer where data is sent in the form of electrical signals from Device A to Device B through Lanes. The PL is divided into two layers PHY and MAC. MAC layer receives data from Data Link Layer(DLL) and send them to PHY layer in parallel. PHY layer serializes them and sends them over the link(PL - PL). Whenever a new device is connected in the PCIe hierarchy the link which connects those two devices undergoes training which is called link training. Link Training ensures errorless transmission of data over the link.

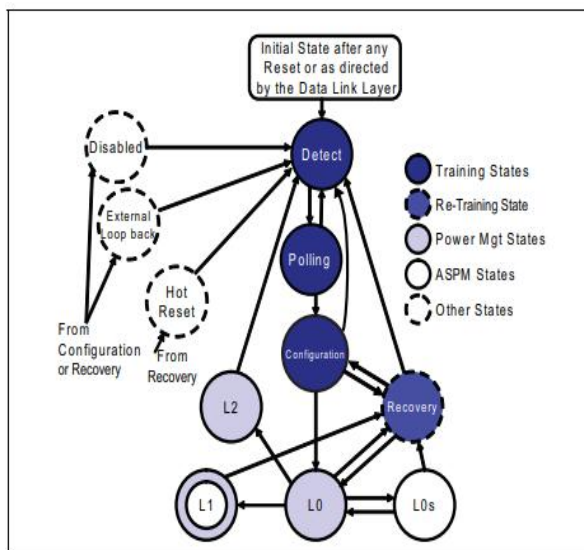


Figure 14: Phases of Link Training

Data Link Layer(DLL) in PCIe:

Data Link Layer receives data from the Transaction layer(TL). It adds a sequence number and LCRC to the data received from the Transaction layer so that on the receiver side it could whether data is received in a particular sequence in which it is sent and it also checks LCRC to check whether received is correct or not.

On the Receiver side if the data link layer receives data in a sequence and if it also satisfies LCRC check, the data link layer send an ACK DLLP to the transmitter to inform that the data has been rightly received and it can send data further. If there is any error in sequence number or LCRC check then Data Link Layer sends a NACK DLLP to the transmitter to inform that there has been an error in the data sent. On the Transmitter side if the Data Link Layer receives a NACK then it resends the data packets so that this time the packets are received correctly by the receiver. After sending a data packet the DLL turns on a replay timer, So if the transmitting device does not receive a NACK or an ACK before the timer expires it resends the data packet to the receiver. If the transmitting device's replay timer expires four consecutive time or if it receives 4 NACK DLLPs then Data Link Layer sends a signal to the adjacent Physical Layer to undergo Link Training and sending of data resumes after the PL-PL link training is complete.

The Data Link Layer also has a Flow Control Mechanism which ensures that the receiving device has enough space to store the incoming data packet. The DLL of receiving device has buffer space for each and every type of data packet which is being sent by the transmitter, once those buffer expires it sends a NACK to transmitting device.

After PL Link Training, both(Transmitter and Receiver) sides exchange their initial limits(InitFC) of their buffer sizes. As each receiver processes incoming packets, it updates the limits for its link partner(Transmitter), so it can use the buffer space released. UpdateFC DLLP packets are sent periodically to announce the new credit limits.

Transaction Layer in PCIe:

The Transaction Layer(TL) is the source of data which is being sent between two PCIe devices. The data generated by Transaction Layer is called Transaction Layer Packet(TLP). On the Receiver side the Transaction Layer receives data from the Data Link Layer in the form of TLP and sends it to the software layer of the receiving side.

