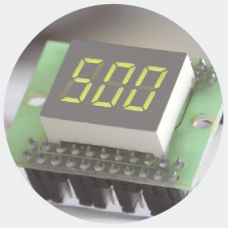
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Prototipagem PCIe com o Speed Adaptor no HAPS-100

FILIPE CHEN
Outubro de 2021

POLITÉCNICO
DO PORTO

Instituto Politécnico do Porto
Instituto Superior de Engenharia do Porto

Prototyping PCIe with Speed Adaptor in HAPS-100

Filipe Chen

Master in Electrical and Computer Engineering
Specialization Area of Telecommunications

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

October, 2021

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Telecommunications.

Candidate: Filipe Chen, No. 1160642, 1160642@isep.ipp.pt

Scientific Guidance: Prof. Manuel G. Gericota, mgg@isep.ipp.pt

Company: Synopsys, Inc.

Advisor: Pedro Moreira, Pedro.Moreira@synopsys.com

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

October, 2021

Acknowledgements

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would first like to thank my supervisor, Professor Manuel Gradim de Oliveira Gericota, whose expertise was invaluable in formulating the research questions and methodology. Your insightful and funny feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank my tutors, Engr. Pedro Morerira and Engr. Juliano Raimundo, for their valuable guidance throughout my stay at Synopsys. You provided me with the tools that I needed to choose the right direction and successfully complete my dissertation.

Finally, I could not have completed this dissertation without the support of my family and my friends who provided stimulating discussions as well as happy distractions to rest my mind outside of my research.

Abstract

FPGAs are powerful components as they allow for faster verification, helping accelerate the development of a prototype. Nonetheless, as technology advances, they need to adapt to the new configurations and speeds which boost the performance previously known.

Therefore, Speed Adaptors are needed to close this speed gap so that the timings of data transfer match, resulting in proper verification procedures and correct debugging.

Throughout this work, many topics related to the Speed Adaptor are introduced, providing enough information for the reader to understand the basic architecture of the Speed Adaptor system.

The Speed Adaptor requirements got met following the RTL design flow, which serves as a guideline to program the FPGA.

Accordingly, Synopsys' tools got used to analyse the Speed Adaptor simulation and search for problems during the implementation phase, such as timing closure and area occupation.

Moreover, as the simulation problems arose, timing closure and some methodologies got explored, displaying why a negative Worst Hold Slack is a problem and how the solution got debugged by rerouting and replacing clocks.

Due to the delayed configuration of the HAPS-100, a Synopsys rapid prototyping platform, and the delay debugging the timing closure, the hardware setup is still half complete. However, it could still be possible to document the successful link up with the high-speed side of the system connected to the PCIe endpoint.

Although it was not possible to test the HAPS-100 to compare to the simulation results, this dissertation successfully delivers the information needed for the reader to comprehend the Speed Adaptor functionalities.

Keywords: FPGA, Prototyping, Simulation, Emulation, Timing Closure, PCIe, Speed Adaptor, Design flow.

Resumo

As FPGAs são componentes poderosos pois permitem uma rápida verificação na fase inicial dos projectos possibilitando a aceleração do desenvolvimento do protótipo. Contudo, a tecnologia evolui constantemente, por isso existe a necessidade de se adaptar às novas configurações e velocidades existentes, uma vez que possuem melhor desempenho.

Deste modo, dispositivos como os *Speed Adaptors* são utilizados para estabelecer ligação entre tecnologias que funcionam a diferentes velocidades. Pois, garante-se o alinhamento das janelas de transferência, permitindo uma verificação exata e uma correção fidedigna.

No desenvolver da dissertação são introduzidos vários tópicos relacionados com o *Speed Adaptor* a fim de que o leitor possua conhecimento suficiente para perceber a arquitetura do sistema do *Speed Adaptor*.

A programação e implementação do *Speed Adaptor* alcançou-se seguindo um guia de desenvolvimento denominado por *Design Flow* que detalha todos os passos desde o esboço à implementação no sistema da FPGA.

Foram utilizadas as ferramentas da Synopsys para efetuar esses passos e analisar a simulação do sistema de forma a procurar na fase da implementação por erros como o *timing closure* ou a área de ocupação das células de memória.

Ademais, explorou-se o tema do *timing closure* e metodologias que solucionassem esse problema. A dificuldade surgiu com a existência do valor negativo para o *Worst Hold Slack*, por isso detalhou-se o obstáculo e a solução que foi obtida através da modificação dos roteamentos existentes e da substituição dos relógios.

Apesar de não se ter realizado testes com o HAPS-100, uma plataforma de prototipagem rápida da Synopsys, foi possível demonstrar a configuração do hardware e a conexão parcial do sistema na ligação do *Speed Adaptor* ao PCIe endpoint. Concluindo, a presente dissertação dispõe a informação necessária para compreender a utilidade e as funcionalidades do Speed Adaptor.

Palavras-Chave: FPGA, Prototipagem, Simulação, Emulação, Timing Closure, PCIe, Speed Adaptor, Design flow.

Contents

List of Figures

List of Tables

List of Acronyms

ADC	Analog-to-digital converter
ASIC	Application Specific Integrated Circuit
CDM	Configuration-Dependent Module
CLB	Configurable Logic Block
CXPL	Common Express Port Logic
DDR	Double Data Rate
DLL	Data Link Layer
DLLP	Data Link Layer Packet
DSP	Digital Signal Processor
DVE	Discovery Visual Environment
ECRC	End to End Cyclic Redundancy Check
EIEOS	Electrical Idle Exit Ordered Set
EIOS	Electrical Idle Ordered Set
EP	End Point
FPGA	Field-Programmable Gate Array
FSM	Finite State Machines
FTS	Fast Training Sequence
GT	Gigatransfer
HAPS	High-Performance ASIC Prototyping System
HDL	Hardware Description Language
IO	Input/Output
IOB	Input/Output Block
ISEP	Instituto Superior de Engenharia do Porto
JTAG	Joint Test Action Group
LCRC	Link Cyclic Redundancy Code
LFSR	Linear-Feedback Shift Register
LTSSM	Link Training and Status State Machine
LUT	Look Up Table
NCD	Native Circuit Description
NGD	Native Generic Database
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PIPE	Physical Interface for PCI Express Specification

PLP	Physical Layer Packet
QoS	Quality of Service
RADM	Receive Application-Dependent Module
RC	Root Complex
RTL	Register Transfer Level
SA	Speed Adaptor
SLL	Super Long Lines
SLR	Super Logic Region
SoC	System-on-Chip
SOS	Skip Ordered Set
SSI	Stacked Silicon Interconnect
STA	Static Timing Analysis
TC	Traffic Classes
TEDI	Thesis/Dissertation
TLP	Transaction Layer Packet
TS	Training Sequence
UCF	User Constraints File
USB	Universal Serial Bus
URG	Unified Report Generator
XADM	Transmit Application-Dependent Module

Chapter 1

Introduction

In this modern world, we witness an ever-evolving technology in almost all fields. And through all of these areas of expertise, the electronic field and its subsequent roots are widely present, whether it's over hardware or software, both these types of mediums are constantly growing and in increased demand.

Therefore, adaptability and compatibility between the existing devices and future devices are present issues and will be future addressable issues. Mainly, in the sector of micro-electronics, there is always the need to progress the performance of the chips and their peripherals so that the devices can achieve higher speed and enable humans to research and develop other sectors faster than ever.

The data transmission between equipment requires devices that are compatible or the presence of a bridge that connects the older or slower devices with the newer, faster devices that might have different and innovative architectures.

Compatibility is a crucial step as it gives room to be flexible and create devices that possess many features which can work together well to form innovative ideas or simply because the system requires the functions of two very distinct appliances.

This dissertation is an introductory guide to a device known as Speed Adaptor that bridges fast technology that runs on Peripheral Component Interconnect Express (PCIe) with slower running devices.

The purpose of this adaptor is to connect a testing and simulation platform (FPGA) that runs at a slower pace due to its restrictions to the high-speed devices, accelerating the development and implementation due to preemptive debugging and optimisation.

Thus, they are needed for their preemptive emulation as there are many fields in which resources are not renewable or replaceable. This technology appeals to a wide range of sectors such as consumer electronics, medical devices, security systems, and defence industry applications [1].

1.1 Contextualisation

This project stems from the unit course Thesis/Dissertation (TEDI) of the program Master in Electrical and Computer Engineering, Specialisation Area of Telecommunications from Instituto Superior de Engenharia do Porto (ISEP) while being developed during the internship at Synopsys.

The development of this project covers the understanding of the overall architecture of the Speed Adaptor (SA) that integrates with the field of FPGA prototyping to allow an FPGA environment to transfer data with external PCIe devices.

Furthermore, this project will enable Synopsys prototyping team in Porto office to get a general overview of this adaptor basic functionalities and operation modes.

1.2 Problem definition

Prototyping IPs allows their users to test their design specifications. However, as technology evolved, they have become more complex, and of higher dimensions, they become prone to constraints due to the clock limitations and finite resources of FPGAs [2].

To mitigate the speed problems, speed adapters that bridge the speed between the fast-physical interface and the testing and simulation logic running in FPGAs are a solution that does not compromise the protocol functionality.

1.2.1 Objectives

The following work exposes a state-of-the-art speed adapter for FPGA Prototyping, followed by the Synopsys Speed adapter solution to interface PCIe Protocol.

Therefore, the main problem that got outlined was achieved by the following objectives set:

1. Acquire the fundamental knowledge to FPGAs and the protocol PCIe;
2. In-depth study of the Speed Adaptor and its integration with PCIe;
3. Basic handling of the Synopsys tools and Register Transfer Level (RTL) flow;
4. Emulating Speed adaptor operation through Synopsys tools;

5. Application of the same testing values on the Synopsys' HAPS-100 machine, a in-house rapid prototyping platform;

6. Analysis and comparison of the output results from each of the tests.

1.3 Document organisation

This document is divided in the following chapters:

- Chapter 1: The first chapter is the introductory chapter where the main objectives of this dissertation are, defining the problem at hand and the solution to achieve;
- Chapter 2: The second chapter is where all the preliminary knowledge to understand the Speed Adaptor gets presented;
- Chapter 3: The third chapter gives detailed information on the speed adaptor's architecture, demonstrating its inner workings, followed by a brief description of its timing requirements;
- Chapter 4: The fourth chapter demonstrates the synthesis procedure and explanation of the results obtained;
- Chapter 5: The fifth chapter explains the method to solve the timing closure issue and outlines the implementation process on the hardware, presenting the devices utilised;
- Chapter 6: The final chapter concludes whether the main problem is or is not satisfied according to the objectives set in the introductory chapter.

Chapter 2

State of the art

This chapter gives the reader a brief background of the technologies used throughout this report.

These technologies include a small introduction to Field-Programmable Gate Array (FPGAs), their applications and user cases, and the current products on the market. Subsequently, the same analysis gets executed for the topics of prototyping and emulation.

Moreover, an in-depth study of PCIe gets explored to enable the reader to understand the inner workings of a Speed Adaptor for PCIe, including the basic architecture, link setup, communication protocols and packet delivery.

2.1 FPGAs

The FPGA is an integrated circuit that enables its users to customise its functionality.

The advantage of using this technology stem from its flexibility, enabling the faster development of new products, and thus reducing time-to-market compared to Application-Specific Integrated Circuits (ASIC).

On the other side, FPGA programmable cells take up more space than ASIC standard cells and perform worse.

The figure 2.1 demonstrates the three modules that exist in a general FPGA architecture:

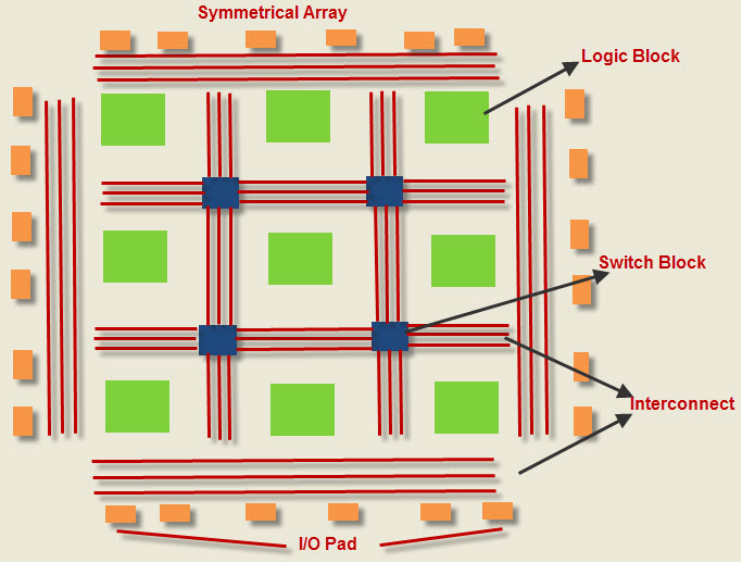


Figure 2.1: FPGA Architecture [3]

- I/O blocks enable communication with applications from outside of the environment;
- Switch Matrix and interconnects provide connections between the logic blocks;
- Configurable Logic Blocks (CLB) contain the memory elements [4].

An example of a CLB shows in figure 2.2. The Look Up Table (LUT) function generator is the core building block and possesses a variable input intake.

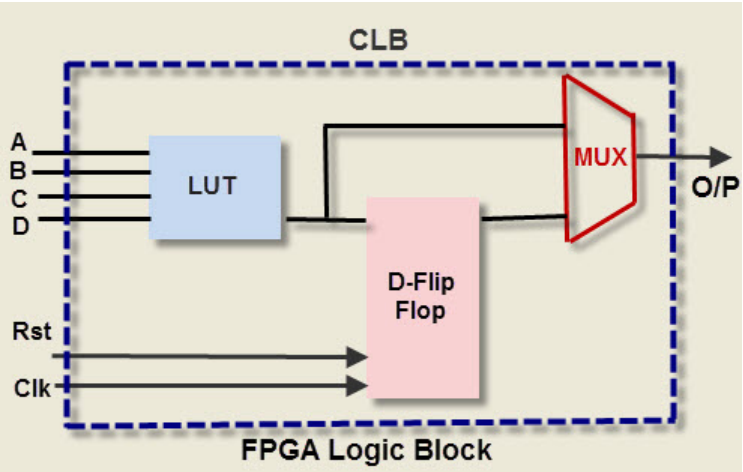


Figure 2.2: FPGA CLB [3]

While FPGA possesses reprogrammable parts, ASIC, as the name entails, is focused on a specific job [5]. However, FPGAs are more versatile as they can be part of any project as there are low, mid, or high-end FPGA for all purposes.

In the current state of the market, ASICs are by far more powerful than the FPGA's. However, ASIC enhancement has slowed down because device technology sizes cannot further reduce, as predicted by Moore's Laws [6].

In the table 2.1 are shown some of the differences mentioned above.

Table 2.1: ASIC vs FPGA comparison (Adapted from [7])

ASIC	**FPGA**
ASICs can't be reconfigured with different designs. They contain only one design in them for their whole life time.	FPGAs can be reconfigured with different designs.
It's suited for bulk production.	Large FPGA platforms are not suited for bulk production.
More energy efficient and less power consumption than FPGAs.	More power hungry than ASICs.
Higher frequencies can be achieved on the same node	Limited in operating frequency on a particular node
It's only built after validating the design for permanent use of that design.	It's used to prototype and validate a design.

In some cases, FPGAs require external high-performance multi-rate transceiver boards to enable FPGA design to achieve its functionality at higher frequencies that were not possible with the stand-alone configuration.

Commonly, the hard IP blocks utilise standard bus interfaces using PHY, such as Ethernet, Universal Serial Bus (USB), PCIe and memory interfaces like Double Data Rate (DDR).

An advantage of using this kind of block is the FPGA load it releases by allocating additional external resources. The result translates into a device with faster bring-up time and increased performance [8].

Moreover, an FPGA can partition into a programmable and non-programmable area. In non-programmable areas lays the parts for the configuration interface and configuration logic.

Additionally, many IPs, such as Analog-to-digital converters (ADC), memory blocks, and Digital Signal Processors (DSP), have been combined with FPGAs to ease the load and lessen the bring-up time of the device as usually these blocks are required by many designs.

Whilst the programmable areas include the CLBs, a portion of Input/Output Blocks (IOB) and routing resources, many ways to configure FPGAs exist, such as [8]:

- Serial configuration which is slower than parallel but uses fewer signals;
- In Master mode FPGA controls the configuration on boot, while on Slave mode, an external device controls the configuration;
- Joint Test Action Group (JTAG): Overrides any other configuration mode and was originally designed for testing purposes.

2.2 Hardware Description Languages

Hardware Description Languages (HDL) are primarily used to model digital circuits, their main applications being Synthesis and Verification.

Synthesis is the process of generating a digital circuit from a high-level description of a behavioural and/or structural model. Therefore the model must be appropriated to the destination embedded digital circuitry.

On the other hand, Verification is the step that follows after the digital circuit gets synthesised. As the name implies, this process tests if the outcome of the design is according to the original requirements to check if the system will work as intended [9].

The main focus of an HDL is concurrency hence the parallel operation of different sections of the code to get applied on hardware is the norm in this type of language.

The Verilog language is a widely used HDL language due to its flexibility to code the HDL models since it grants a lot of freedom, which results in more human-friendly code readability and interpretation.

2.3 FPGA Prototyping

FPGA prototyping involves testing ASIC and System-on-Chip (SoC) designs on an FPGA environment to run hardware verification for early software or hardware development.

This method is often called rapid circuit prototyping. It has gained popularity over the past few years for a verification methodology aimed at hardware design [10]. The popularity of prototyping rises from the need to simulate the designs on end-use portable devices that might get placed in adverse conditions [11].

The possibilities inside prototyping provide a large range of advantages [12]:

- Higher speed and capacity to run complex designs that software-only simulator can not run;

- Functional hardware platform before silicon is available;
- Test platform that ensures adaptability for many commercial IP components to work together.

However, there are also some challenges that FPGA prototyping has yet to overcome, for example:

- Many complex designs require more than a single FPGA to be functional, thus needing a partitioning method like pin-multiplexing scheme;
- Long bring-up time is a big issue for many prototypes because of the number of layers and the need to verify the connectivity of these signals and pins. Additionally, accommodating a design to fit in an FPGA prototype hardware is very time-consuming and error-prone;
- Low debug capabilities due to limited memory size and long place and route times to change probes.

Therefore, FPGA prototyping focus on the validation of tasks and testing the integrity of software on hardware. Here are shown some examples of prototyping environments, ranging from small project environments:

- Xilinx Virtex/Arty [13];
- Intel Stratix [14];
- Microsemi RTG4 [15];
- Lattice Semiconductor [16];

to large scale ones, which become platforms that enable the implementation of higher complexity ASICs:

- Synopsys HAPS [17];
- Aldec HES [18];
- Cadence Protium [19].

As mentioned previously, FPGAs enable the rapid prototyping of circuits. In other words, they can quickly validate the functionalities implemented.

The research and testing parts of this work involve the utilisation of Synopsys' tools, and Synopsys possesses its prototyping system, HAPS, which can deliver high performance and easy integration with physical interfaces within a short time frame [17].

The most recent version of the Synopsys prototyping system is the HAPS-100. It comes with one of the latest and most advanced technology on the market, which presents with the advantages shown in figure 2.3.

Figure 2.3: HAPS100 [17]

2.4 Hardware Emulation

While prototyping focuses on validating and testing, hardware emulation focuses on the verification and debugging side of the module design.

With the increased complexity of the chip designs, Hardware Emulation surges as one of the primary solutions to address code errors and bugs on large scale designs.

This verification process can not be debugged by the RTL simulator because of its slow speed when running on the embedded software. As for FPGA-based prototypes, it lacks suitability to perform this task due to the lack of visibility and access into the design [20].

As technologies evolved, Hardware Emulation has become more sophisticated, easier to utilise and increasingly versatile in addressing multiple-use models and verification tasks [21].