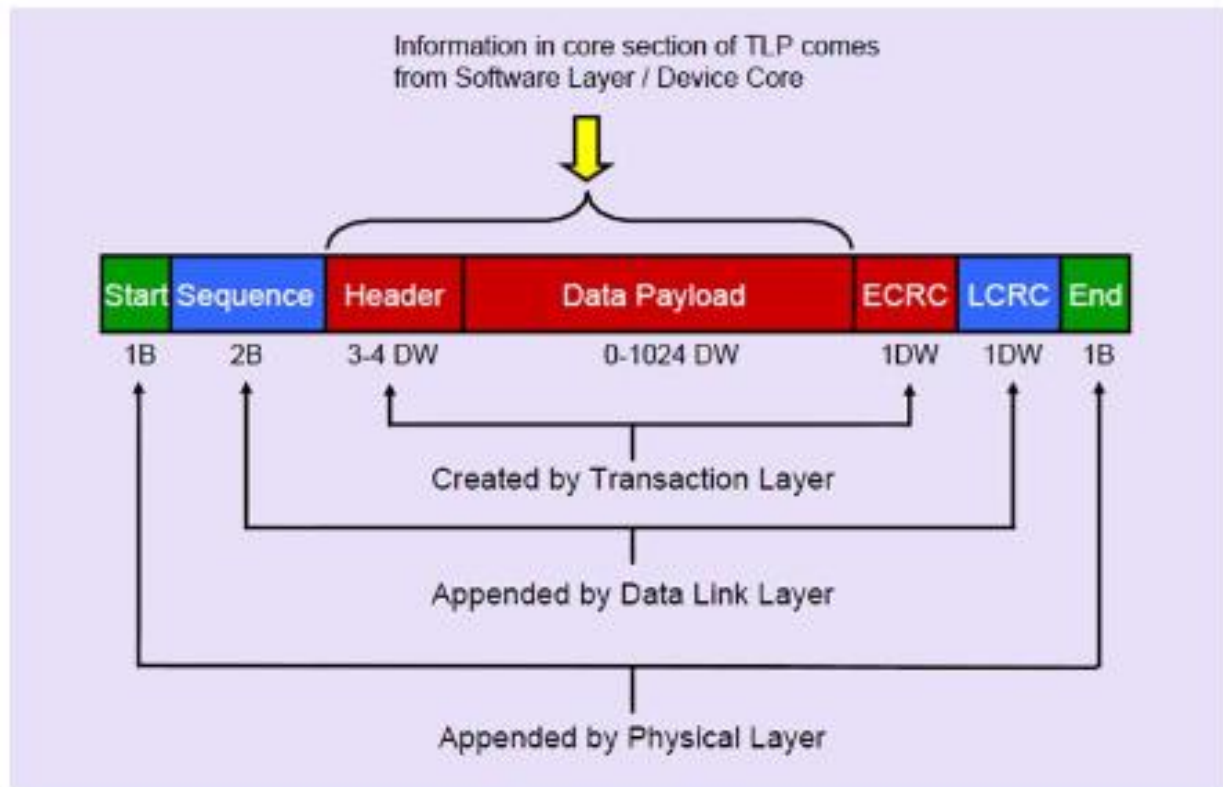


Figure 15: General TLP format

Each TLP has three entities:

1. Header: It indicates types of transactions(Memory Read, Memory Write etc.)
2. Data: This is the actual data which is to be transferred.
3. ECRC(Optional): Use to check whether or not the TLP which is sent got corrupted on its way to the endpoint.

TLP Categories:

There are three types to TLP categories.

1. **Non-Posted Transactions:** When these types of transaction are sent the end receiver is expected to return some data to the transmitting device.
Example: Memory Read, Cfg Read, Cfg Write, I/O Read, I/O write etc.,
2. **Posted Transactions:** When these types of transaction are sent the end receiver is not expected to return some data to the transmitting device.
 - a. Example: Memory write, Message.
3. **Completion Transaction:** Response to Non-Posted Transaction.

There are 4 types of TLP requests which are sent among PCIe devices

- ❖ Memory Read/Write(MRD/MWR): These are used to transfer data from/to the mapped memory location.
- ❖ IO Read/Write(IORD/IOWR): These are used to transfer data from/to I/O location.
- ❖ Configuration Read/Write(CFGRD/CFGWR): These are used to configure the PCIe devices in the bus hierarchy.
 - Always originate from Root Complex.
- ❖ Message Transaction: These are used for signaling an event and general messaging. Supports Vendor defined messages.

Each TLP request generated at the transmitter is routed to its by three ways.

- ID Routing: Configuration requests, ID routed messages and completions.
- Address Routing: It is used with memory and I/O request.
- Implicit Routing: Used for Interrupts and general purpose messaging.
 - Packet is Routed based on Code present in header.

Chapter 4

My Contribution

4.1 Work done in Verilog:

After Learning Verilog if have written code for some basic designs which are shown below.

1. Not gate:

```
module not_gate(output y, input a);  
    not #1(y,a);  
endmodule
```

2. Andgate :

```
module andgate(output y, input a,b);  
    assign y = a & b;  
endmodule
```

3. D flip flop:

```
module dflop(output reg Q, input D,clk,rst);  
    always @(posedge clk,negedge rst)  
    begin  
        if(rst)  
            Q <= 1'b0;  
        else if(1)  
            Q <= D;  
    end  
endmodule
```

4. 2:1 Multiplexer:

```
module mux21(output reg y, input a,b,s);  
    always @(s,a,b)  
        begin  
            if(s)  
                y = a;  
            else  
                y = b;  
            end  
        end  
endmodule
```

5. Adder:

```
module fullAdder(output s,cout, input a,b,cin);  
    assign s = a ^ b ^ cin;  
    assign cout = (a&b) | (b&cin) | (cin&a);  
endmodule
```

6. 4 bit Counter:

```
module counter4b(output reg[3:0] Q,  
                output y,  
                input reg[3:0] D,  
                input clk,ct,ld,clr);  
    always @(posedge clk, posedge clr)  
        begin  
            if(clr)  
                Q = 4'b0000;  
            else if(ld)  
                Q = D;  
            else if(ct)  
                Q <= Q + 4'b0001;  
            end  
            assign y = ((Q == 4'b1111) && (~ld) && (ct));  
        end  
endmodule
```


7. Shift Register:

```
module shift(output reg[1:0]Q, input clk,rst,x);
    always @(posedge clk, posedge rst)
        begin
            if(rst)
                Q = 2'b00;
            else
                Q = {x,Q[1]};
        end
    end
endmodule
```

8. Johnson Counter:

```
module johnson(output reg[2:0]Q,input clk,rst,x);
    always @(posedge clk, posedge rst)
        begin
            if(rst)
                Q = 3'b000;
            else if(x)
                Q = {~Q[0],Q[2:1]};
        end
    end
endmodule
```

9. Ring Counter:

```
module ringc(output reg[2:0] Q, input clk,rst,x);
    always @(posedge clk,posedge rst)
        begin
            if(rst)
                Q = 3'b001;
            else if(x)
                begin
                    if(Q == 3'b100)
                        Q = 3'b001;
                    else
                        Q = Q*2;
                    end
        end
    end
endmodule
```

11. Sequence Detector(11001):

```

module moore101(output y,input x,clk,rst);
    parameter S0 = 3'b000,S1 = 3'b001,S2 = 3'b010,S3 = 3'b011,S4 = 3'b100,S5 = 3'b101;
    reg [2:0]state,next_state;
    always @(posedge clk,posedge rst)
    begin
        if(rst)
            state = S0;
        else
            state = next_state;
    end
    always @(state,x)
    begin
        case(state)
            S0: next_state = x?S1:S0;
            S1: next_state = x?S2:S0;
            S2: next_state = x?S2:S3;
            S3: next_state = x?S1:S4;
            S4: next_state = x?S5:S0; //S4: next_state = x?S1:S0; for mealy
            S5: next_state = x?S2:S0; //no S5 for mealy
        endcase
    end
    assign y = (state == S5); // for mealy:((state == S4) & (x == 1));
endmodule

```

12. Sequence Detector(101):

```

module moore101(output y,input x,clk,rst);
    parameter S0 = 2'b00,S1 = 2'b01,S2 = 2'b10,S3 = 2'b11;
    reg [1:0]state,next_state;
    always @(posedge clk,posedge rst)
    begin
        if(rst)
            state = S0;
        else
            state = next_state;
    end
    always @(state,x)
    begin
        case(state)
            S0: next_state = x?S1:S0;
            S1: next_state = x?S1:S2;
            S2: next_state = x?S3:S0;
            S3: next_state = x?S1:S2;
        endcase
    end
    assign y = (state == S3);
endmodule

```

4.2 Work Done in SystemVerilog:

In Systemverilog if have written code to build a test bench to test the functionality of a Simple Read/Write Memory which has 256 Registers each capable of storing 32 bit data.