Many large scale hardware emulation systems are utilised to prototype processor designs and architectures, some of which are:

- Synopsys ZeBu Server [22];
- ALDEC HES-DVM [23];
- Siemens Veloce Strato [24];
- Cadence Palladium [25].

The most recent and evolved technologies enable users to perform some of the following tasks that increase the performance of this already powerful tool, such as [22]:

- Software early bring-up allows the overall time necessary to create an end product lower since programmers can begin developing code before silicon is available;
- The size and quantity of circuitry inside an emulator allows for power analysis of billions of cycles which in return will provide fast and accurate power estimations;
- Performance validation can measure latency on critical design paths to ensure sufficient software executions speed.

2.5 PCIe

The Peripheral Component Interconnect (PCI) is a hardware bus, which serves as a device to connect internal components to a computer. The express version of this device, Peripheral Component Interconnect Express (PCIe), was introduced later and replaced the original device due to its increased throughput, reduced latency, and improved error detection [26].

Table 2.2 shows the evolution of express technology to the present. For each update that followed, there were changes in the bandwidth that this protocol could achieve, almost doubling its bandwidth from one generation to the next generation. Higher bandwidths mean that lanes can accommodate more data to be transferred, increasing the communication speed [27].

Table 2.2: PCIe Gen. Evolution (Adapted from [28])

	Raw Bit Rate (GT/s)	**Link BW (Gb/s)**	**BW/Laneway (GB/s)**	**Total BW X16 (GB/s)**
PCIe 1.x	2.5	2	0.250	8
PCIe 2.x	5.0	4	0.500	16
PCIe 3.x	8.0	8	∼1	∼32
PCIe 4.0	16	16	∼2	∼64
PCIe 5.0	32	32	∼4	∼128

The figure 2.4 demonstrates a basic PCIe topology. In this topology, there is a Root Complex (RC) which ensures the communication between the Central Processing Unit (CPU), memory and the End Points (EP). However, the PCIe protocol only acts in the connection between the following devices [29]:

- Root Complex (RC);
- End Point (EP);
- Switch;
- PCIe Bridges.

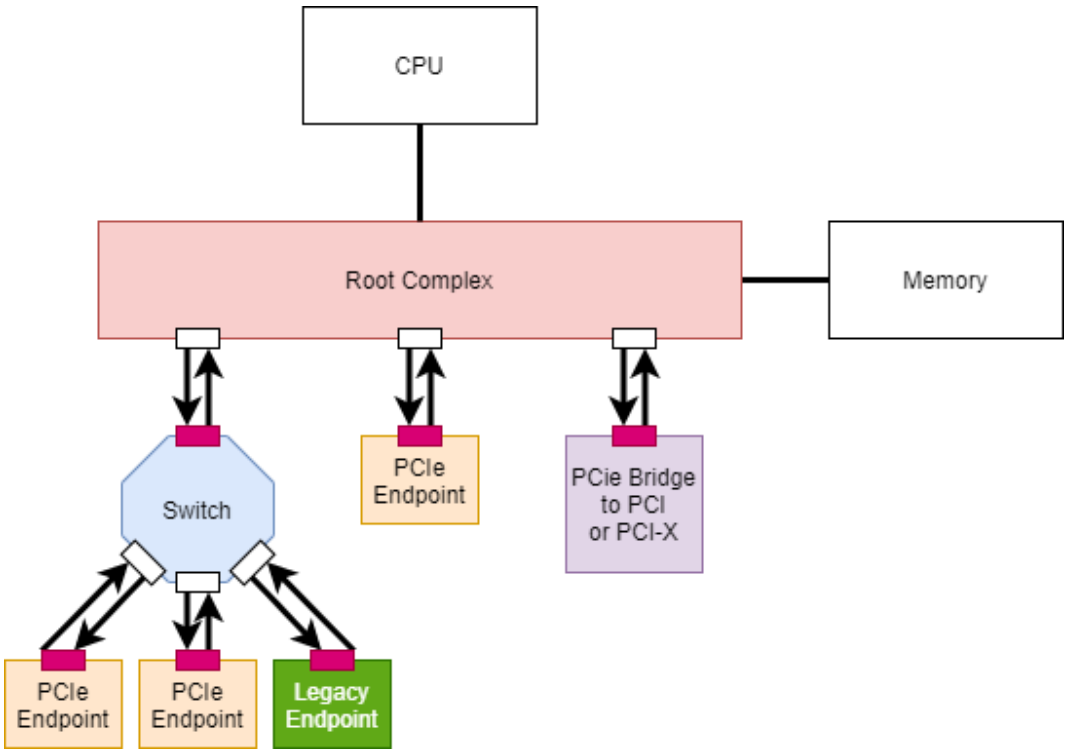


Figure 2.4: PCIe topology (Adapted from [29])

The PCIe transactions most of the time divide into requests and completions.

A requester is a device that begins a transaction in the PCIe links, while a completer is a device that receives the transaction and sends a completion package back to confirm the reception.

EPs are peripheral devices that perform request or completion PCIe transactions to exchange data with interfaces such as Ethernet, graphic cards or USB. Moreover, the EPs divide into PCIe endpoints and legacy endpoints.

This second variant can support Input/Output IO transactions and locked transaction semantics as a completer but not as a requester. It does not require support for 64-bit memory addressing capabilities.

On the other side, the native EPs must be capable of 64-bit memory addressing capabilities.

Also, instead of the previous transactions present on the legacy version, the native EP must support Message Signalling Interface (MSI) style interrupt generation.

A Switch is a device made up of two or more PCI-to-PCI bridges.

While the transactions get sent from one device to another, the switch permits their communication between devices by receiving the Transaction Layer Packets (TLP) packet first and then forward to the respective destination.

The destination address is imprinted in the devices when the system initialises.

Differently, the PCIe Bridge is a device that enables the communication of the PCIe protocol with a different technology which might run at lower speeds, for example, when interacting with an FPGA environment.

2.5.1 PCIe Layers

The architecture of PCIe divides into the following discrete logical layers: Transaction Layer, the Data Link Layer, and the Physical Layer.

The communication between PCIe devices is established through point-to-point serial links enabling requests and interrupts to be sent from one to another endpoint [30].

Figure 2.5 shows the configuration of a PCIe link. Each link can contain one or more lanes. Further, a single lane has two differential signalling pairs: one dedicated for transmission and another for receiving.

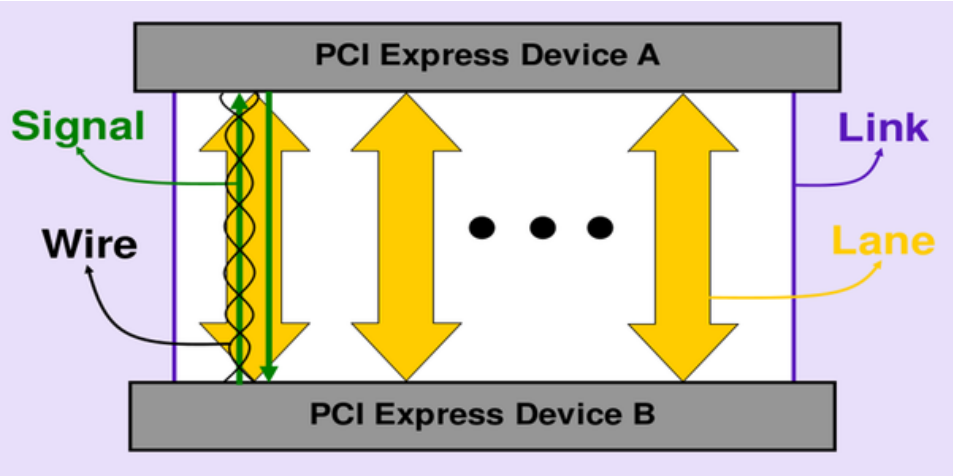


Figure 2.5: PCIe Link Terminology [31]

These PCIe links form protocol layers on both sides, which in return verify the integrity of the packet sent like the one in figure 2.6.

Throughout these layers exists many security measures so that packets are transmitted without any errors, missing packets or out of sequence.

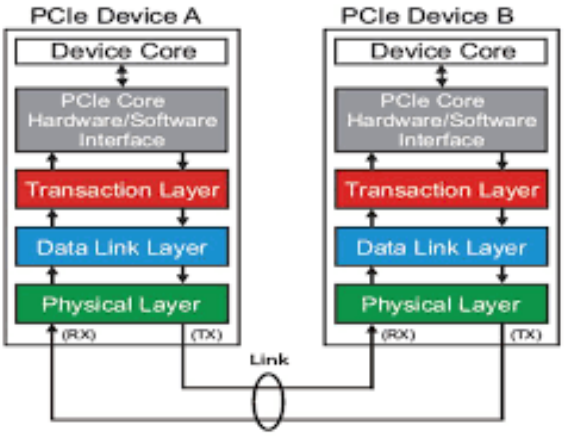


Figure 2.6: PCIe Layers [32]

Transaction Layer

The transaction layer is responsible for creating the Transaction Layer Packets (TLP) that will contain the information we want to deliver.

The packets have functionalities that will verify flow control by tagging the message with a header number and place it on a queue which will dictate if there's an overflow of messages sent. Figure 2.7 shows a visual representation of this mechanism.

The End to End Cyclic Redundancy Check (ECRC) is an additional verifying mechanism that only gets checked on the final destination to confirm that the packet was correctly delivered.

Otherwise, an error report gets generated and sent back to the device which sourced the packet [33].

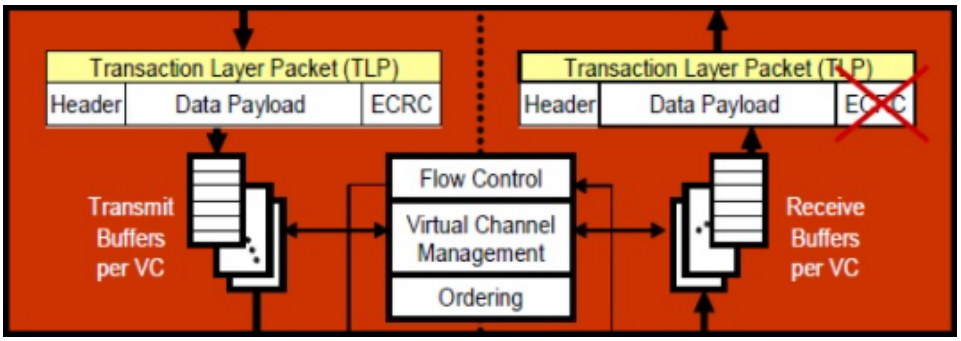


Figure 2.7: Transaction Layer [34]

Data Link Layer

On the Data Link Layer (DLL), the TLP created by the transaction layer gets further modified by adding a sequence and a Link Cyclic Redundancy Code (LCRC) field, which dictates the number of packets sent, and detects if there are any errors on arrival.

Additionally, there is a buffer block that saves a copy of the most recent acknowledged packet. In other words, the block acts as a replay mechanism that sends packets when prompted. Figure 2.8 displays an overall image of the layer.

If the receiver successfully retrieves the packet with the expected sequence number, an acknowledgement gets communicated between both end-users.

However, if the packets received do not match the expected number, it sends a NACK for a retry.

When that happens, the replay buffer generates the packet that follows the most recent acknowledge. And whenever it receives the acknowledgement of a successful package, it discards the saved packet on the replay buffer as it is not needed anymore.

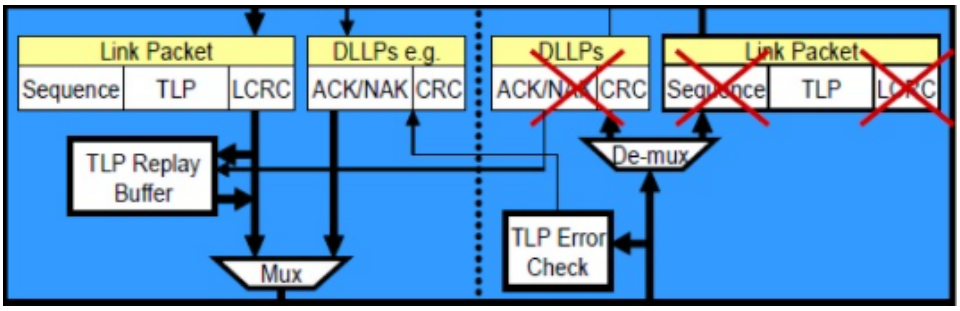


Figure 2.8: Data Link Layer [34]

Physical Layer

The physical layer adds to the Data Link Layer Packet (DLLP) received from the upper layer a start and ending, following the encoding of the packet into a binary format which will form an ordered set and sent through the link, as represented by the figure 2.9.

Furthermore, the link gets established via Link Training and Status State Machine (LTSSM), a block that will indicate all the procedures and configurations for the link.

For example, in the recovery state, the number of available lanes and the speed they will run can be modified.

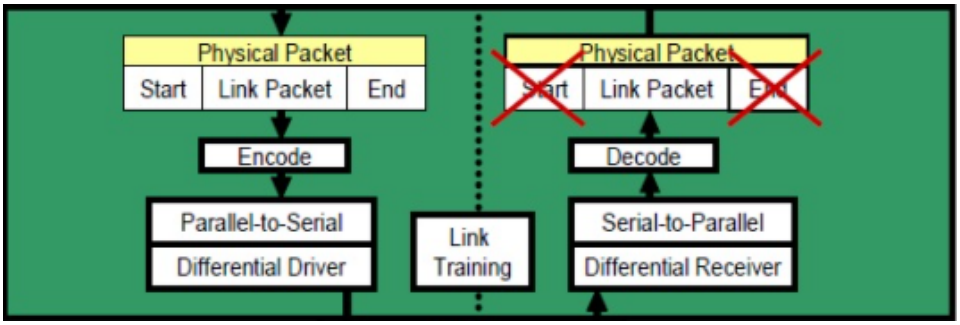


Figure 2.9: Physical Layer [34]

2.5.2 PCIe Protocol

As mentioned, the PCIe architecture divides into three layers, but the physical layer is responsible for managing the links between devices.

Therefore Link Training is the first process that will initialise a device's Physical Layer, port and Link so that traffic can flow in the Link. This process automatically initialises after reset without any software involvement.

The top-level of LTSSM consists of multiple states, each with its sub-states, like the example shows in figure 2.10.

The first LTSSM state entered after exiting the Fundamental reset state, or the Hot reset state is the Detect State. The LTSSM consists of 11 top-level states as presented below [35]:

1. Detect: Detects the presence of an end termination;
2. Polling: Transmits and responds to training ordered sets to establish bit and symbol lock and configure lane polarity;
3. Configuration: Data transfer between transmitter and receiver at a negotiated data rate;
4. Recovery: Enables reconfiguration of data rate operation, bit or symbol lock, block alignment and lane-to-lane skew;
5. L0: Normal operational state where data transfer occurs and where all power management states enter from;
6. L0s (Fast wake-up): Power saving state, which allows skipping recovery state enabling a quick re-establishment of the Link;
7. L1 (Slow wake-up): Additional power-saving state which works on top of L0s with the additional cost of resume latency;

8. L2 (Link suspended): Power conservation state by a forceful shutdown of transmitter and receiver if required;
9. Disabled: Makes a configured Link activate Electrical Idle;
10. Loopback: Test and fault isolation process. The only inputs required are entry and exit behaviour. Can be applied per lane or on a configured link;
11. Hot reset: Reset directed by a higher Layer.

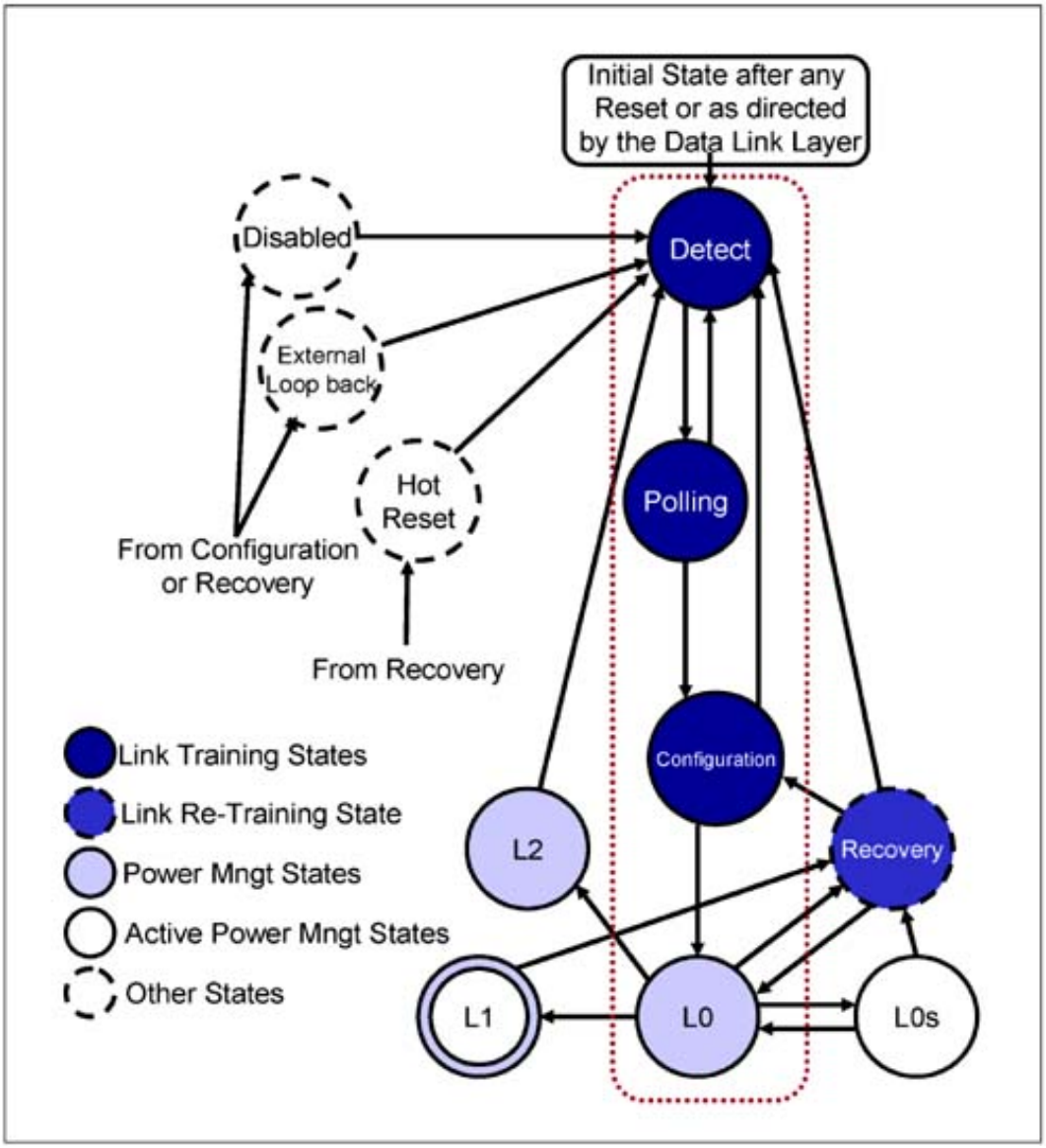


Figure 2.10: LTSSM tree [29]

2.5.3 Link Equalisation

As PCIe evolved, the new versions enabled enhanced speed over the same infrastructure deployed as its first versions, PCIe 1.0 and 2.0.

The difference in speed between devices can be a daunting problem to fix. Therefore Link Equalisation appears to solve this problem by establishing a connection with precise timing to tune the transmitter and receiver.