Block Diagram of the Memory Model:

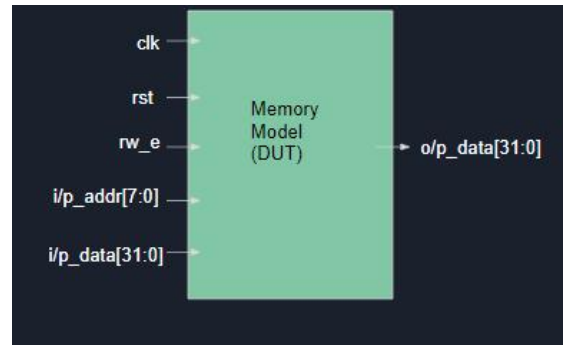


Figure 16: Memory Model

Test Plan:

- Write Random values to each register and read.
- Write all 0's and read.
- Write all 1's and read.
- Write 0's to a particular address and read from all.
- Write 1's to a particular address and read from all.

Block Diagram of TestBench:

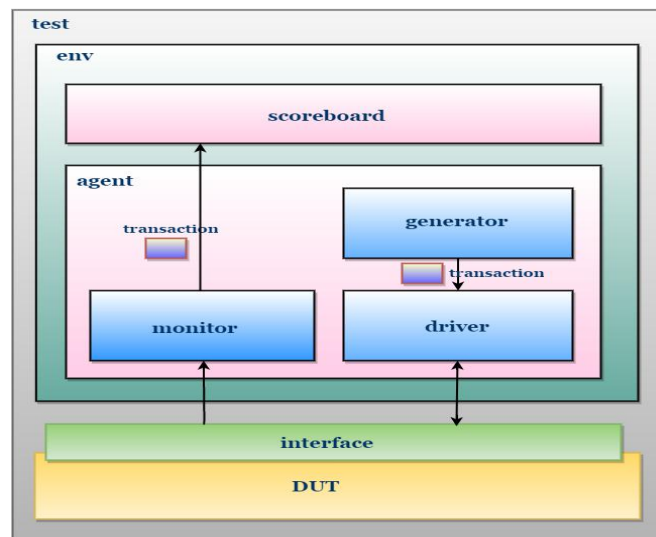


Figure 17 : Testbench for Memory Model

Results obtained after running above test cases:

Test case 1: Write Random values to all registers and Read from them.

```
-----GENERATOR-----
PACKET PLACED IN MAILBOX.
PACKET DATA: ADDR : 5d, WR_DATA : 59e33d12, READ/WRITE: 0
-----DRIVER-----
DATA WRITTEN : 59e33d12, ADDR : 5d
-----MONITOR-----
PACKET PLACED IN MAILBOX.
PACKET DATA:
ADDR : 5d, RD_DATA : 3510896e, READ : 0
-----SCOREBOARD-----
ADDR: 5d DATA EXPECTED: 3510896e DATA OBTAINED : 3510896e
DATA SUCCESSFULLY TRANSFERED!
SUCCEEDED 500 TIMES
```

Test case 2 and 3 : Write all zeros and ones to all registers and read from each register.

```
-----GENERATOR-----
PACKET PLACED IN MAILBOX.
PACKET DATA: ADDR : 5d, WR_DATA : 59e33d12, READ/WRITE: 0
-----DRIVER-----
DATA WRITTEN : 59e33d12, ADDR : 5d
-----MONITOR-----
PACKET PLACED IN MAILBOX.
PACKET DATA:
ADDR : 5d, RD_DATA : 00000000, READ : 0
-----SCOREBOARD-----
ADDR: 5d DATA EXPECTED: 00000000 DATA OBTAINED : 00000000
DATA SUCCESSFULLY TRANSFERED!
SUCCEEDED 500 TIMES
```

Test case 4 and 5: Write all zeros or ones to one register and read from all registers.

```
-----GENERATOR-----
PACKET PLACED IN MAILBOX.
PACKET DATA: ADDR : 5d, WR_DATA : 59e33d12, READ/WRITE: 0
-----DRIVER-----
DATA WRITTEN : 59e33d12, ADDR : 5d
-----MONITOR-----
PACKET PLACED IN MAILBOX.
PACKET DATA:
ADDR : 5d, RD_DATA : ffffffff, READ : 0
-----SCOREBOARD-----
ADDR: 5d DATA EXPECTED: ffffffff DATA OBTAINED : ffffffff
DATA SUCCESSFULLY TRANSFERED!
SUCCEEDED 500 TIMES
```

4.3 Work done in UVM:

After learning UVM I have written code to build a testbench which test the functionality of a full adder model.

Description of the Model: Output(c_out) is addition of two inputs(a_in and b_in) whenever val is '1'.

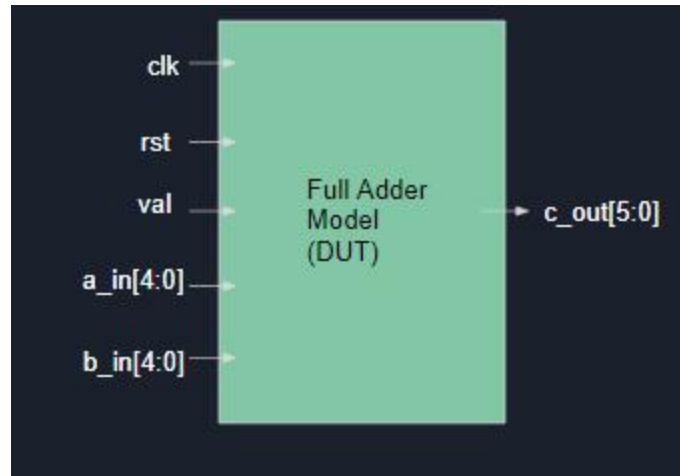


Figure 18: Full Adder Model

Block Diagram of Testbench:

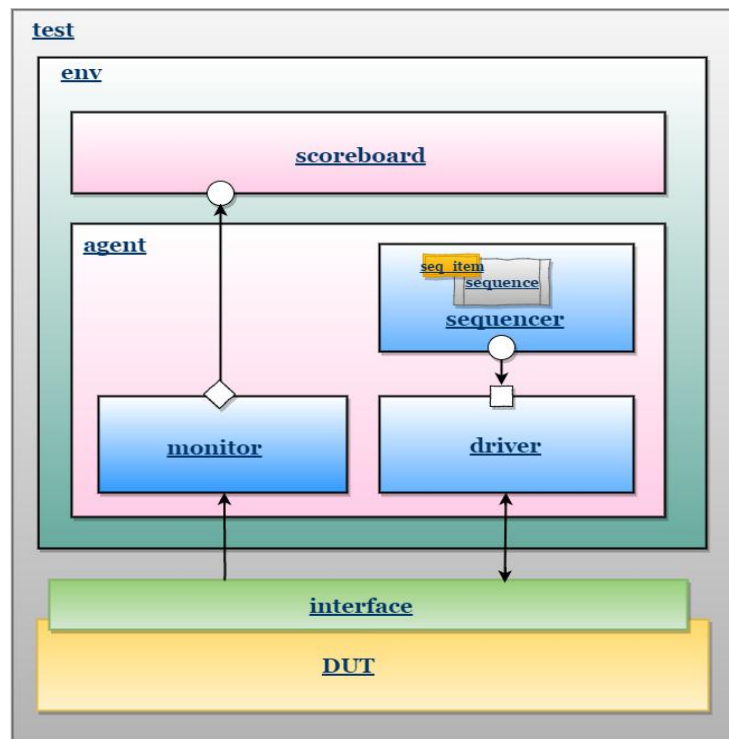


Figure 19: Testbench for Full Adder Model

Test Plan: Give Random values as input to a_in, b_in and val and check whether c_out is addition of a_in and b_in whenever val is '1'.

Results obtained after running test cases:

1. val is '1', a=10, b = 13 , So c = 23 which is true.

Name	Type	Size	Value
fa_req	fa_data_item	-	@669
a	integral	5	'h10
b	integral	5	'h13
c	integral	6	'h23
val	integral	1	'h1

UVM_INFO fa_scoreboard.sv(26) @ 1991: uvm_test_top.env.scoreboard [fa_scoreboard] Addition success!!

2. val is '0' , So irrespective of values of inputs its and invalid signal.

Name	Type	Size	Value
fa_req	fa_data_item	-	@669
a	integral	5	'h12
b	integral	5	'hd
c	integral	6	'h0
val	integral	1	'h0

UVM_INFO fa_scoreboard.sv(30) @ 2011: uvm_test_top.env.scoreboard [fa_scoreboard] Invalid Signal

4.4 Work Done in PCI Express:

Several test cases were written to check the functionality of Root Complex and EndPoint devices in PCIe. The main purpose of verification is to check whether the design is as per specification or not. This is done through running the designs in different possible scenarios(test cases). To ensure that design is correct, first we have to ensure that the code written for verifying those features is correct, otherwise we can get errors even though there aren't any in the design. This is exactly what I have done, resolving code written for those test cases(more than 60) and to check that there are no errors.