This tuning process occurs during the LTSSM recovery state, which allows the configuration of the data rate, symbol and bit lock and block alignment. In figure 2.11, there are many possibilities starting from the Recovery.RcvrLock [36]:

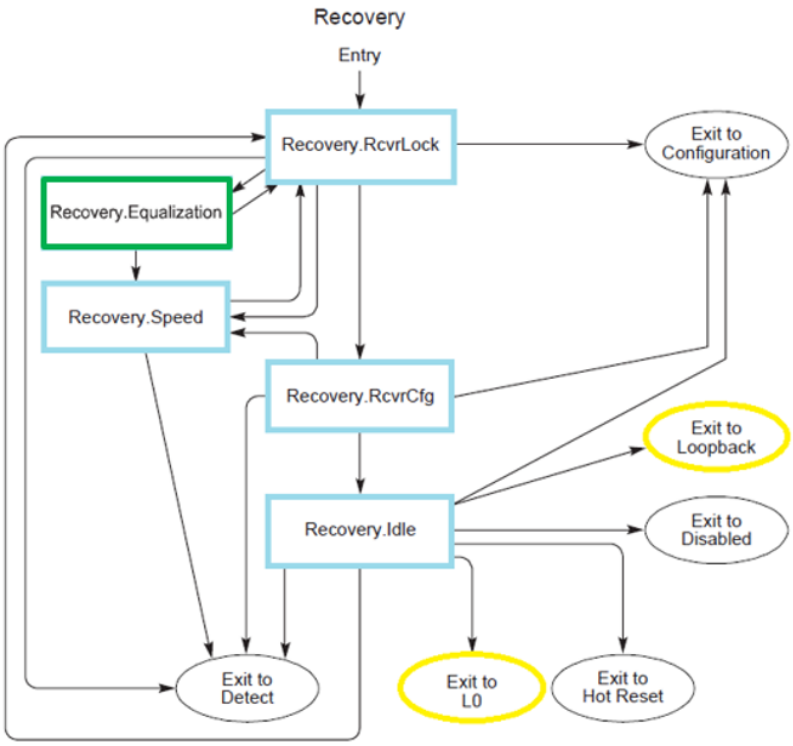


Figure 2.11: Recovery state negotiation [37]

- Recovery.RcvrCfg : Link is trained to work at 5 GT/s (Gigatransfer/second);
- Recovery.Speed : Link is unable to operate at speeds greater than 2.5 GT/s;
- Configuration : Core is receiving miss matched TS1 and TS2 sets;
- Detect : Link is lost.

From all the presented possibilities, the first option is the only positive outcome. This option trains the link for higher speeds by following the sequence:

1. Recovery.RcvrLock;
2. Recovery.RcvrCfg;
3. Recovery.Speed;
4. Exit to Detect.

In contrast, all the other options report an error or are not able to train speeds higher than 2.5 GT/s.

2.5.4 Ordered sets encoding

Amidst the initialisation of the link, Physical Layer Packets (PLP) get traded between neighbour devices.

These PLPs can also be called Ordered sets, and their purpose is to transport the initialisation details for the link that will get established.

During the equalisation process, there are five types of Ordered Sets used, as shown in figure 2.12:

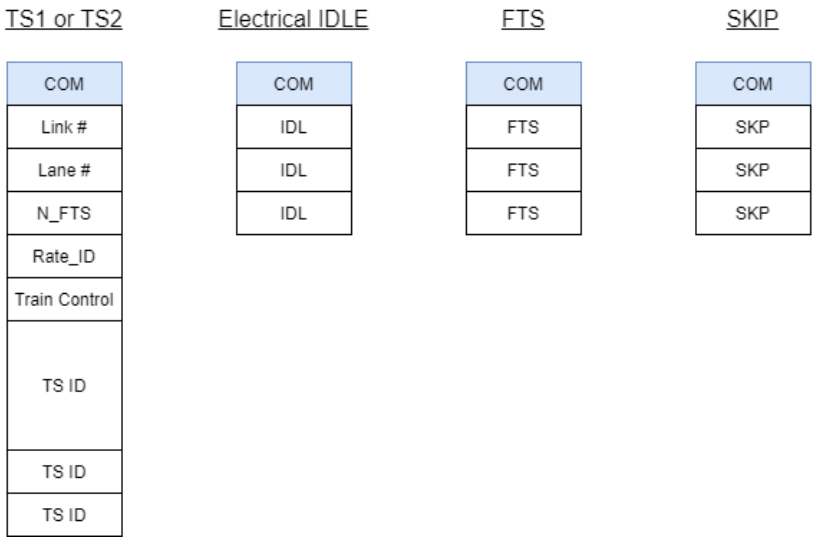


Figure 2.12: Ordered sets (Adapted from [29])

- Training Sequence 1 and 2 (TS1/TS2): Responsible for Link initialisation and training. Allow Link partners to achieve bit and symbol lock, set link speed and report variables status[34];
- Electrical Idle Ordered Set (EIOS): Set must be sent before entering the low-power link-state, Electrical Idle;

- Electrical Idle Exit Ordered Set (EIEOS): It is only used when the link is operating at a speed higher than 2.5 GT/s to ensure that Electrical Idle exit is detected;
- Skip Ordered Set (SOS): This set makes up for the difference in frequencies between the bit rates of two ends of a link;
- Fast Training Sequence (FTS): It is the mechanism used for bit and Symbol lock when transitioning from L0s to L0. The FTS is used by the Receiver to detect the exit from Electrical Idle and align the Receiver's bit and Symbol receive circuitry to the incoming data [36].

In a situation where there exist multiple channels available to a connection, a method called Byte Striping can apply, as figure 2.13 shows.

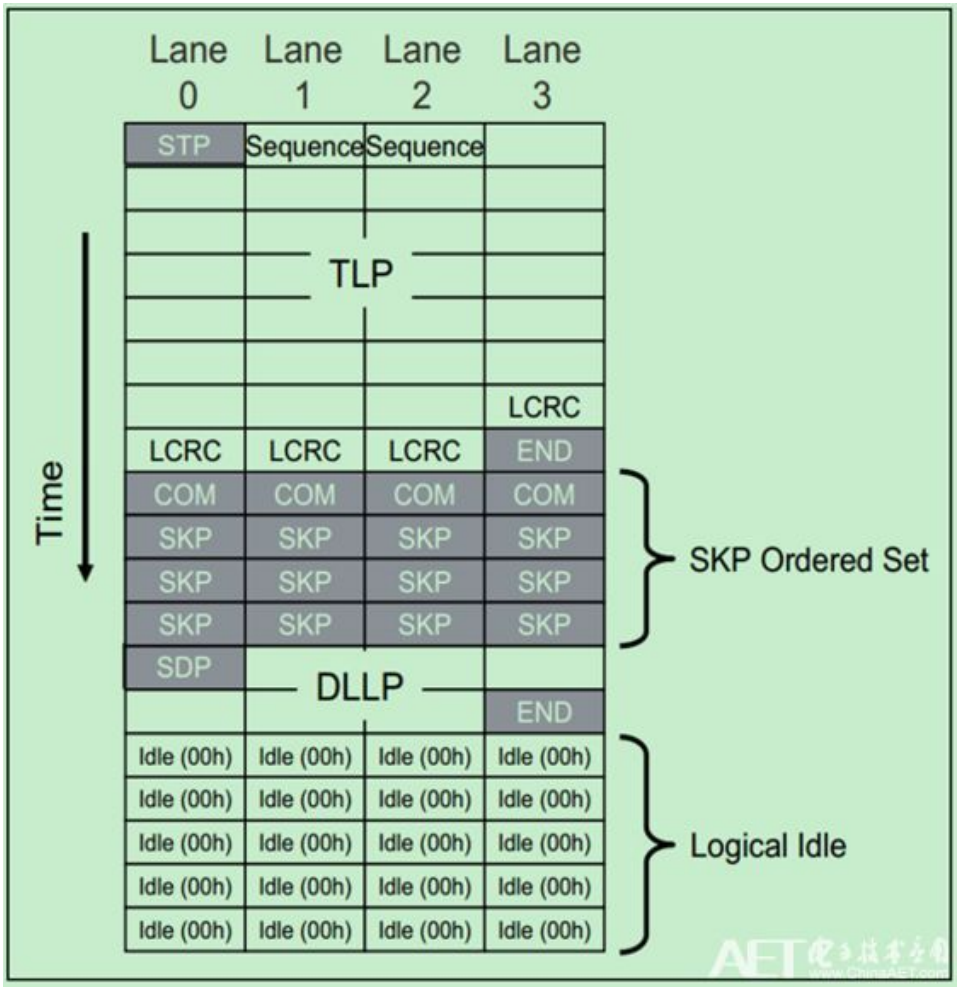


Figure 2.13: Byte Striping [38]

This method derives from a commonly used process which is data striping, and its main advantage is the ability to utilise many of the communication lanes by

dividing the message throughout the multiple ports and sending it simultaneously [39].

The scrambling process mixes TS1 and TS2 ordered sets so that the information maintains its synchronous behaviour and prevent repetitiveness from happening [40].

The method improves the transition density and power balance over long periods. Figure 2.14 shows an example of the scrambling process utilising multiple Linear-Feedback Shift Register (LFSR).

The final result appears as a group of bits numerically balanced between zeros and ones while also delivering the initial message.

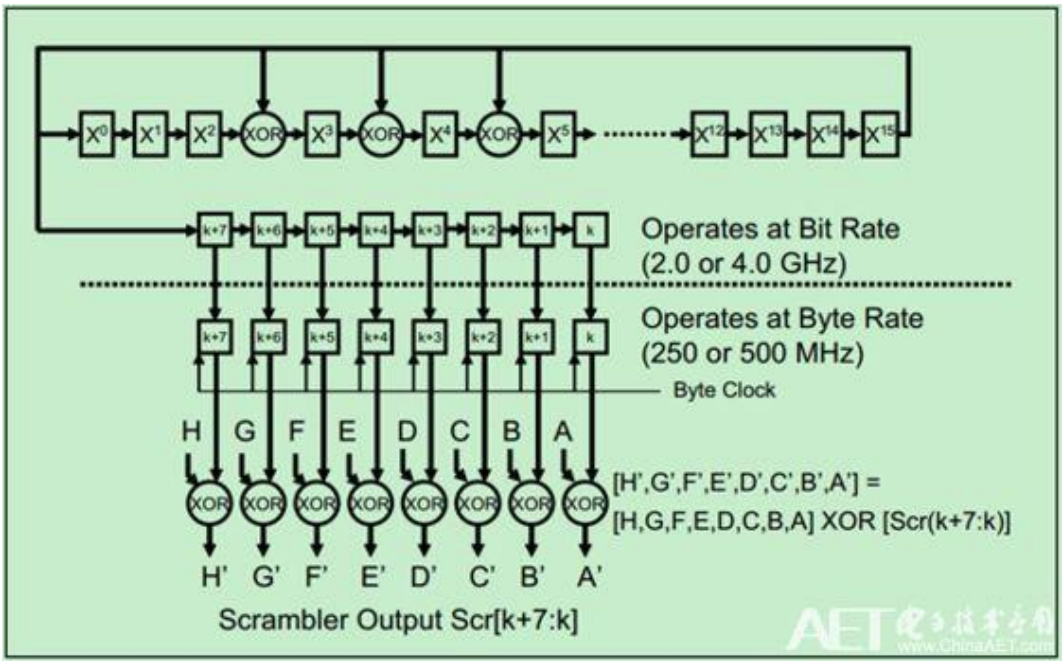


Figure 2.14: Scrambler [38]

The link and lane numbering gets defined through the TS1 and TS2 ordered sets, which numerically link an upstream device to a downstream device.

The numbering order gets sent by the upstream device to check if it corresponds to the downstream one. Usually, the devices are linked according to the matching number and sequentially.

However, the layout of some devices can be miss-matched. And to correct this problem, some downstream devices have a lane reversal functionality that alters the order to match the order defined by the upstream device, as presented by figure 2.15 [34].

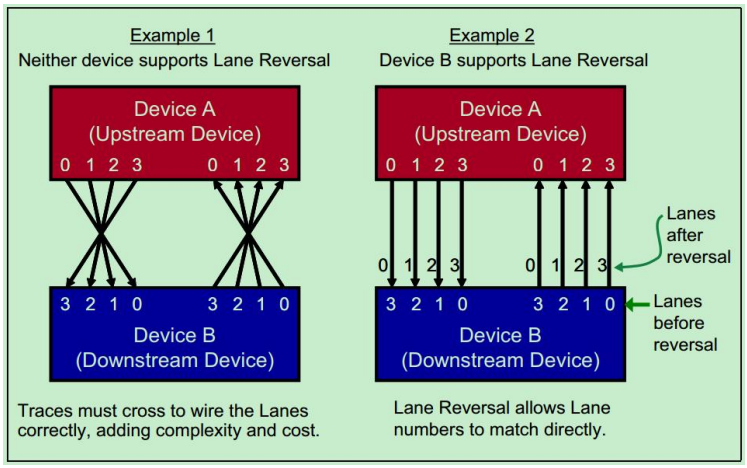


Figure 2.15: Lane reversal example [34]

2.6 Channel arbitration

Initialising the communication levels for the transactions of a PCIe device Traffic Classes (TC) must be defined to ensure the Quality of Service (QoS) required.

The communication in PCIe formally establishes through the connection of point-to-point devices with a switch. Therefore, exists arbitration which is performed in two steps, as shown in figure 2.16.

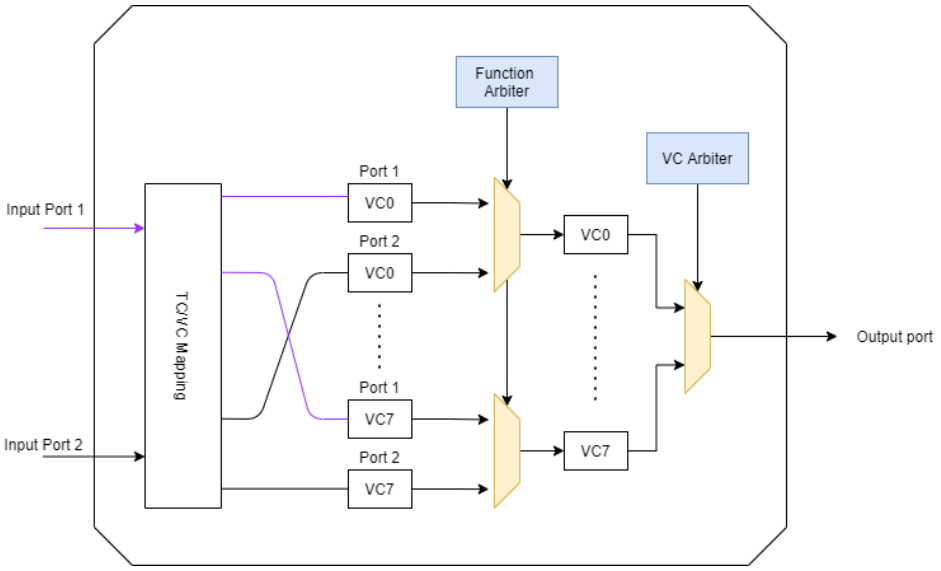


Figure 2.16: Switch arbitration structure (Adapted from [34])

Firstly, if transactions that come from the different input ports aim for the same virtual channel in the output port, then port arbitration must be processed before forwarding to the corresponding virtual channel.

On the other hand, in the virtual channel, the resources are limited because only one link is available to be shared by the many virtual channels.

Thus, the traffic must be arbitrated depending on the priority of the path, as shown in figure 2.17.

These channels can group into Low priority channels (VC0-VC3) and High priority channels (VC4-VC7).

While the upper group classify as a strict priority arbitration, mainly for isochronous traffic, the lower group is chosen when there are no packets on the upper group channels, so it is mostly to provide QoS [41].

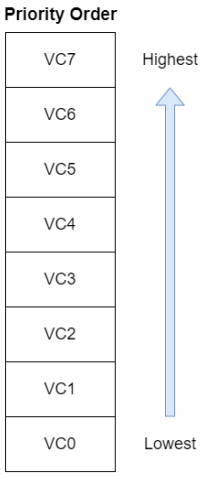


Figure 2.17: Virtual Channel Priority Order (Adapted from [34])

In conclusion, the basic knowledge to handle the Speed Adaptor gets delivered in this chapter.

This knowledge includes information regarding the devices that the connector will bridge and their main functionalities, along with the theory around the physical layer protocol, which is responsible for data transmission.

The next chapter will provide a basic structure of the Speed Adaptor and the requirements to work around with the device.

Chapter 3

Project Requirements

This chapter will address the overall architecture of the Speed Adaptor (SA) and the subsequent blocks that form this device.

In other words, the chapter explains the requirements for test usage of the SA, timing closure, restraints and variables. All of the information displayed in this chapter refers from the Synopsys SA data book.

3.1 Speed Adaptors

Prototypes are composed of many components such as FPGAs, a circuit board, custom IPs and additional peripheral devices. This mixed environment requires a system that enables data transfer at different speeds to occur.

Therefore, a device like a speed adaptor is needed to set up a communication medium that allows a slower running system to interact with high-speed external devices.

The purpose of the speed adaptor studied is to connect the PCIe Endpoint (EP) or PCIe Root Complex (RC) to the prototyping platform that operates at a much slower speed.

Thus, there is a need to control the high-speed data incoming from the PCIe bus to the low speed inherent in the destination environment. Figure 3.1 shows a simple representation of an SA architecture.

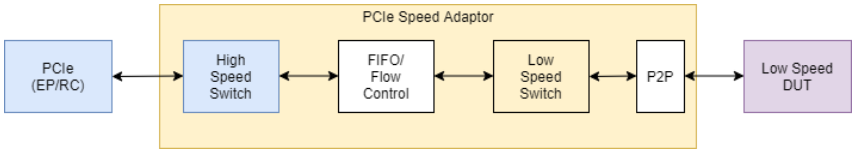


Figure 3.1: Basic SA architecture

3.2 Speed Adapter Architecture

Figure 3.3 shows the SA architecture that operates with two domains that run at different speeds.

The left side, represented in yellow, shows the high-speed side that connects to external high-speed PCIe devices like EPs and RCs. The right side, colourised in red, locates the low-speed side, which connects to the ZeBu emulation server that runs at lower frequencies due to its testing purposes.

As Zebu is an emulation platform, it enables the early bring-up and testing of hardware emulation. In other words, ZeBu attempts to mimic the end product to check for compatibility and performance issues that might arise.

One of the crucial pieces of this device is the Physical Interface for PCI Express Specification (PIPE), as they connect all of the internal components one to each other.

Using figure 3.2 as an example, each domain possesses a controller block that will function as the manager of the communication between devices so that the TLP packets can arrive without problems. This step is crucial since there are many PIPE speeds between the varied blocks.

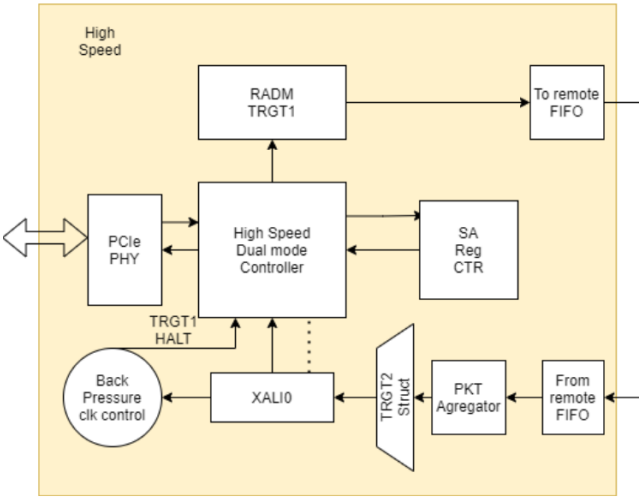


Figure 3.2: High Speed Domain

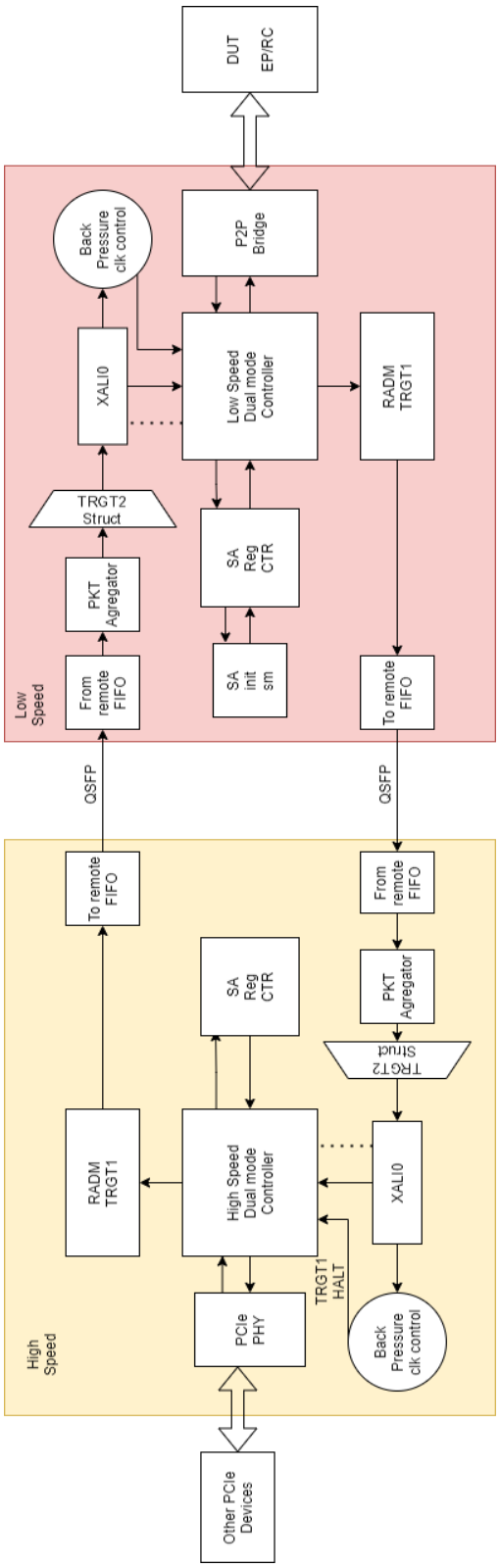


Figure 3.3: Transaction Layer

Furthermore, in both domains exists mechanisms of clock control to prevent packet overflow from the High-speed side to the Low-speed side. These mechanisms apply to the interfaces that connect to the existing FIFOs.

Previously, it was mentioned that the PCIe layers are formed from the three layers as shown in the figure 3.4.

However, in figure 3.4, these PCIe layers are grouped and named Common Express Port Logic (CXPL) to simplify the interpretation and the diagram.

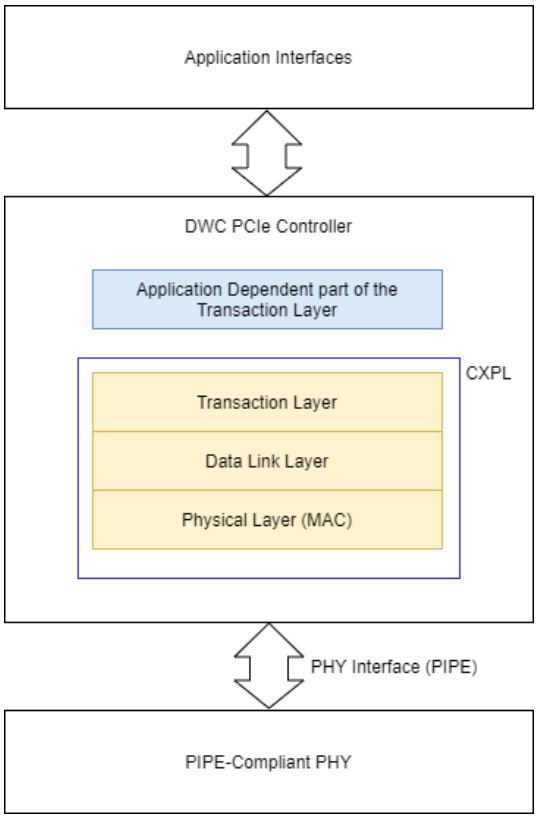


Figure 3.4: Low Speed Domain

Additionally, represented in the figure 3.5, there is the Application Dependent part of the transaction layer which will allocate space in the memory and forward the TLP packet to wherever it need to go.

These blocks which manage the packets are the:

- Receive Application-Dependent Module (RADM);
  - Sorts and filters received TLPs;
  - Forwards packets to controller's internal interfaces;
  - Buffering and queuing.
- Transmit Application-Dependent Module (XADM);

  - TLP arbitration and formation;
  - Flow control and credit checking.

- Configuration Dependent Module (CDM);
  - PCIe configuration space;
  - Port Logic Registers.

- Local Bus Controller (LBC).
  - Core Processing Unit (CPU) access, through a Data Bus Interface (DBI), to internal registers, located in CDM;
  - Access to external application registers through an External Local Bus Interface (ELBI) connection.

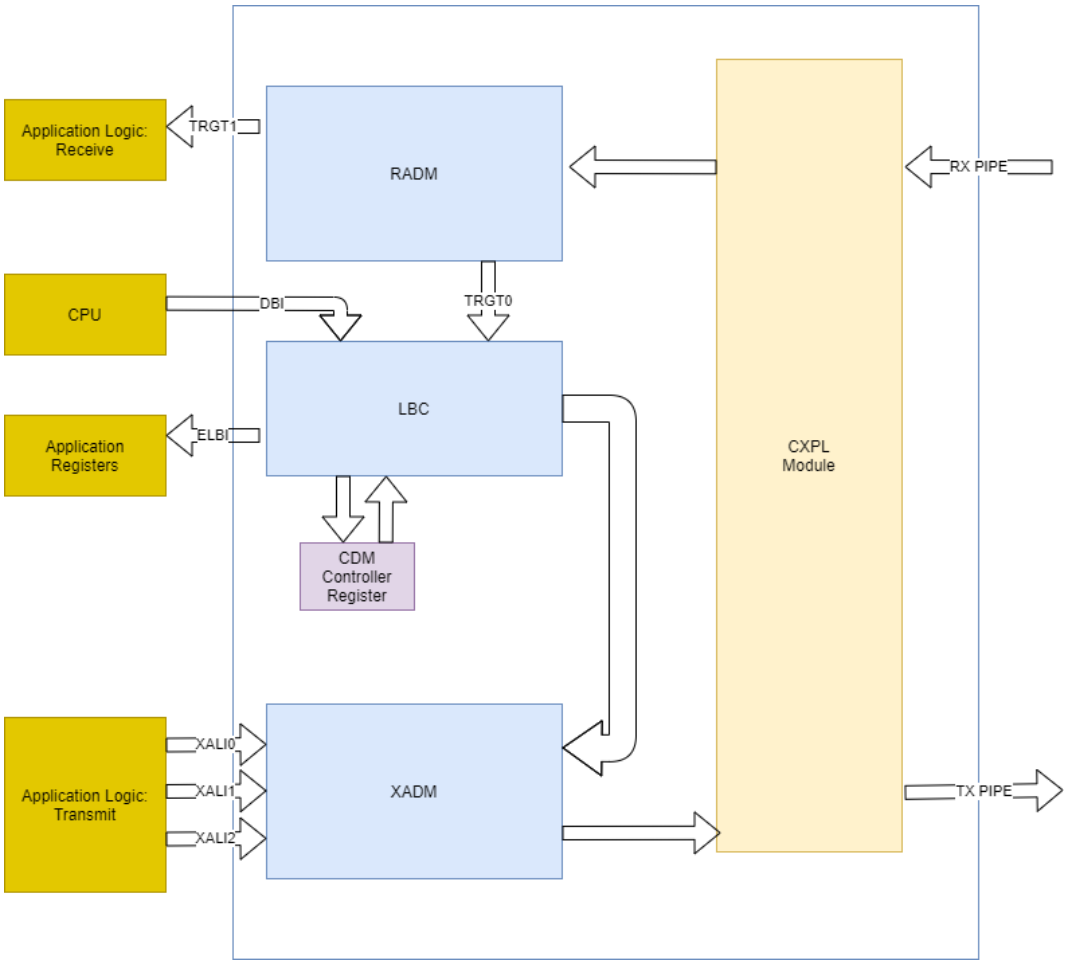


Figure 3.5: Transaction Layer

The interfaces and FIFOs have a complementary purpose as SA components, and their focus is the transportation and verification of TLP packets transferred between different clock domains.

These devices are mandatory as forms of packet control. Processing needs to exist so that the integrity of these packets is assured, preventing the loss of packets from overflow to occur.

Both sides include FIFOs that directly exchange packets from one side to the other, having a transmitter and receiver FIFO on both sides, adding up to a total of 4 blocks.

Furthermore, connecting the core components of the controller to the FIFOs exists the transmitter and receiver interfaces, TRGT and XALI. The TRGT interfaces mainly connect the RADM block to other internal blocks.

On the other hand, XALI interfaces can transmit different types of packets TLP.

For example, the interface XALI0 only for Posted packets, XALI1 for Non-Posted packets and XALI2 for Completion packets.

3.3 SA Link Speeds

As there are two speed domains for each part of the adapter, an analysis of the speed links for each side is represented in figure 3.7.

The speed values got obtained through the Discovery Visual Environment (DVE) simulator, which enables a representation of the signals in waveforms as shown in figure 3.6.

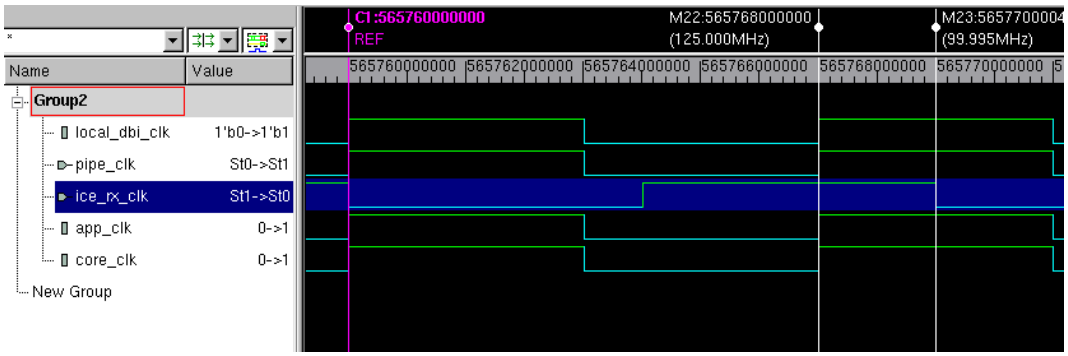


Figure 3.6: Waveform of clock speeds HS

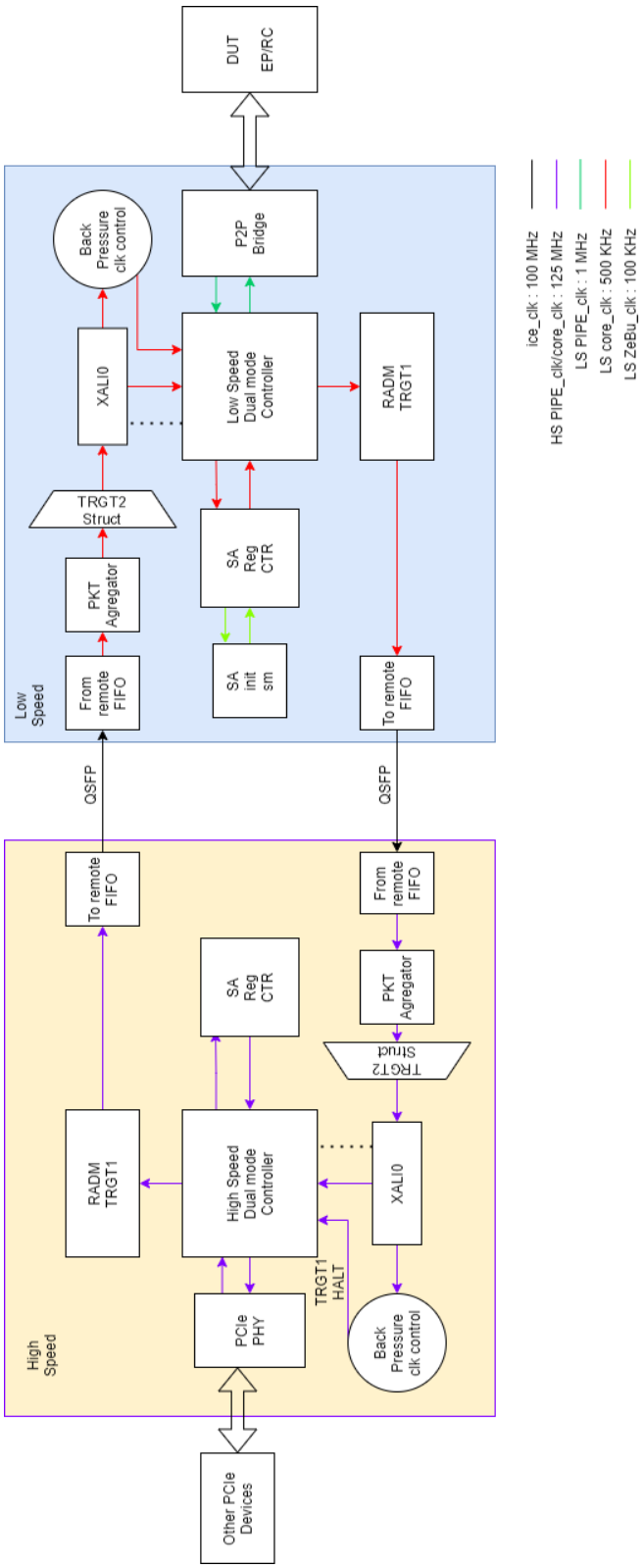


Figure 3.7: SA link speeds

While figure 3.6 shows the clock speed for the High-Speed side of the adaptor, the figure 3.8 displays the clock speed for the different Low-Speed components. Typically, the PIPE clock speed is identical to the core clock speed. However, depending on the generation of PCIe utilised, the PIPE clock speed can modify to achieve higher frequency.

Therefore, the core speed is 500 kHz, and the PIPE speed is 1 MHz. Nonetheless, figure 3.8 shows the clock for the ZeBu signal, which connects to a memory block that works at a slower clock speed. Due to image compression, figure 3.8 only represents half of the ZeBu signal.

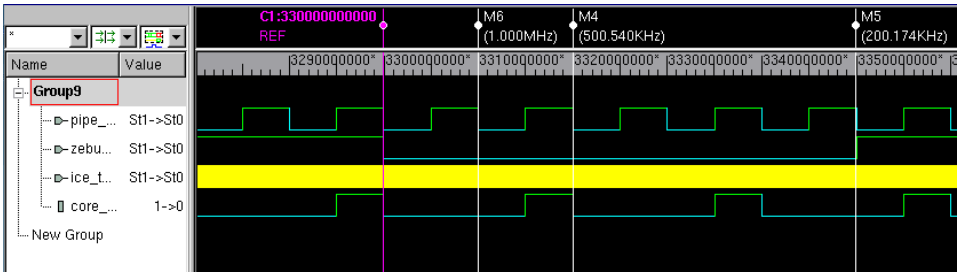
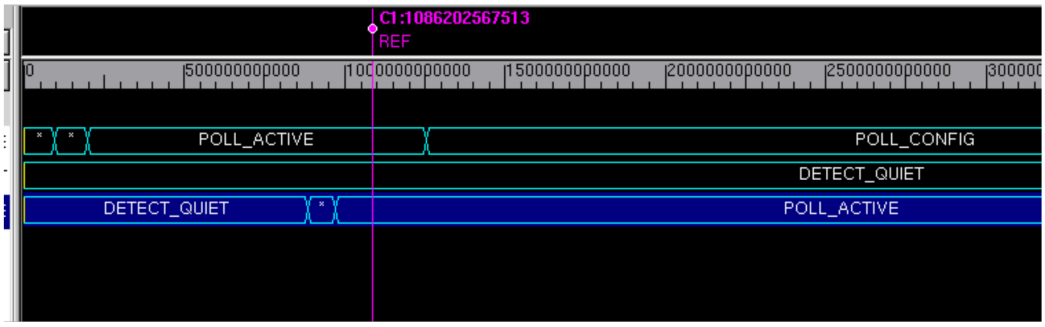


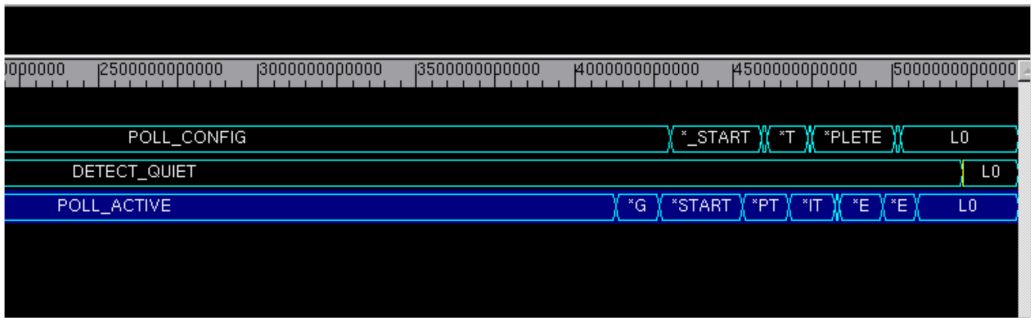
Figure 3.8: Waveform of clock speeds LS

During the initialisation of the PCIe links, the lower speed interfaces start sooner since they take longer to set up. In the set-up process, the link possesses the following states that will verify its integrity as demonstrated in figure 3.9:

1. Detect Quiet: Waits for a receiver;
2. Detect Act: Detects a receiver and prepares for polling;
3. Poll Active: Starts Polling;
4. Poll Config: Receiver and Transmitter exchange TS1 and TS2 ordered sets to achieve symbol and bit lock;
5. Config Linkwidth Start: Starts Configuration;
6. Config Linkwidth Accept: Upstream port establishes link width;
7. Config Lanenum Wait: Downstream port transmits TS1 ordered set with lane number;
8. Config Lanenum Accept: Upstream port accepts lane number;
9. Config Complete: Negotiated lanes get trained;
10. Config Idle: Checks lane integrity;
11. L0: Normal active state.



(a) Part1



(b) Part2

Figure 3.9: LTSSM initialisation

In this set-up phase, bit error rate, compliance pattern, cross talk, link and lane numbering get negotiated and defined, reaching the end goal, L0, a full-on power state where packet transmission and reception begins.

3.4 Backpressure/Clock Control

The difference in speed between the two settings is bound to create challenges in the context of a bidirectional data flow system.

The main obstacle is to transfer the information from one domain to another without the loss of packets or errors.

Hence, this system possesses the backpressure mechanism, which controls the flow in the interfaces to prevent overflow.

Specifically, the system overflows because of the speed difference at which data gets sent. It mainly happens when data is sent from the High-speed side to the Low-speed side.

Consequently, the FIFOs quickly get filled to their maximum capacity making the backpressure mechanism enter into action by halting the interface so that no packets are being sent further and getting lost, as there are no more available spots in the FIFOs.

The figure 3.10 represents the basic blocks that are involved in the halting process. As there are different types of packets, there exist separate FIFOs queues depending on the packet sent.

Differentiation is needed because packets have priority levels attributed to their type, meaning messages that cause more latency on the system get sent last, like the Non-Posted packets.