Errors found while resolving those test cases were of three types:

1. Genuine error in design
2. Error in code written for testbench
3. Error in code written in the script which validates the simulation report obtained after running the test case.

4.5 Work done in UART:

1. Testing of Xterm enable feature: When a UART device is xterm enabled, a window or a terminal called xterm opens in which we can write any data which we want to send to other UART device or if some other device is sending data it gets displayed on them on the xterm terminal. To this feature I had sent data from other device to our UART device containing a string of characters whose ASCII values are from 32 to 127. Later I also tried to enter some text on the xterm to check whether it gets received at the other end.

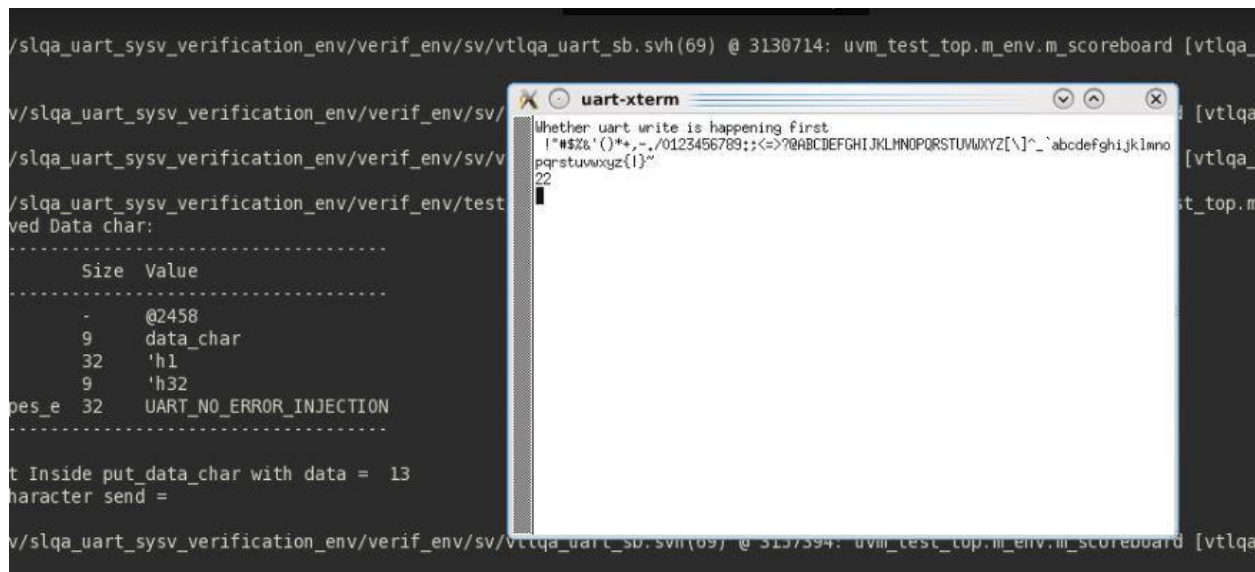
Code for test case:

```
class test_xterm_1 extends vtlqa_base_test; //ASCII : 32 -> 127
`uvm_component_utils(test_xterm_1)
typedef uart_data_sequence_xterm vip_data_seq_t;
bit[7:0] test_str[256];
int i = 0;
function new( string name , uvm_component parent );
    super.new(name,parent);
endfunction

function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    for(i=0;i<96;i++) begin
        test_str[i] = 32+i;
    end
endfunction

task run_phase(uvm_phase phase);
    vip_data_seq_t vip_data_seq = vip_data_seq_t::type_id::create("vip_data_seq");
    phase.raise_objection(this, "Started test_xterm_1");
    xterm_enable = 1;
    super.run_phase(phase);
    vip_data_seq.send = string'(test_str);
    // test_str contains character with ASCII values 32 to 127
    fork
        begin
            $display("VIP SEQUENCE STARTED");
            vip_data_seq.start(m_env.m_virtual_sequencer.uart_vip_sqr);
            $display("VIP SEQUENCE FINISHED");
        end
        join_none
        $display("WAIT CLOCK STARTED");
        repeat(2000000)
            m_config.m_uart_vip_cfg.wait_for_clock();
    phase.drop_objection(this, "completed test_xterm_1");
endtask
endclass
```


Result: In the below we can see string of characters(from ! to ^) getting displayed on the xterm. After that we entered text '22' on the xterm and in the background we can see char with ASCII value 32(which is 2) getting received by the other device.



The screenshot shows a terminal window with the following content:

```

/sqlqa_uart_sysv_verification_env/verif_env/sv/vtlqa_uart_sb.svh(69) @ 3130714: uvm_test_top.m_env.m_scoreboard [vtlqa_
v/sqlqa_uart_sysv_verification_env/verif_env/sv/vtlqa_uart_sb.svh(69) @ 3130714: uvm_test_top.m_env.m_scoreboard [vtlqa_
/sqlqa_uart_sysv_verification_env/verif_env/sv/vtlqa_uart_sb.svh(69) @ 3130714: uvm_test_top.m_env.m_scoreboard [vtlqa_
ved Data char:
-----
Size Value
-----
- @2458
9 data_char
32 'h1
9 'h32
pes_e 32 UART_NO_ERROR_INJECTION
-----

t Inside put_data_char with data = 13
character send =
v/sqlqa_uart_sysv_verification_env/verif_env/sv/vtlqa_uart_sb.svh(69) @ 3130714: uvm_test_top.m_env.m_scoreboard [vtlqa_

```

Overlaid on the terminal is an xterm window titled 'uart-xterm'. It displays the following text:

```

Whether uart write is happening first
! " # $ % ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
22

```

2. Pure Receive Mode: When a UART device is in pure receive mode in xterm enable form, then even if any text is entered in the xterm terminal, that data won't get received at the other end. Only the data which the device receives is displayed on the xterm terminal.

Code for pure receive mode test case:

```

class pure_rece_mode_test extends vtlqa_base_test;
`uvm_component_utils(pure_rece_mode_test)
typedef uart_data_sequence_xterm vip_data_seq_t;
function new( string name , uvm_component parent );
    super.new(name, parent);
endfunction

function void build_phase(uvm_phase phase);
    super.build_phase(phase);
endfunction

task run_phase(uvm_phase phase);
    vip_data_seq_t vip_data_seq = vip_data_seq_t::type_id::create("vip_data_seq");
    phase.raise_objection(this, "Started pure_rece_mode_test");
    xterm_enable = 1;
    pure_rece_enable = 1;
    super.run_phase(phase);
    vip_data_seq.send = "12345";
    fork
        begin
            $display("VIP SEQUENCE STARTED");
            vip_data_seq.start(m_env.m_virtual_sequencer.uart_vip_sqr);
            $display("VIP SEQUENCE FINISHED");
        end
    join_none
    $display("WAIT CLOCK STARTED");
    repeat(2000000) m_config.m_uart_vip_cfg.wait_for_clock();
    phase.drop_objection(this, "completed pure_rece_mode_test");
endtask
endclass

```


[illegible]

Page

Chapter 5

Conclusion

During the course of my Internship, I have learnt the importance of verification, how to build test benches for different digital designs, how to write test cases to cover all possible test cases. I have also explored one of the important protocols called PCI Express and tested features it offers. I have also tested UART xterm enable feature and pure receive mode feature provided by the designers.

5.1 Future Works

As data transfer speeds between different peripherals is increasing day by day, new features are being to PCI Express to ensure effective transfer of data between devices. As new complex features are added to PCI Express, it also the responsibility of verification engineers to verify those new features by preparing complex test cases to cover each and every scenario provided by those new features.

References

- [1] PCI Express® Base Specification Revision 5.0 Version 0.9, PCI-SIG 18 October 2018
- [2] PCI Express System Architecture: Mindshare Inc., Ravi Budruk, Don Anderson, Tom Shanley:ISBN-13: 978-0321156303, ISBN-10: 0321156307
- [3] <http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1>
- [4] <https://resources.infosecinstitute.com/system-address-map-initialization-x86x64-architecture-part-2-pci-express-based-systems/#gref>
- [5] <http://www.sionsemi.com/whitepapers/pcie-overview.html>
- [6] <https://www.verifcationguide.com/p/home.html>
- [7] <https://www.chipverify.com/>
- [8] <http://www.circuitbasics.com/basics-uart-communication/>