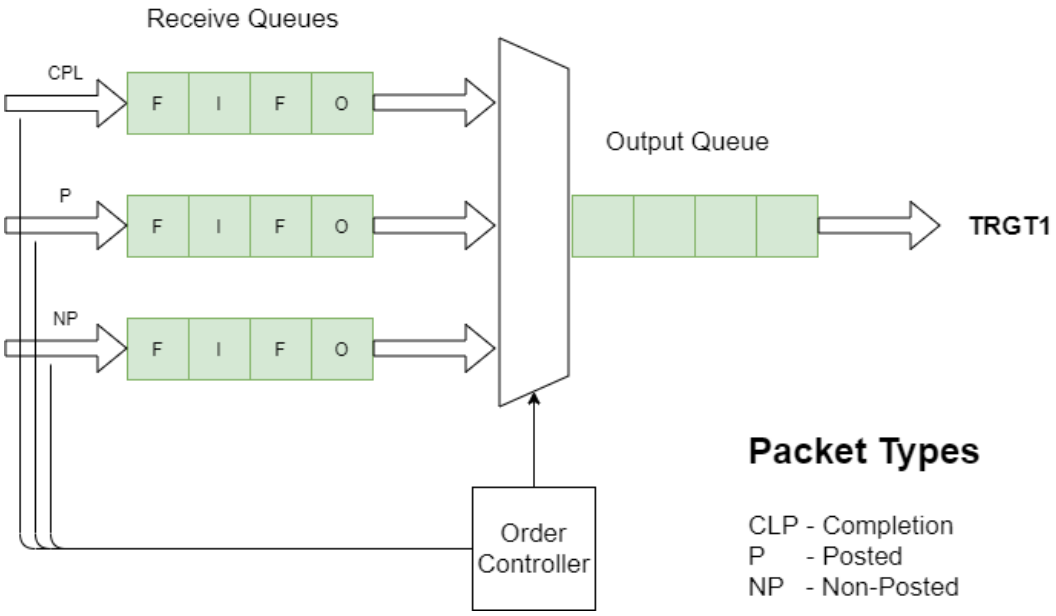


Figure 3.10: Halting mechanism

3.5 Timing report requirements

Static Timing Analysis (STA) is a validation method that occurs when synthesising that will test all of the design timing paths.

During its execution, the method will verify the presence of delays and timing violations of the constraints, generating a report with the possible outcomes, along with the shortest and longest path.

Figure 3.11 demonstrates the STA methodology, which takes an ordinary design with its inputs and outputs and fragments the path into several smaller, calculating the delay between the logic blocks inside the schematic.

While there is another alternative to verify the timing performance, for example, a dynamic simulation that uses the input of stimulus vectors as its execution method, STA is much faster due to the difference in the application process.

One downside of the STA is that it relies on the accuracy of the delay models making it hard to detect false paths [43].

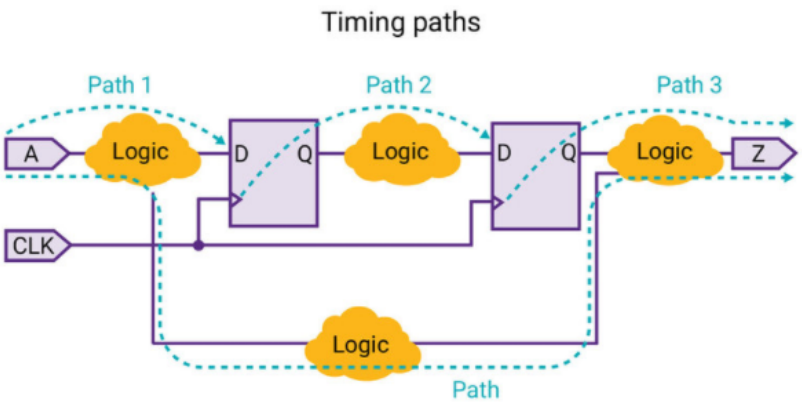


Figure 3.11: STA breakdown [42]

Table 3.1 shows the requested clocks for each domain of the speed adaptor. However, these values are not the deployed clock values.

Achieving the timing closure requires the guarantee of a positive time, so the actual value does not deviate from the requested value, consequently causing the malfunctioning of the system.

Table 3.1: Timing Request on Synthesis

Clock	**Requested Frequency (MHz)**
haps_clk	2.0
aux_clk	10.0
pinmux_clk	100.0
tunnel_clk	80.0
he_ss_tunnel_rx	10.0
clk_100k	0.2
sample_clk	20.0

Concluding, this chapter delivered the basic knowledge of the core components that compose the SA along with a summary of their functionalities.

Following the next chapter of the thesis, this last section explained the analysis method utilised and its timing requirements for the implementation of the SA to work with the development board.

Chapter 4

Proposed Solution

Chapter four presents a brief introduction of the Register Transfer Level (RTL) development flow step-by-step. Additionally, the procedures for the simulation get demonstrated through the respective Synopsys tools.

Finally a summary of the results gets analysed by utilising the path timings and area report.

4.1 RTL flow

In the development of a design, some steps must be followed to ensure the integrity of the final system, some of which the figure 4.1 shows.

Firstly, in the RTL steps, the specifications of the system parameters must get defined. For example, parameters such as clock frequency, strobe polarity, reset and port allocation.

Secondly, a draft of the high-level design should get drawn to outline the necessary modules and ports. Furthermore, an illustration of the low-level design helps to display the behaviour of the ports through an Finite State Machine (FSM).

Therefore, the code is generated accordingly to the design requirements. Then, a test bench gets generated to perform a functional simulation of the created code.

After these steps, the code is synthesised either after or before the partitioning. If the SoC design is not big enough, there is no need to include the partitioning step. The synthesis will transform the RTL code into a hardware schematic that will integrate with the board.

Synthesis gets divided into three steps: Translate, Map and Place and route. Translate transforms the Verilog code into a group of netlists and their respective connections. The Map stage adapts the information generated in the Translate step to fir with the development board. Place and route is the following step and consists of loading to the FPGA a netlist and user constraints to generate the design behaviour.

This step requires the proper setup of the constraints applied to the top-level ports, enabling it to function correctly with the FPGA chip.

Finally, the implementation of the RTL schematic performs additional timing tests and is programmed directly to the FPGA [44].

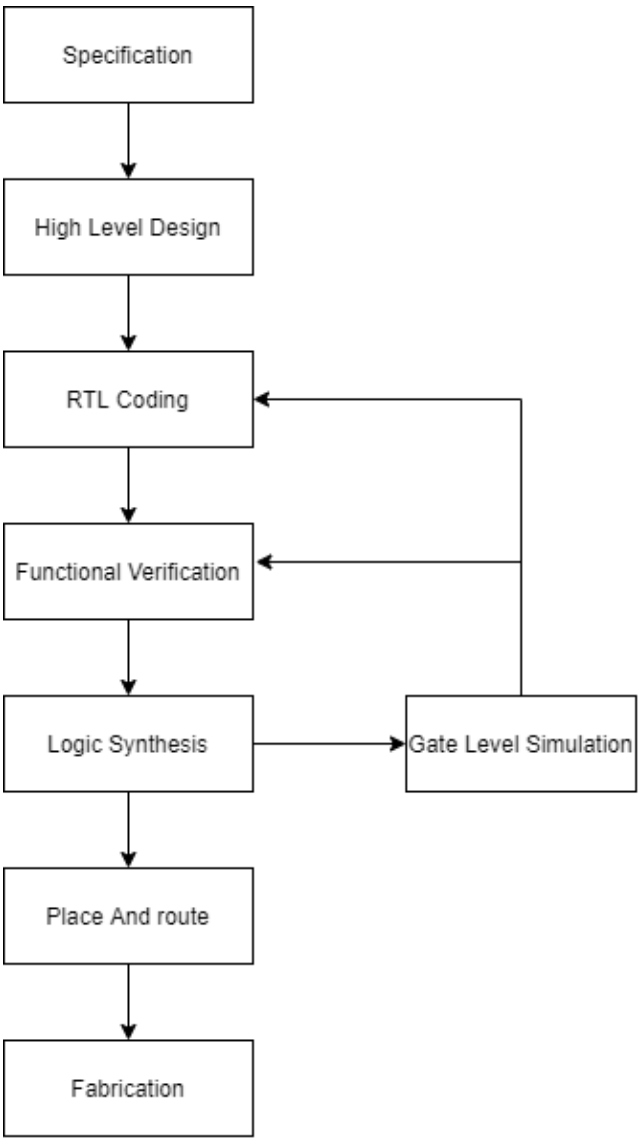


Figure 4.1: Design Flow (Adapted from [45])

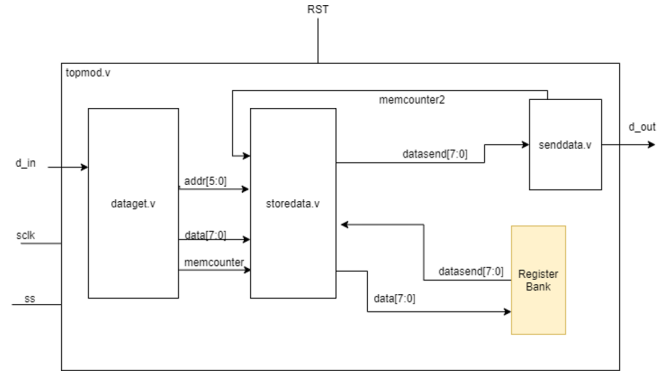
4.1.1 Code

As one of the primary HDL languages, Verilog is the coding language used by Synopsys to program RTL designs.

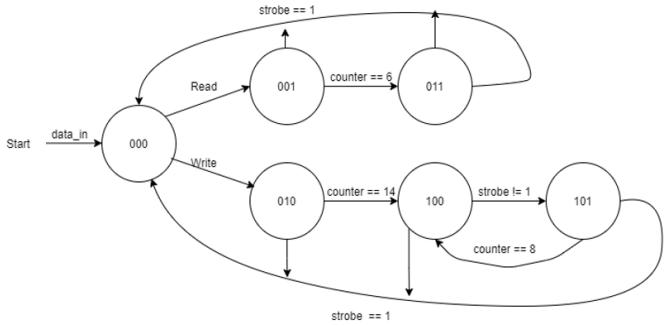
Writing RTL code translates to creating and manipulating hardware to fit our needs.

Furthermore, contrary to traditional programming languages, the code does not run sequentially. It runs in parallel, which means that hardware gets initialised and configured simultaneously.

The RTL code can be defined and interpreted through a high-level design, as a block diagram, or through a low-level design which can get expressed with an FSM, as figure 4.2 shows.



(a) Block Diagram



(b) FSM

Figure 4.2: RTL design interpretations

On one side, the block diagram gives a general view of the system main blocks, their respective roles and the ports associated with each.

On the other hand, an FSM exhibits the inner workings of the design through the interaction of the ports and a state machine which will dictate the output signal.

4.1.2 RTL Simulation/Testbench

Behavioural simulation is the first step to draft RTL code that validates the solution before the hard compiling.

Nonetheless, the RTL code must get modified to be synthesizable because behavioural is often not suited to adapt to hardware.

The simulator used in Synopsys is called the VCS and can be used through its interface called Discovery Visual Environment (DVE), as shown in figure 4.3.

This method is a quick simulation that is very helpful to debug the RTL code and make the needed changes to reach the set requirements or further optimise.

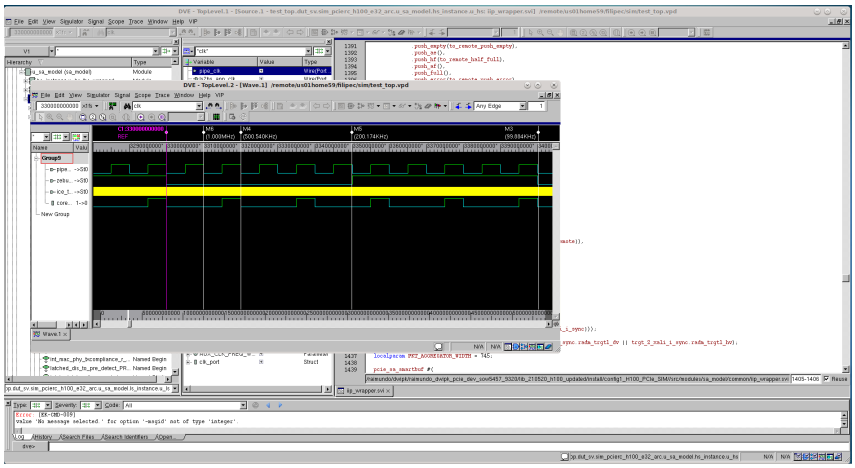


Figure 4.3: DVE interface

Additionally, the VCS simulator integrates an RTL code verification called Unified Report Generator (URG).

This report checks for incomplete "if" conditions, if all the Finite State Machines (FSM) are triggered at least once, and looks at unused registers or untoggled bits. Figure 4.4 shows an example of successful code coverage.

Total Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM
100.00	100.00	100.00	100.00	100.00

Hierarchical coverage data for top-level instances

SCORE	LINE	COND	TOGGLE	FSM	NAME
100.00	100.00	100.00	100.00	100.00	testbench

Total Module Definition Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM
100.00	100.00	100.00	100.00	100.00

Figure 4.4: RTL code Coverage

4.1.3 Synthesis

The Synthesis process involves converting RTL into a schematic of logic gates that will constitute the hardware representation of the RTL, as presented in figure 4.5.

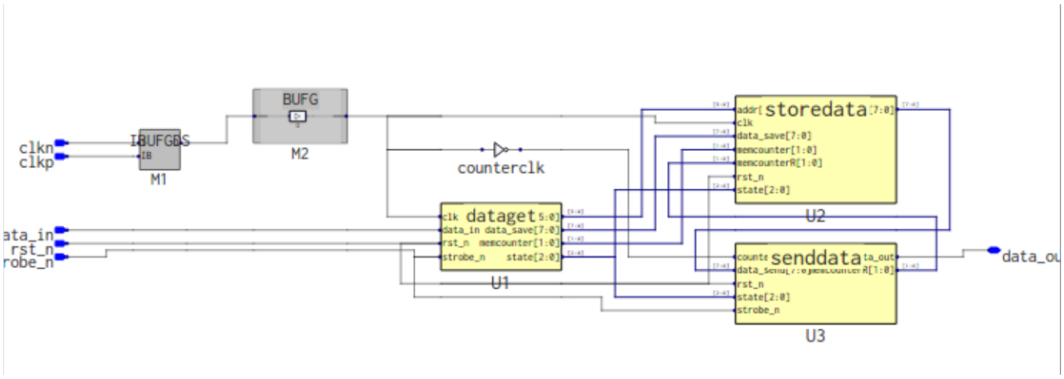


Figure 4.5: Synthesis Schematic

The conversion process requires a synthesis tool like the Design compiler made by Synopsys, which gets used as one of the primary choices in the industry. This process divides into three main implementation stages:

- Translate:

  Converts the existing RTL code into netlists which save as Native Generic Database (NGD) files.

  In turn, as presented in the figure 4.6, the file can define the constraints which equate to allocating the correct clock frequency, the type of power input used and respective pin in the FPGA prototyping board.

  This information typically gets stored in a User Constraints File (UCF) [46].

```
20 ###==== SCOPE_TAB: "Compile Points" (mouse over to show contents)
21 create_clock {p:clkn} -waveform {0 10.0} -period 20.0
22 create_clock {p:clkp} -waveform {10.0 20.0} -period 20.0
23
24 define_haps_io {p:clkn} -haps_io {A1_B[0]}
25 define_haps_io {p:clkp} -haps_io {A1_B[1]}
26 define_haps_io {p:rst_n} -haps_io {A1_C[2]}
27 define_haps_io {p:strobe_n} -haps_io {A1_A[2]}
28
29 define_io_standard {p:clkp} syn_pad_type {LVDS}
30 define_io_standard {p:clkn} syn_pad_type {LVDS}
31 define_io_standard {p:rst_n} syn_pad_type {LVCMOS18}
32 define_io_standard {p:strobe_n} syn_pad_type {LVCMOS18}
```

Figure 4.6: Synthesis constraints file

- Map:

  Outlines and optimises a circuit description into a LUT network to accommodate an FPGA architecture [47].

  In other words, this step compiles the logic stored in the NGD file to fit the FPGA elements and output a Native Circuit Description (NCD) file, which is the result of the physical representation of the mapping [46].

- Place and Route (PAR):

  The PAR process is responsible for routing and optimally connecting the different logic blocks. Additionally, adjusting their positioning so that the constraints do not get compromised.

4.2 Simulation and Synthesis Setup

In this section, the concepts from the RTL flow get applied using the tools provided by Synopsys.

These tools include an external machine that runs separate jobs whenever a user requires to use its resources. One advantage of this method is the scalability provided and the performance.

Additionally, the core tool setup enables the test of all the RTL flow procedures needed:

- VCS - Simulator;
- DVE - GUI for the simulator;
- Protocompiler - Synthesis GUI;
- Vivado - Place and Route;

4.2.1 coreConsultant

The coreConsultant is a GUI tool that enables an error-free design configuration for the synthesis and simulation.

Running this software necessitates the tool set up detailed above, which requires running the "isetup -s" command.

Additionally, corekit, a Synopsys tool package, needs to get built so coreConsultant can instantiate.

For example, figure 4.7 shows a workspace set up to generate the simulation through the order displayed.

After finishing the setup for the simulation and synthesis, the results get demonstrated through the dedicated tools.

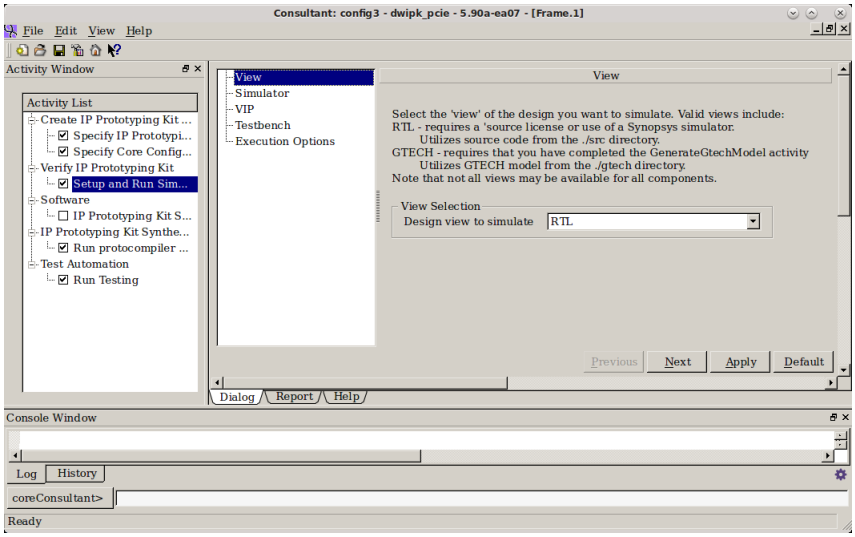


Figure 4.7: coreConsultant GUI

If the setup is successful, a simulation folder containing a VPD file will generate inside the workspace directory. This type of file enables the user to explore the waveforms and file hierarchy in the DVE interface.

Moreover, it can also check the respective ports and the Verilog code, consequently enabling the drawing of the speed adaptor diagram. An in-depth example gets depicted in the figure 4.8.

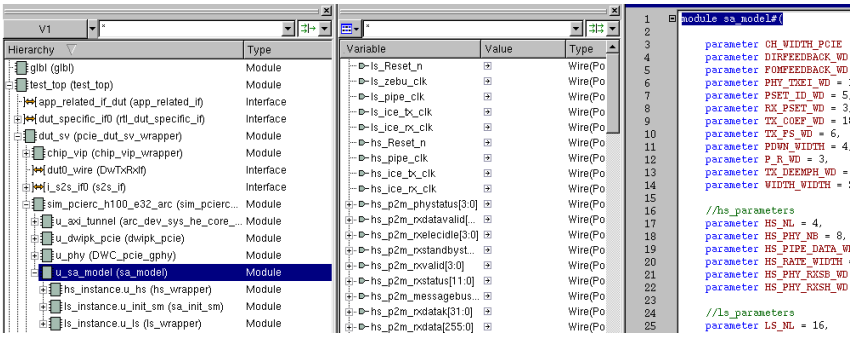


Figure 4.8: DVE GUI

4.2.2 Protocompiler

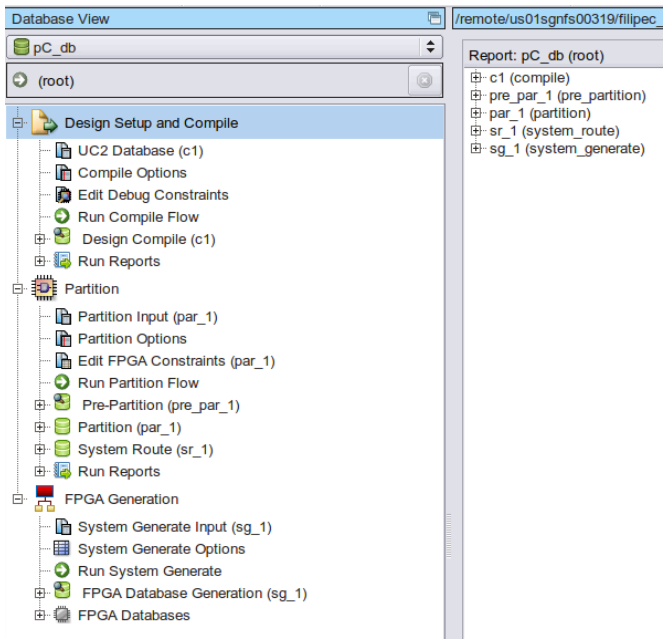
The synthesis process is a lengthy and resourceful one. So it takes the machine around one day to simulate and deliver the results.

All the results and images got taken from the Synopsys Synthesis tool, Protocompiler.

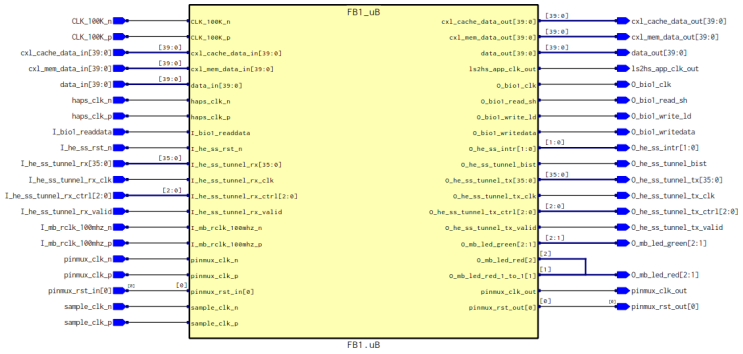
In this process, partitioning gets executed because of the simulation size we want to test, meaning processing divides through various FPGAs.

Therefore, as presented in the figure 4.9, the following steps are required to implement the speed adaptor with the development board:

1. Compile;
2. Pre-partition;
3. Partition;
4. System route;
5. System generate.



(a) Synthesis Setup



(b) System generated schematic

Figure 4.9: Partitioning process

The compile process attributes all the RTL code ports into physical FPGA ports.

Next, the pre-partition and partition phases fragment the code to accommodate the draft schematic to the many FPGAs inside the prototyping platform.

Then, to simplify and remove unused ports, the software runs through a system route, followed by the generation of the final version of the available ports on the platform for the synthesis.

After the System Generation phase, an FPGA database generates to build the synthesis files.

As last-mentioned, the synthesis process runs through the RTL compiler, followed by the pre-map and finally the final schematic map, as figure 4.10 shows.

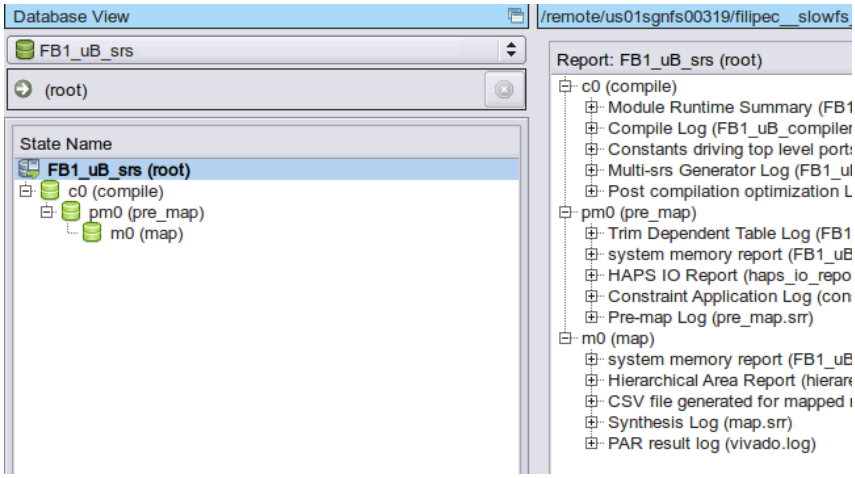


Figure 4.10: Synthesis of RTL

4.3 Timing closure and area limitations

As synthesis requires the transformation of the RTL code into a physical netlist, there must be hardware constraints, such as time and space limitations. Once the setup satisfies the constraints set, then the system will work as intended.

There must exist area restrictions because of the constricted space which the device can accommodate. Moreover, the area and the logic resources available are limited. Both these aspects can take a toll on the device's performance.

Timing closure is the adjustment of the basic logic gates to cater for the timing requirements defined. If these settings do not meet the sufficient requirements for the timing windows, the result can lead to failure on packet delivery.

4.3.1 Timing Report

As Protocompiler finishes running through its synthesis schedule, it will generate a summary of the timing delays and its worst cases which can affect the overall performance of the system, as table 4.1 shows.

Table 4.1: Worst Path Information

Path Information	**Values (ns)**
Requested Period	6.250
User constraint on ending point	1.500
Required time	4.750
Propagation time	**1.307**
Slack (critical)	**3.443**

The corresponding values of the worst slack get mitigated by modifying some configurations. For example, changing the synchronous nature of the clock into an asynchronous clock.

While this solution seems to cause problems, it will barely affect the global performance of the system because most of the timing closure problems stem from the presence of low-frequency clocks which delays the critical path of the system. In other words, devices that run at low frequency can afford to miss one or two cycles of the high clock speed domain to which they are connected, evading the timing issue.

However, these early values obtained from the Protocompiler might not need to be fixed because the Vivado software will try to correct and optimise the values generated from the Protocompiler, and the slack might not be there anymore.

Therefore, Vivado will go through multiples paths to find the worst and best timing case scenario from the multiple iterations, reaching the Worst Negative Slack (WNS) and Worst Hold Slack (WHS), which are required to study the timing closure issues.

If either of the slack values is negative, it means that the time constraints were not fulfilled. So the design might not work as intended, thus the need to correct the problem. The table 4.2 and 4.3 show the result of the Vivado optimisation for the previously mentioned variables in study.

Table 4.2: WNS Report

WNS Report	**Report Values**
Worst Negative Slack (WNS) (ns)	2.094
Total Negative Slack (TNS) (ns)	0.000
TNS Failing Endpoints	0
TNS Total Endpoints	1440239

Table 4.3: WHS Report

WHS Report	Report Values
Worst Hold Slack (WHS) (ns)	**-2.831**
Total Hold Slack (THS) (ns)	**-7092.301**
THS Failing Endpoints	24777
THS Total Endpoints	1440239

If either result is negative, the following steps are some solutions that can improve the delay present in the implementation [48]:

- Gate sizing: Different gate sizes translate into different inherent delays;
- Add buffer gates: A buffer gate can easily improve the delay of the circuit at the cost of more area occupation and power consumption;
- Netlist restructuring: Multiple IOs can put a gate under heavy load. Cloning a gate to split the capacitance is a solution sometimes implemented;
- Alternative algorithms: Some algorithms enable a better calculation of the critical path, resulting in paths with less delay.

However, the following methods can apply preemptively to ensure the delays get throughout minimised without resorting to secondary solutions:

- Specify good pin constraints;
- Apply proper RTL coding techniques;
- Apply global and path type timing constraints;
- Not over constraining;
- Set constraint period to the operation frequency of the system.

The solution, in this case, was changing the constraint of the Low-frequency side for it to work at High speeds. Although the device continues working at the lower speed, the system will perceive that part of the device working at higher speeds, not alerting any issues in the timing paths.

The solution is feasible because a device working at a higher frequency only results in faster performance. Yet, if the inverse would occur, the system might not run the task at the required time, inducing errors or delaying the system heavily.

Figure 4.11 shows the difference between setup and hold times. While setup time is the minimum required time for an input to be stable before the clock ticks, the hold time has the same conditions but after the clock edge.

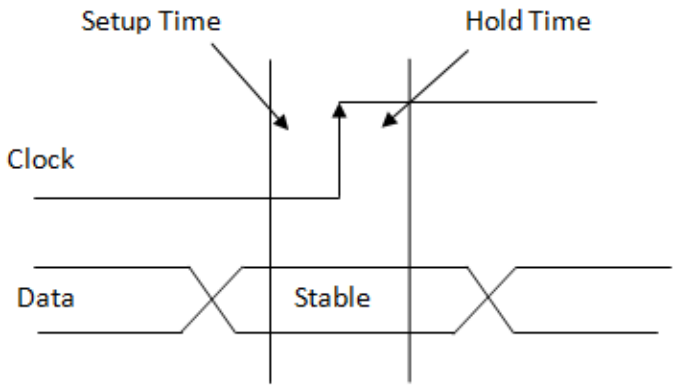


Figure 4.11: Setup and Hold time [49]

4.3.2 Area report

The technology Stacked Silicon Interconnect (SSI) was developed to employ a complex design consisting of multiple FPGAs. It combines the logic layers of the different FPGAs into a single larger one.

Consequently, the resources spread over different Super Logic Regions (SLRs) and the communication between these layers gets established through Super Long Lines (SSL) routes.

While utilising these routes might cause some delays in the overall performance of the system, this effect can be minimised by approaching the correct methodology, as shown in figure 4.12 [50, 51].

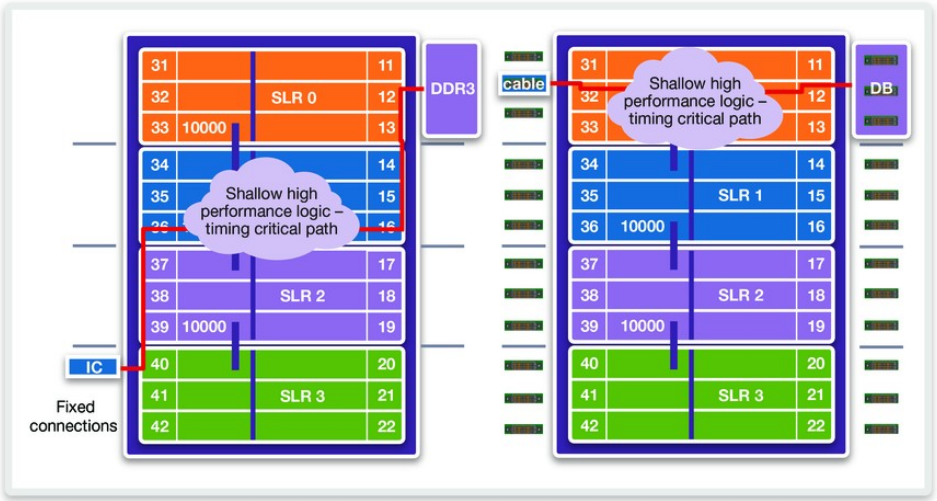


Figure 4.12: SLR methodologies [52]

The implementation method of the connectors can have an impact on the system performance. For instance, the utilisation of fixed connectors might lead to the dispersion of the pin banks through different SLRs, which might increase the overall latency of the system.

On the other hand, a flexible connector system dodges this problem by aggregating most of the IOs in one SLR, resulting in less routing. Additionally, this method has to utilise high-speed cables to compensate for the reduced routing, consequently having to plug an external device into the desired connector IO [52].

Table 4.4 demonstrates the allocated resources enabled by the FPGA to implement the RTL code.

Table 4.4: Resources Utilisation

Site Type	**Utilisation (%)**
CLB LUTs	13.21
CLB Registers	5.67
CARRY8	1.76
F7 Muxex	1.5
F8 Muxex	1.49

While table 4.4 presents the overall load of the components on the system, table 4.5 further specifies the utilisation of these components on each Super Logic Region (SLR).

Table 4.5: Super Logic Region Utilisation

Site Type	**SLR0 (%)**	**SLR1 (%)**	**SLR2 (%)**	**SLR3 (%)**
CLB	0.26	4.25	21.05	79.77
CLB LUTs	0.08	2.09	12.79	37.90
CLB Registers	0.12	1.44	3.89	17.23
CARRY8	0.01	0.21	3.55	3.25
F7 Muxex	0	0.03	0.26	5.70
F8 Muxex	0	0	0.26	5.69
Block RAM Tile	0.37	9.72	2.04	9.54
URAM	0	0	22.50	3.75
DSPs	0	0	0.10	0.73
Unique Control Sets	0.04	0.30	0.97	1.98

In figure 4.13 a visual representation of the resource utilisation is shown. This result gets generated through the Design Check Point (DCP) file, which Vivado generates after optimising the synthesis result.

Chapter four showed an introduction and explanation of how the system should work and behave. Therefore, the following chapter details the available methodologies to solve the timing slacks present in the system and an introduction to the equipment utilised on an emulation.

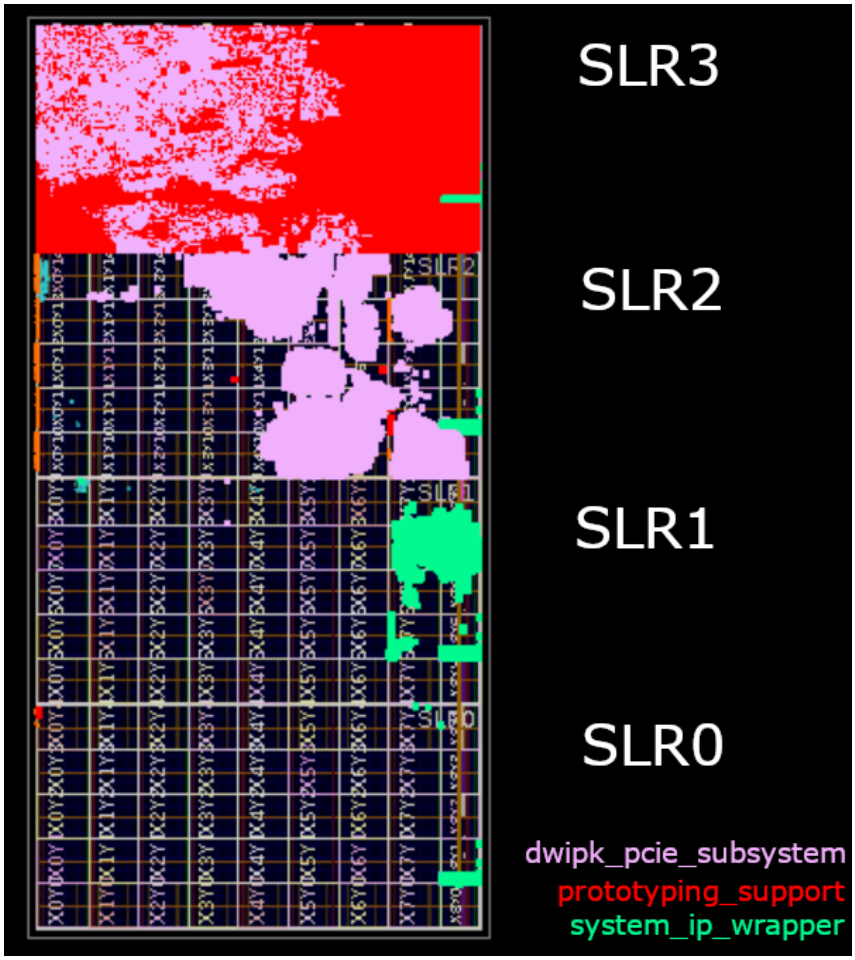


Figure 4.13: DCP resource utilisation graph

Chapter 5

Implementation setup and simulation analysis

Chapter five explores some available methods to improve and fix the timing closure issue. In other words, chapter four presented the results of the place and route for the timing analysis, and the current chapter demonstrates the methods attempted to correct the negative Worst Hold Slack (WHS) obtained.

Additionally, the chapter explains the setup of the hardware constituents and the overall architecture of the system.

All the information present got cited from the Designware PCIe install document from Synopsys.

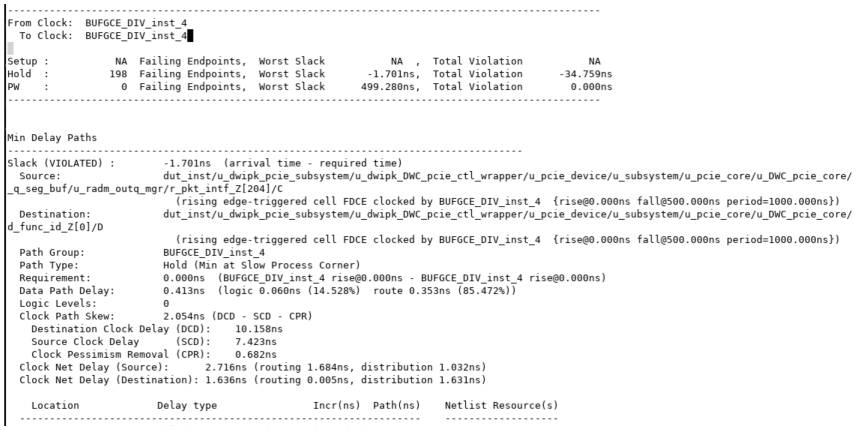
5.1 Timing closure methods

In the previous chapter, the timing report had a negative Worst Hold Slack value, meaning the invalidation of the result.

Figure 5.1 shows an example of a hold violation report.

Consequently, there were some attempts to fix the issue by:

- Modifying the time constraints to shorten the speed difference between domains and lower the delay present in the report;
- Utilising floorplanning to change the IO layout of the clocks to shorten the paths;



```
----------------------------------------------------------------------------------------
From Clock:  BUFGCE_DIV_inst_4
  To Clock:  BUFGCE_DIV_inst_4

Setup :           NA  Failing Endpoints,  Worst Slack           NA  ,  Total Violation           NA
Hold  :          198  Failing Endpoints,  Worst Slack       -1.701ns,  Total Violation      -34.759ns
PW    :            0  Failing Endpoints,  Worst Slack      499.280ns,  Total Violation        0.000ns
----------------------------------------------------------------------------------------


Min Delay Paths
--------------------------------------------------------------------------------------
Slack (VIOLATED) :        -1.701ns  (arrival time - required time)
  Source:                 dut_inst/u_dwipk_pcie_subsystem/u_dwipk_DWC_pcie_ctl_wrapper/u_pcie_device/u_subsystem/u_pcie_core/u_DWC_pcie_core/
_q_seg_buf/u_radm_outq_mgr/r_pkt_intf_Z[204]/C
                          (rising edge-triggered cell FDCE clocked by BUFGCE_DIV_inst_4  {rise@0.000ns fall@500.000ns period=1000.000ns})
  Destination:            dut_inst/u_dwipk_pcie_subsystem/u_dwipk_DWC_pcie_ctl_wrapper/u_pcie_device/u_subsystem/u_pcie_core/u_DWC_pcie_core/
d_func_id_Z[0]/D
                          (rising edge-triggered cell FDCE clocked by BUFGCE_DIV_inst_4  {rise@0.000ns fall@500.000ns period=1000.000ns})
  Path Group:             BUFGCE_DIV_inst_4
  Path Type:              Hold (Min at Slow Process Corner)
  Requirement:            0.000ns  (BUFGCE_DIV_inst_4 rise@0.000ns - BUFGCE_DIV_inst_4 rise@0.000ns)
  Data Path Delay:        0.413ns  (logic 0.060ns (14.528%)  route 0.353ns (85.472%))
  Logic Levels:           0
  Clock Path Skew:        2.054ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    10.158ns
    Source Clock Delay      (SCD):    7.423ns
    Clock Pessimism Removal (CPR):    0.682ns
  Clock Net Delay (Source):      2.716ns (routing 1.684ns, distribution 1.032ns)
  Clock Net Delay (Destination): 1.636ns (routing 0.005ns, distribution 1.631ns)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
```

Figure 5.1: Hold violation report

- Removing and changing clocks placement on the architecture.

5.1.1 Constraints

Constraints help the user to achieve the targeted outcome by influencing the synthesising tools to follow specific requirements.

At the Synthesis stage, the constraints are located in a .fdc file. However, after finishing the flow and reading the timing reports, running the Synthesis process all over again can be very time consuming.

Therefore, only the implementation process needs to run, and the implementation constraints can get modified in the .xdc file.

Figure 5.2 shows an example of the creation of clocks and their changeable parameters.

```
#2143 : create_clock -name haps_clk_0 -period 500.0 [get_ports haps_clk_p]
# line 2 in /slowfs/us01dwt2p856/filipec/dwipk2/dwipk_pcie_dev_sow5457_2740/lib/install
ime.fdc

create_clock -period 100.000 -name haps_clk_0 [get_ports haps_clk_p]

#2144 : create_clock -name pinmux_clk_0 -period 10.0 [get_ports pinmux_clk_p]
# line 3 in /slowfs/us01dwt2p856/filipec/dwipk2/dwipk_pcie_dev_sow5457_2740/lib/install
ime.fdc

create_clock -period 10.000 -name pinmux_clk_0 [get_ports pinmux_clk_p]

#2145 : create_clock -name sample_clk_0 -period 50.0 [get_ports sample_clk_p]
# line 4 in /slowfs/us01dwt2p856/filipec/dwipk2/dwipk_pcie_dev_sow5457_2740/lib/install
ime.fdc

create_clock -period 50.000 -name sample_clk_0 [get_ports sample_clk_p]

#2146 : create_clock -name clk_100k_0 -period 5000.0 [get_ports CLK_100K_p]
# line 5 in /slowfs/us01dwt2p856/filipec/dwipk2/dwipk_pcie_dev_sow5457_2740/lib/install
ime.fdc

create_clock -period 5000.000 -name clk_100k_0 [get_ports CLK_100K_p]

#2147 : create_clock -name I_mb_rclk_100mhz p:I_mb_rclk_100mhz_p -period 10
# line 7 in /slowfs/us01dwt2p856/filipec/dwipk2/dwipk_pcie_dev_sow5457_2740/lib/install
ime.fdc

create_clock -period 10.000 -name I_mb_rclk_100mhz [get_ports I_mb_rclk_100mhz_p]
```

Figure 5.2: Clock constraints (.xdc)

Although there was improvements on the WHS value, the result was a minimal decrease of value, making this solution not apropriate for the problem.

5.1.2 Floorplan

Floorplanning involves gathering information of FPGA components such as size and location to try and optimise the main functional block on their paths, delays and area utilisation. The blocks created from floorplanning are often called Pblocks.

Figure 5.3 demonstrates the Vivado GUI that enables the tampering of signals to place them all on the same SLR block, thus avoiding SLR crossing, which might introduce undesirable delays on the signal path.

Vivado will then generate the respective constraints to put the selected signals in the chosen SLR.

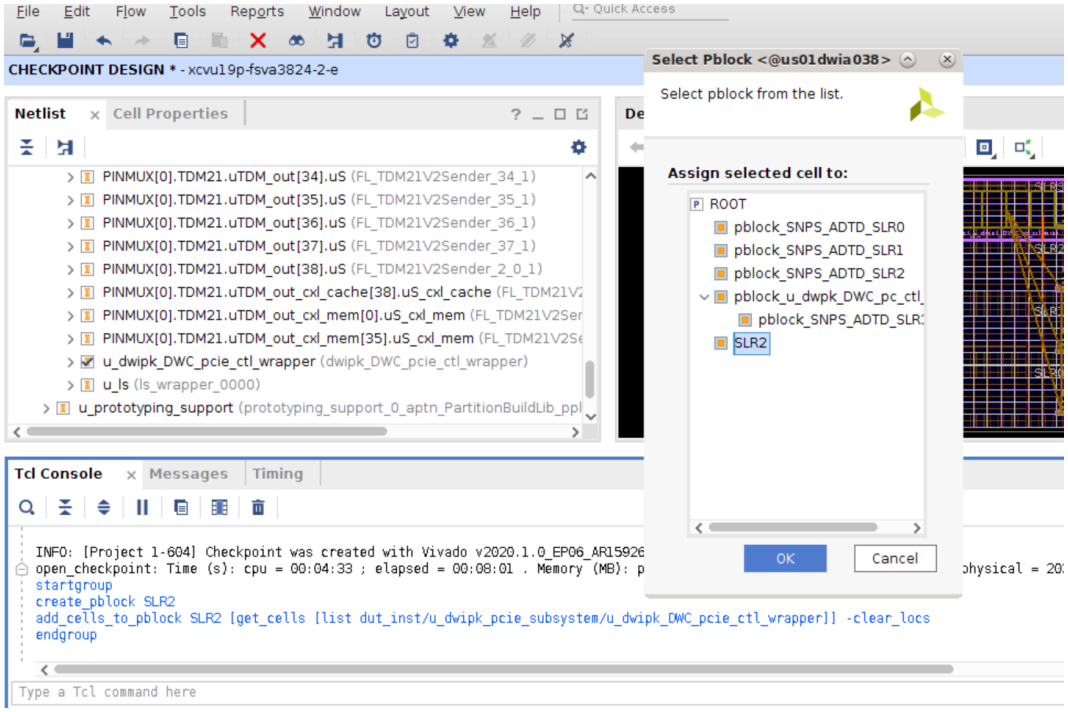


Figure 5.3: Pblock assignment

Table 5.1 shows the IO bank associated with each of the signals that go through the SLRs.

Contrary to the previous method, with this solution, the decrease of the negative WHS was substantial. However, even after allocating all of the clock IO banks to the same SLR, the result (-2.112) was still far from reaching the neutral/positive value.

Table 5.1: IO banks

Signal Name	IO Bank	SLR
I_he_ss_tunnel_rx	74	3
O_he_ss_tunnel_tx	75	3
cxl_cache_data_in	32	2
cxl_cache_data_out	31	2
cxl_mem_data_in	30	2
cxl_mem_data_out	29	2
haps_clk	69	2
sample_clk_p_o	69	2
CLK_100K_p	70	2
I_mb_rclk_100mhz_p	70	2
Pinmux_rst_out	72	2
ls2hs_app_clk_out	72	2
data_out	33	2
data_in	72	2

5.1.3 Clock removal/alteration

As the previous attempts were not successful, another approach got taken. Verifying the timing reports, the paths are traced back to delays caused by the clk_rst clock, which is present in another block domain from the PCIe subsystem block.

Consequently, this fact led to the discovery that two of the clocks generated by the clk_rst were begin utilised by the PCIe subsystem block, causing delays because of the path these two clocks had to take.

Figure 5.4 demonstrates how originally was configured and how the problem got solved.

The solution to this problem required removing the aux and axi clock generated by the clk_rst because they were causing too many path delays from being shared one block to another.

Moreover, to replace the removal of those clocks, pclk, which has a frequency equal or higher to those clocks and a better path, replaced them.

Table 5.2 shows the result that allows for a successful timing report generated after applying the fix.

Table 5.2: WHS fixed report

WHS Report	Report Values
Worst Hold Slack (WHS) (ns)	0.010
Total Hold Slack (THS) (ns)	0.000
THS Failing Endpoints	0
THS Total Endpoints	2499882

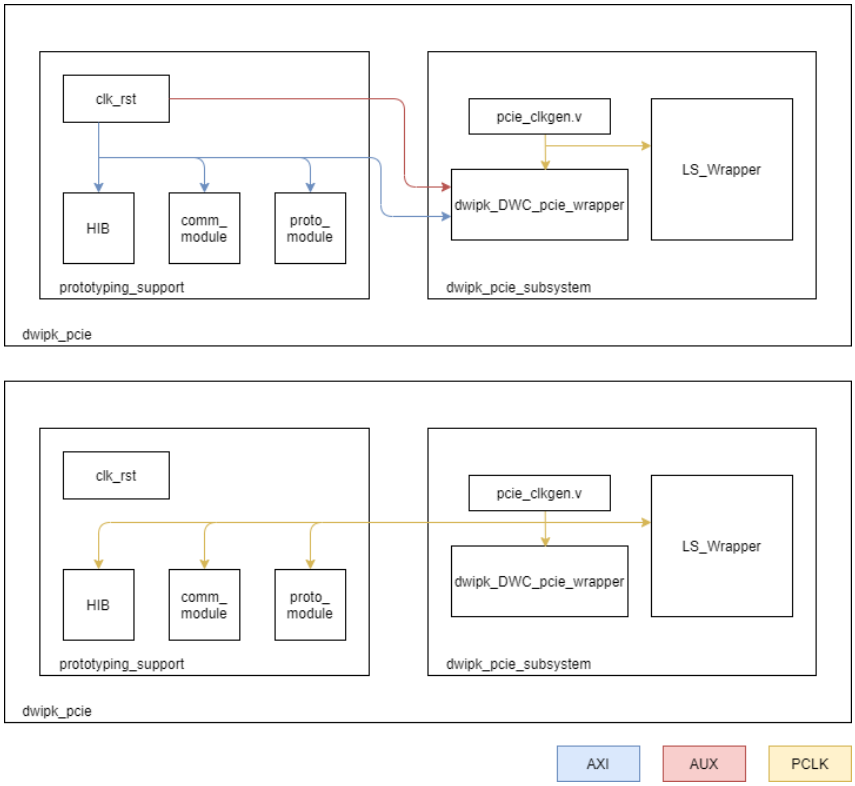


Figure 5.4: Clock modifications

5.2 Hardware PCIe Setup with HAPS-100

The present section details the prototyping process, from explaining the hardware components needed and the respective setup procedure.

The hardware setup follows the schematic shown in the figure 5.5.

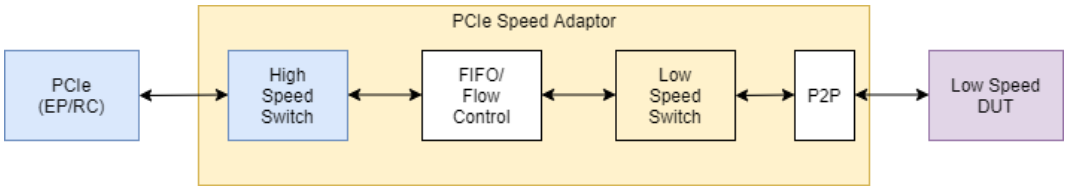


Figure 5.5: Basic SA architecture (Duplicate)

Table 5.3 lists all of the utilised pieces for the SA testing.

Figure 5.6 shows a visual representation of the materials present in the previous table.

Setting up the configurations requires following the next steps:

1. Connect HAPS-100 and ARC System using HT3 cables;

2. Place the SD card containing the Linux image in the ARC SDP system board;

Table 5.3: Hardware components

Item	Description
ARC AXS101 SDP	DesignWare ARC AXS 101 Software Development Platform
HAPS-100 4F	High Performance ASIC Prototyping System HAPS-100
Speed Adaptor	Synopsys PCIe Speed Adaptor Solution
CON_CABLE_25_HT3	HT3 cable
CLX/PCIe Backplane Board	CXL/PCIe Root Complex - Endpoint Backplane
PCIe iPass cable (X4 to X4) 1m	PCIe iPass cable to connect Speed Adaptor and PCIe Backplane board
Onestop PCIe Card	Adapter card between CXL iPass cable and CXL Backplane boardb

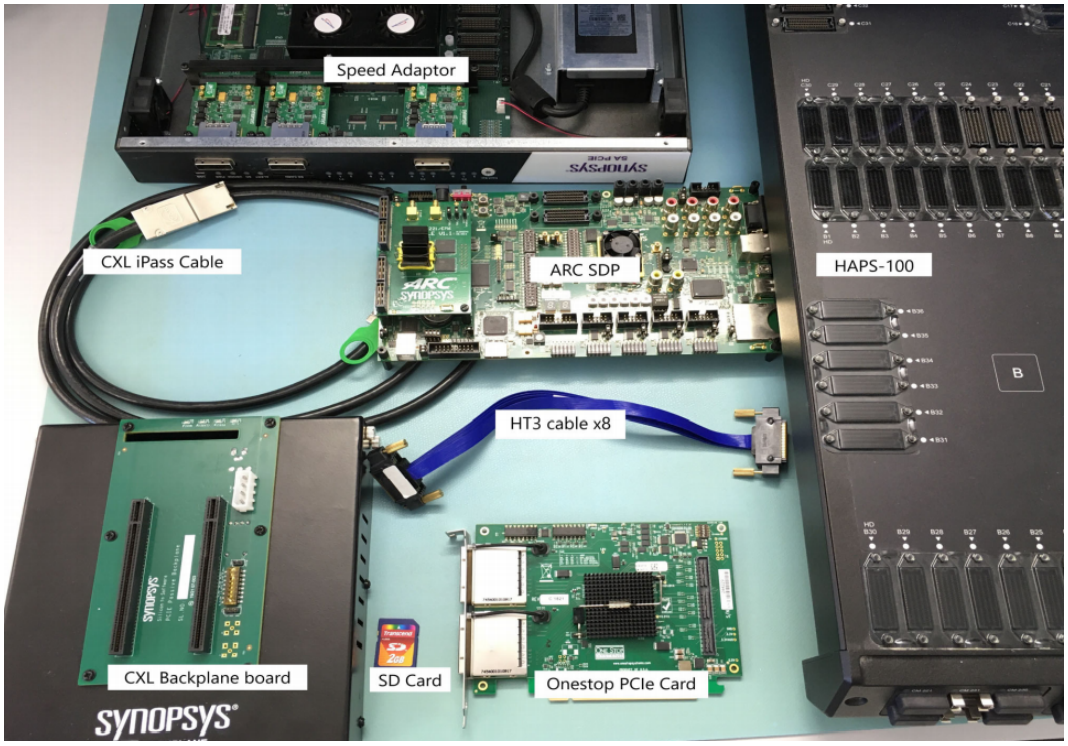


Figure 5.6: SA hardware configuration

3. Connect the USB cable to the ARC SDP and the computer;
4. Set up the HAPS-100 and the Speed Adaptor by connecting them with HT3 cables;
5. Connect one end of the PCIe iPass cable to the PCIe port one of the Speed Adaptor;
6. Connect the other end to the PCIe Backplane board;
7. Insert the PCIe Endpoint in the PCIe Backplane slot reserved for it;
8. Complete the power-on sequence.

5.2.1 Configure Host PC

Configuring the devices requires a host computer that will let the user program the bit files and set up the start-up sequences for the HAPS controller and the Speed Adaptor.

In this section, the power sequence and bit file programming is achieved through the following steps:

1. Invoque a terminal for Speed Adaptor

```
$ tcsh
$ cd /home/synopsys/<user>/MSFT/Example/PCI_HOST/SA2_GUI/Test
$ setenv SA2_INSTALL_ROOT $PWD
$ source SA2-GUI/settings.csh
$ cd <workspace_with_sa_bitfile>
```

2. Open another terminal for HAPS-100

```
$ tcsh
$ cd <workspace_with_haps_targetsystem>
```

3. Power-on Backplane (Press RST button)
4. Power-on Speed Adaptor
5. Configure Speed Adaptor

```
$ sa2_gui &
```

Figure 5.7 shows on the GUI the following steps:

5.1. Select SA

5.2. Select project.conf file

5.3. Click 'Configure' button

5.4. Check that Backplane's #RESET LED is off

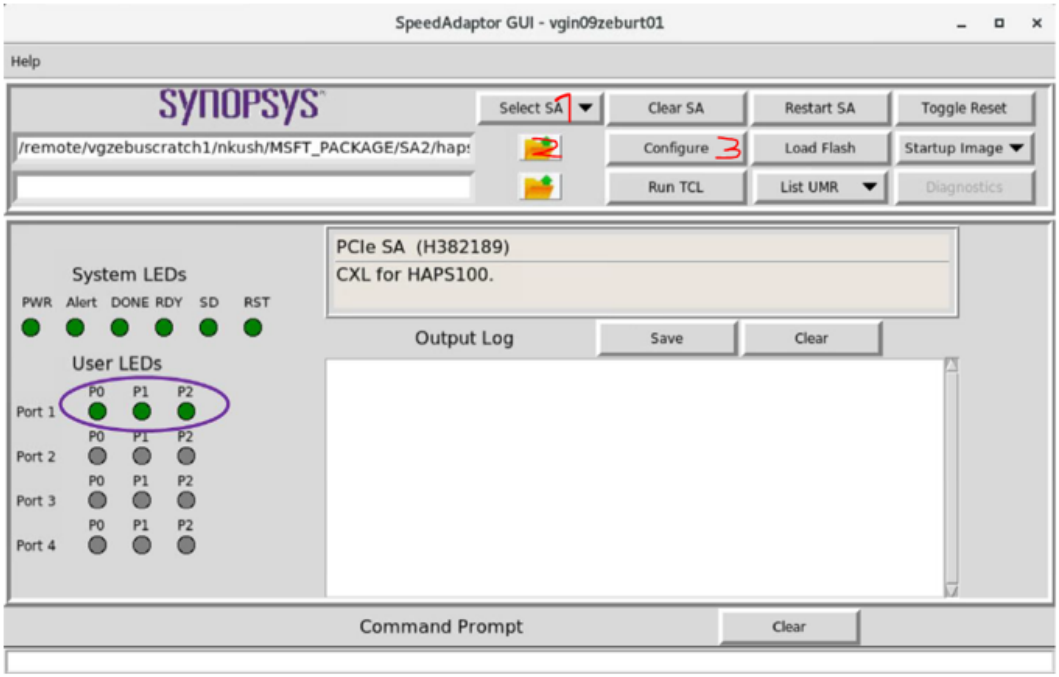


Figure 5.7: Speed Adaptor GUI

6. Power-on HAPS-100 (Wait for blinking LEDs in the front panel)

7. Configure HAPS-100 by running the following commands on HAPS-100 terminal:

```
1    $ confprosh -esm #terminal setup to communicate with HAPS
     device
2    $ cfg_scan #search for devices
3    $ cfg_open /umr3usb0/mod-X007761 -map "mb1" #select device
      X007761(HAPS100) as cfg0
4    $ cfg_project_configure cfg0 targetsystem.tsd #load
     configuration file to cfg0
5    $ cfg_reset_toggle cfg0 FB1.uD #Toggle reset
```

8. Wait for the following conditions to validate:

- Green LED in SA P1-0
- Green LED in SA P1-1
- Blinking LED in SA P1-2

5.2.2 Debug with Protocompiler100

After the setup of the systems, it is possible to check each of the link statuses through the terminals, as figure 5.8 shows.

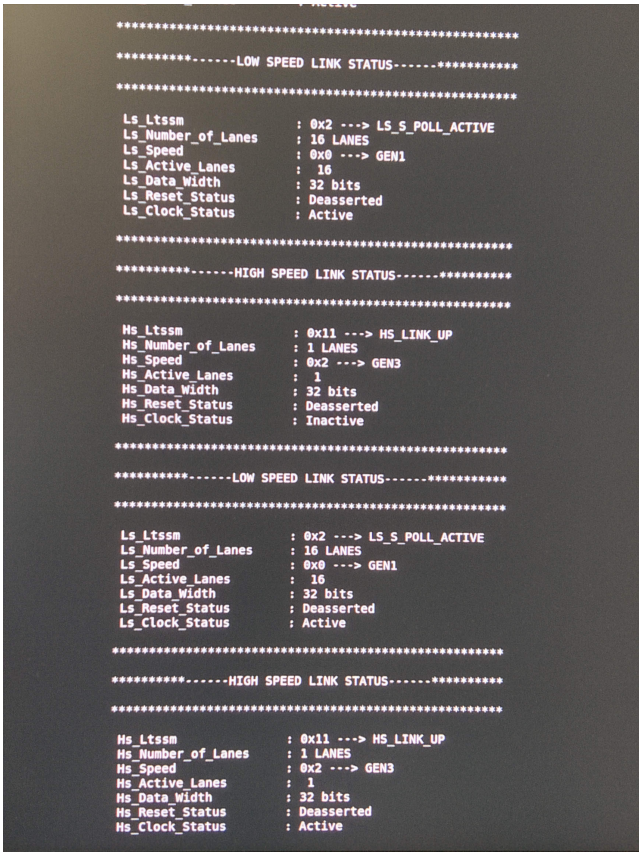


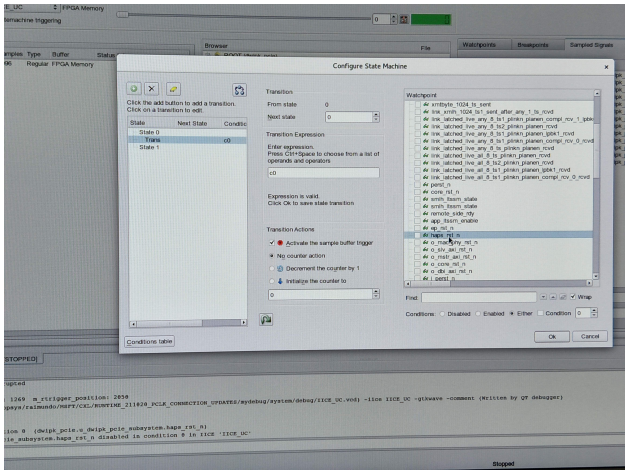
Figure 5.8: Link Status

While the high-speed side of the system is active, the low-speed side is still on Polling, which means there is some problem in connecting from the low-speed domain that links the Speed Adaptor to the HAPS-100.

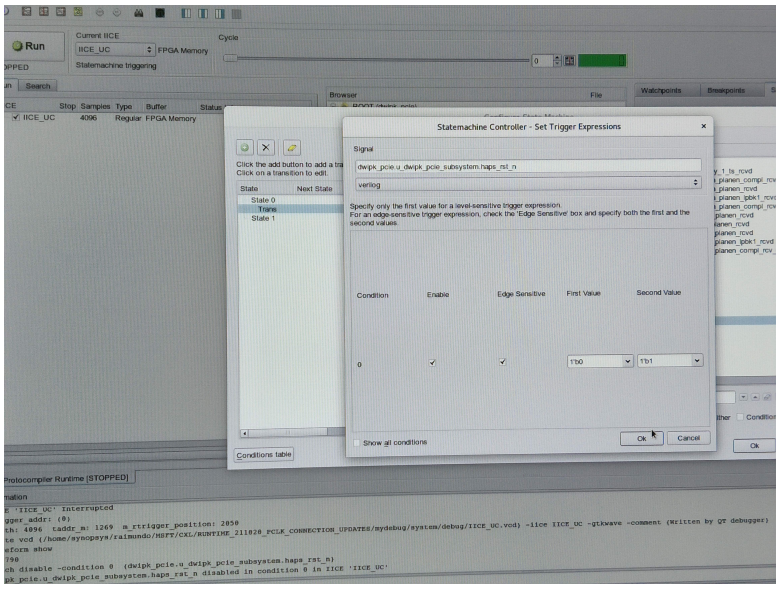
Therefore, some parameters can change to focus on a specific time window (Watchpoint) to help the debugging, as figure 5.9 shows.

Figure 5.10 presents the Protocompiler GUI, which demonstrates the waveforms of the signals that go through the links.

To summarise, the simulation issue with timing closure presented in chapter four got successfully fixed by utilising the many methods attempted in this chapter to get to the solution.



(a) Watchpoint selection



(b) Trigger Window

Figure 5.9: Watchpoint configuration window

Furthermore, the hardware setup got put together along with the linkup of the high-speed side of the system, which means the connection from the PCIe endpoint and the Speed Adaptor was successful.

However, the problem with the simulation and the configuration on the HAPS-100, which is a new prototyping model, delayed the possibility of testing the fully functioning data transactions between the two end devices that the project wanted to demonstrate.

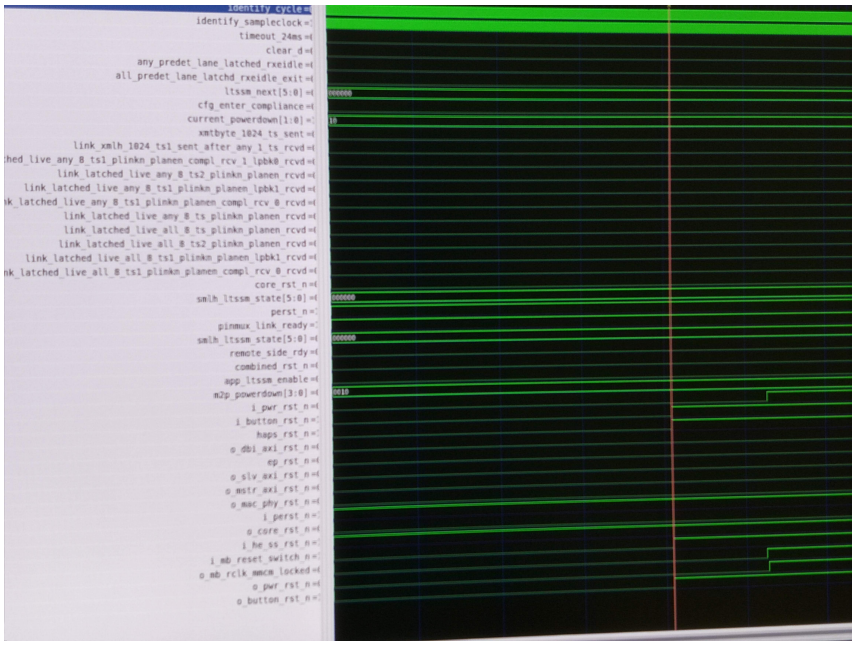


Figure 5.10: Watchpoint Waveforms

Chapter 6

Conclusion

The completion of this dissertation empowers the knowledge of the reader on many topics regarding the semiconductors field. Throughout the state of the art section of the dissertation, the structure of PCIe and the architecture of the Speed Adaptor gets explored along with their functionalities and limitations.

The primary example for this case study is the Speed Adaptor, so an explanation of its architecture got presented to showcase the hardware test example that follows.

The studied case showed the division of the SA system into two domains. On the high-speed side is the PCIe endpoint and on the low-speed side is the prototyping system.

The simulations performed demonstrated how the system is intended to work and helped discover any underlying issues that might occur, such as the timing closure invalidation.

Achieving timing closure requires positive values for the WNS and the WHS. The problem that surged during the simulation was the negative WHS value, which led to studying the hold time window and then attempting to fix the problem by changing the clock frequency in the constraints file to bridge the gap between the source and destination path.

However, the solution was not feasible as the source and destination paths were almost identical. The second try at the problem ventured through floorplanning which helped improve the timing paths by optimising the cell occupation to a single SLR, resulting in better pathing and a higher WHS value.

Although the effect of the second solution had translated into better results, even allocating all of the cells to the same SLR was not enough to overcome the negative value.

Lastly, delving deeper into the clock paths, the issue got revealed as inefficient clock utilisation. In other words, there were two clocks (aux,axi) with an inefficient path allocation, moving around two different domains.

Therefore, these two clocks got replaced by a similar frequency clock (pclk) which had a more straightforward path to the cells to which they were previously connected, solving the WHS issue entirely.

Finally, to demonstrate the Speed Adaptor can overcome the speed limitation from having two different speed technologies. An example is set utilising the HAPS-100 controller, which runs at PCIe gen one with low speed and a PCIe Endpoint, which runs at PCIe gen five with high speed.

Even though it was not possible to test the HAPS-100 due to a lack of time, this work will serve as a guide for future readers who get introduced to the area of prototyping. Moreover, it will aid designers who work on the Speed Adaptor to get a quick and better understanding of this architecture and future issues that come to exist.

6.1 Future work

In the further development of this project, the consideration of utilising this technology to apply on protocols other than PCIe might exist.

In other words, the methodologies exposed throughout the dissertation can be a consideration for bridging a different set of devices, linking different models of the same base technology as shown in this work, or connecting independent base technologies.

Additionally, once everything is running, a few tests could get performed to study how applying different loads to the prototyping system can affect the overall performance.

References

[1] H. Yu, H. Lee, S. Lee, Y. Kim, and H. M. Lee, "Recent advances in FPGA reverse engineering," *Electronics (Switzerland)*, vol. 7, no. 10, pp. 1–14, 2018. [Cited on page 2]

[2] D. Mathias, B. Hicks, C. Snider, and C. Ranscombe, "Characterising the affordances and limitations of common prototyping techniques to support the early stages of product development," *Proceedings of International Design Conference, DESIGN*, vol. 3, pp. 1257–1268, 2018. [Cited on page 2]

[3] T. Agarwal, "Basics of FPGA Architecture and Applications." [Online]. Available: https://www.elprocus.com/fpga-architecture-and-applications/ (Accessed 2021-10-28). [Cited on page 6]

[4] X. Slices, "FPGA Logic Cells Comparison," *1-CORE Technologies*, vol. 1, 2014. [Online]. Available: http://ee.sharif.edu/$\sim$asic/Docs/fpga-logic-cells_V4_V5.pdf [Cited on page 6]

[5] D. Oliver and J. Fedtke, "Comparative analysis," *Human Rights and the Private Sphere: A Comparative Study*, pp. 467–557, 2007. [Cited on page 7]

[6] T. N. Theis and H. S. Philip Wong, "The End of Moore's Law: A New Beginning for Information Technology," *Computing in Science and Engineering*, vol. 19, no. 2, pp. 41–50, 2017. [Cited on page 7]

[7] P. S. Pavan, "ASIC vs FPGA," 2020. [Online]. Available: http://www.signoffsemi.com/asic-vs-fpga/ (Accessed 2021-05-15). [Cited on page 7]

[8] D. Amos, A. Lesea, and R. Richter, *FPGA-Based Prototyping Methodology Manual*, S. Press, Ed., 2011. [Cited on pages 7 and 8]

[9] J. Mealy, "FreeRange Verilog Foundation Modeling," 2019. [Cited on page 8]

[10] S. Jose, "What is FPGA prototyping ? Why is it important to prototype ? Current Challenges in FPGA Prototyping Long Bring-Up Time." [Online]. Available: https://docplayer.net/20935024-Fpga-prototyping-primer.html (Accessed 2021-10-28). [Cited on page 8]

[11] The Curation Company, "Emulation," 2019. [Online]. Available: https://www.techdesignforums.com/practice/guides/guide-to-emulation/ (Accessed 2021-07-15). [Cited on page 8]

[12] G. Ron, "Five Challenges to FPGA-Based Prototyping," 2014. [Online]. Available: https://www.eetimes.com/five-challenges-to-fpga-based-prototyping/ (Accessed 2021-07-20). [Cited on page 8]

[13] T. Virtex and X. T. Fpgas, "Virtex-7 T and XT FPGAs Data Sheet : DC and AC Switching Characteristics," vol. 183, pp. 1–68, 2013. [Cited on page 9]

[14] Intel, "Intel® Stratix® 10 FPGAs SoC FPGA." [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10.html (Accessed 2021-09-15). [Cited on page 9]

[15] R. T. G. Fpga, "DS0131 RTG4 FPGA Datasheet," pp. 1–134, 2020. [Cited on page 9]

[16] Lattice, "Low Power FPGAs (General Purpose Optimized FPGAs)." [Online]. Available: https://www.latticesemi.com/Products.aspx#_D5A173024E414501B36997F26E842A31 (Accessed 2021-09-15). [Cited on page 9]

[17] Synopsys, "HAPS-100 Prototyping Solution," 2021. [Online]. Available: https://www.synopsys.com/verification/prototyping/haps-100.html (Accessed 2021-07-15). [Cited on pages 9 and 10]

[18] ALDEC, "HES Prototyping Boards." [Online]. Available: https://www.aldec.com/en/products/emulation/hes_fpga_boards (Accessed 2021-09-15). [Cited on page 9]

[19] Cadence, "Protium S1 Single-FPGA Board," pp. 1–3, 2017. [Online]. Available: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/system-design-verification/protium-s1-single-fpga-board-ds.pdf (Accessed 2021-05-15). [Cited on page 9]

[20] L. Rizzatti, "Hardware Emulation: A Weapon of Mass Verification," 2014. [Online]. Available: https://www.electronicdesign.com/technologies/test-measurement/article/21800385/hardware-emulation-a-weapon-of-mass-verification (Accessed 2021-07-15). [Cited on page 10]

[21] B. Bailey, "Emulator, accelerator, prototype – what's the difference?" 2010. [Online]. Available: https://www.eetimes.com/emulator-accelerator-prototype-whats-the-difference/ (Accessed 2021-07-19). [Cited on page 10]

[22] Synopsys, "Emulation Use Cases," 2021. [Online]. Available: https://www.synopsys.com/verification/emulation/use-cases-fast-emulation.html (Accessed 2021-07-15). [Cited on page 10]

[23] ALDEC, "HES-DVM Hybrid Verification Platform." [Online]. Available: https://www.aldec.com/en/products/emulation/hes-dvm--fpga-emulation-platform (Accessed 2021-09-15). [Cited on page 10]

[24] SIEMENS, "Veloce HW-Assisted Verification System." [Online]. Available: https://eda.sw.siemens.com/en-US/ic/veloce/ (Accessed 2021-09-15). [Cited on page 10]

[25] Cadence, "Palladium Z2 Enterprise Emulation Platform." [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html (Accessed 2021-09-15). [Cited on page 10]

[26] J. Ajanovic and I. Corporation, "PCI Express * (PCIe *) 3 . 0 Accelerator Features," pp. 1–10, 2006. [Cited on page 11]

[27] M. Rose, "FPGA PCIe Bandwidth," Department of Computer Science and Engineering University of California San Diego, San Diego, Tech. Rep., 2010. [Online]. Available: https://docuri.com/download/fpga-pci_59c1e86df581710b286c9d9b_pdf [Cited on page 11]

[28] J. Hruska, "The PCIe 5.0 Specification is Now Available, Before PCIe 4.0 Has Even Shipped," 2019. [Online]. Available: https://www.extremetech.com/computing/292251-the-pcie-5-0-specification-is-now-available-before-pcie-4-0-has-even-shipped (Accessed 2021-07-19). [Cited on page 11]

[29] D. A. T. S. Mindshare Inc. Ravi Budruk, *PCI Express System Architecture*. Addison-Wesley Professional, 2003. [Online]. Available: http://gen.lib.rus.ec/book/index.php?md5=66fc01319ede1e13760fc214dbfe4af8 [Cited on pages 11, 12, 17, and 19]

[30] M. Bellato, G. Collazuol, I. D'Antone, P. Durante, D. Galli, B. Jost, I. Lax, G. Liu, U. Marconi, N. Neufeld, R. Schwemmer, and V. Vagnoni, "A PCIe Gen3 based readout for the LHCb upgrade," *Journal of Physics: Conference Series*, vol. 513, no. TRACK 1, 2014. [Cited on page 13]

[31] L. Reese, "The High-Frequency Signals of PCIe 4.0 Demand Higher Performance from Engineers," 2018. [Online]. Available: http://lreese.dotsenkoweb.com/2018/03/20/

the-high-frequency-signals-of-pcie-4-0-demand-higher-performance-from-engineers/ (Accessed 2021-07-19). [Cited on page 13]

[32] E. Engineering, "Design and Verification of ACK / NAK Protocol of PCI Express Data Link Layer in System Verilog," *International Journal for Research in Applied Science Engineering Technology (IJRASET)*, vol. 4, no. XII, 2016. [Cited on page 14]

[33] S. M. Balabatti and R. C. Radha, "Calculation of ECRC FOR TLP-Packets of PCIe Protocol," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 3, no. 4, pp. 9151–9158, 2014. [Cited on page 14]

[34] M. Jackson, R. Budruk, and M. T. Series, *PCI Express 3.0*, 2013. [Online]. Available: papers3://publication/uuid/970B366C-557A-45D2-A597-6D8E36A4FA22 [Cited on pages 14, 15, 16, 19, 21, 22, and 23]

[35] I. Technologies, "Link Initialization and Training in MAC Layer of," vol. 6, no. 3, pp. 2717–2719, 2015. [Cited on page 16]

[36] Xilinx, "Debugging Guide for 7-Series Integrated PCI Express Block Link Training Issues," 2021. [Online]. Available: https://support.xilinx.com/s/article/56616?language=en_US (Accessed 2021-10-28). [Cited on pages 18 and 20]

[37] T. Lecroy, "PCIe LTSSM Link Partner TxEQ Response Characterization and Debug during Link Equalization Training," pp. 1–9, 2018. [Online]. Available: https://teledynelecroy.com/doc/leq-response-during-training-app-note (Accessed 2021-10-28). [Cited on page 18]

[38] E. T. ChinaAET, "PCIe literacy - the basic part of the physical layer logic (2)," 2018. [Online]. Available: https://programmersought.com/article/84311699099/ (Accessed 2021-07-19). [Cited on pages 20 and 21]

[39] J. Liu, A. Mamidala, A. Vishnu, and D. K. Panda, "Evaluating infiniband performance with PCI express," *IEEE Micro*, vol. 25, no. 1, pp. 20–29, 2005. [Cited on page 21]

[40] Dadi Sahiti, "Design Implementation of High Speed PCI using MAC Transmitter with PIPE Interface," *International Journal of VLSI Design Communication Systems*, vol. 05, no. 05, pp. 402–406, 2017. [Cited on page 21]

[41] A. Vasjaeva , N.S. Ivanov , K.V. Suhih, "Increase of productivity of the PCI Express switch for cluster s," *V mire nauchnykh otkrytiy*, vol. 0, no. 10, p. 24, 2014. [Cited on page 23]

[42] Synopsys, "What is Static Timing Analysis (STA)?" [Online]. Available: https://www.synopsys.com/glossary/what-is-static-timing-analysis.html (Accessed 2021-09-01). [Cited on page 35]

[43] K. Balston and B. A. Sc, "FPGA Emulation for Critical-Path Coverage Analysis," Ph.D. dissertation, The University Of British Columbia, 2012. [Cited on page 34]

[44] P. P. Chu, *FPGA Prototyping by VHDL Examples*, WILEY, Ed., 2008. [Cited on page 38]

[45] D. K. Tala, "Verilog Introduction." [Online]. Available: https://www.asic-world.com/verilog/intro1.html#Design_Styles (Accessed 2021-09-15). [Cited on page 38]

[46] S. Akthar, "FPGA Design Flow," 2014. [Online]. Available: https://allaboutfpga.com/fpga-design-flow/ (Accessed 2021-07-20). [Cited on pages 41 and 42]

[47] A. Ling, D. P. Singh, and S. D. Brown, "FPGA technology mapping: A study of optimality," *Proceedings - Design Automation Conference*, pp. 427–432, 2005. [Cited on page 42]

[48] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning To Timing Closure*, S. NETHERLANDS, Ed., 2011. [Cited on page 47]

[49] VLSICoding, "Setup Time and Hold Time," 2014. [Online]. Available: https://vlsicoding.blogspot.com/2014/05/setup-time-and-hold-time.html (Accessed 2021-09-20). [Cited on page 48]

[50] K. Saban, "Interconnect Technology Delivers Breakthrough FPGA Capacity , Bandwidth , and," 2011. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp380_Stacked_Silicon_Interconnect_Technology.pdf [Cited on page 48]

[51] Xilinx Inc., "Vivado Design Suite User Guide," *Ug903*, vol. 4, pp. 1–173, 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug903-vivado-using-constraints.pdf [Cited on page 48]

[52] A. Sutton and N. Songcuan, "Achieving performance targets with multi-die FPGA-based prototyping hardware in the face of design changes," 2021. [Online]. Available: http://signal-processing.mil-embedded.com/articles/achieving-hardware-the-face-design-changes (Accessed 2021-09-20). [Cited on pages 48 and 49]