FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

PCI Express In-System Performance Analyzer

Diogo André Duarte Correia

MSc DISSERTATION

Integrated Master in Electrical and Computers Engineering

Supervisor at FEUP: João Canas Ferreira

Supervisor at Synopsys: Takuya Omura

October 7, 2015

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES 2014/2015

A Dissertação intitulada

"PCI Express In-System Performance Analyzer"

foi aprovada em provas realizadas em 07-10-2015

o júri

[illegible]

Presidente Professor Doutor António José Duarte Araújo
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

[illegible]

Professor Doutor Manuel Gradim de Oliveira Gericota
Professor Adjunto do Departamento de Engenharia Eletrotécnica Instituto Superior de Engenharia do Porto

[illegible]

Professor Doutor João Paulo de Castro Canas Ferreira
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

[illegible]

Autor - Diogo André Duarte Correia

Faculdade de Engenharia da Universidade do Porto

Abstract

This document presents a behavioral system-level infrastructure for a device-centered in-system performance analyzer for use in PCI Express systems, motivated by the necessity of providing the costumer with integrated solutions for detection of performance anomalies.

Taking into consideration the restrictions imposed by an implementation in hardware, namely increase in area and cost, a SystemVerilog class based solution was proposed as a proof of the concept. A simplified approach that aims at minimizing the cost of the implementation has been adopted. Supported features include throughput measurement and bottleneck identification.

The solution was designed in a verification environment that runs traffic based simulations, thus enabling the creation of different scenarios for testing. The validation of the implementation is done within that environment by using SystemVerilog Assertions, among other strategies. Additional test cases were designed to allow testing the accuracy of the bottleneck detection.

The outcome is an infrastructure capable of supporting the main PCI Express configurations and be part of the verification environment.

Resumo

Este documento apresenta uma validação de conceito, num ambiente comportamental, de uma proposta de análise de performance integrada no produto, para uso em sistemas PCI Express. A escolha é motivada pela necessidade de fornecer aos clientes soluções que permitam identificar anomalias no comportamento do sistema real, fora de um ambiente de verificação controlado.

Partindo das restrições inerentes a uma implementação em hardware, onde se destacam o aumento de área e custo, uma solução baseada numa arquitectura de classes em SystemVerilog é proposta como uma prova do conceito. Uma abordagem simplista procurou minimizar os possíveis custos de uma futura implementação física. A proposta inclui medição do débito de dados e identificação das causas para perdas no mesmo partindo do valor de referência.

A solução apresentada é desenvolvida num ambiente de verificação estruturado para correr simulações baseadas em tráfego, permitindo assim definir e testar diferentes cenários. A validação da implementação também é levada a cabo dentro do ambiente de verificação ao recorrer a SystemVerilog Assertions, entre outras estratégias. Testes adicionais foram desenvolvidos de forma a testar o aparecimento das diferentes causas para perda de débito.

A principal contribuição para o problema consiste numa infraestrutura capaz de suportar as configurações mais importantes da tecnologia e de integrar o ambiente de verificação.

Acknowledgments

The development of this dissertation marks the end of my academic path in the Integrated Masters in Electrical and Computers Engineering at the Faculty of Engineering of the University of Porto. Throughout the 5 years spent here, I developed skills that were applied into this document and will certainly be applied during my professional career from now on. Having said this, I thank all the FEUP community for the transmitted knowledge, especially to the professor João Canas Ferreira, who guided me in my work for the dissertation.

I also want to thank my friends from university. Being able to share the study environment with you was very pleasant and I have learned much with it. Another special thanks to my friends from Erasmus, in Sweden and in Ireland, with whom I learned new languages, habits and above everything a new way of facing life.

A very special thanks goes to my family for their support under all stages of my academic and personal life. Without you I would not have been able to accomplish this mark.

Finally I want to thank Synopsys for the opportunities directed to me. In special, thanks to João Gonçalves, Tiago Prata and Miguel Sousa Falcão, from the office in Porto, to Frank Kavanagh, Paul Cassidy and Takuya Omura, from the office in Dublin, and also to Taichi Ishitani, from the office in Tokyo.

To all of you, many thanks!

Diogo Correia

Contents

List of Figures

List of Tables

Abbreviations

ACK	Acknowledged
API	Application Programming Interface
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DLLP	Data Link Layer Packet
DUT	Device Under Test
ECRC	End-to-End Cyclic Redundancy Check
EMI	Electromagnetic Interference noise
FC	Flow Control
IP	Intellectual Property
I/O	Input/Output
LCRC	Link Cyclic Redundancy Check
MPS	Maximum Payload Size
NAK	Not-Acknowledged
PCI	Peripheral Component Interconnect
PCI-X	Peripheral Component Interconnect Extended
PLL	Phase-Locked Loop
PLP	Physical Layer Packet
PM	Power Management
QoS	Quality of Service
TC	Traffic Class
TLP	Transaction Layer Packet
VC	Virtual Channel
VTB	Verilog Test Bench

Chapter 1

Introduction

Nowadays society relies on information technologies to ensure a sustainable functioning of the human world. As systems become more complex, and consequently more demanding, the search for thorough solutions increases.

In the semiconductor industry, according to Moore's law, the number of transistors in a dense integrated circuit doubles approximately every two years. In practice, it defines the rate by which CPU speed increases, thus dictating the maximum amount of operations per unit of time. The result is a metric to address the potential of a system.

However this carries some negative implications as well. Obsolescence is a metric that defines the state of something that is no longer wanted, even though it still may work in the expected conditions. In other words, advances in the industry have strong influence on the lifetime of the involved technologies. This translates into the necessity of keeping up with the pace of the industry at all levels. As new alternatives emerge, companies put a lot of effort into determining whether, and how much, their technology can be enhanced to meet the new scenarios.

This is particularly important in the IP World, composed by IP vendors and system integrators, as decisions made in this field have influence over a wide range of systems and possible failures to comply with the requirements may severely affect the involved entities. Consequently a big pressure is put on IP vendors to provide cutting-edge solutions. The implications are that performance analysis and optimization represent fields of work in which companies are allocating significant resources in the expectation of fulfilling their business goals.

Performance analysis is important because it evaluates the behavior of the system and its margin for optimizations [8]. Through this, developers and vendors possess means to detect worst-case scenarios and carry out the necessary fixes that allow meeting the costumers' requirements. Furthermore, it enables predicting the evolution of the requirements and with it working on features early in time.

Optimization is in part what can be achieved based on the performance analysis, so that the behavior of the system be improved. It represents the previously mentioned fixes and aim at maximizing the performance of the system as much as possible.

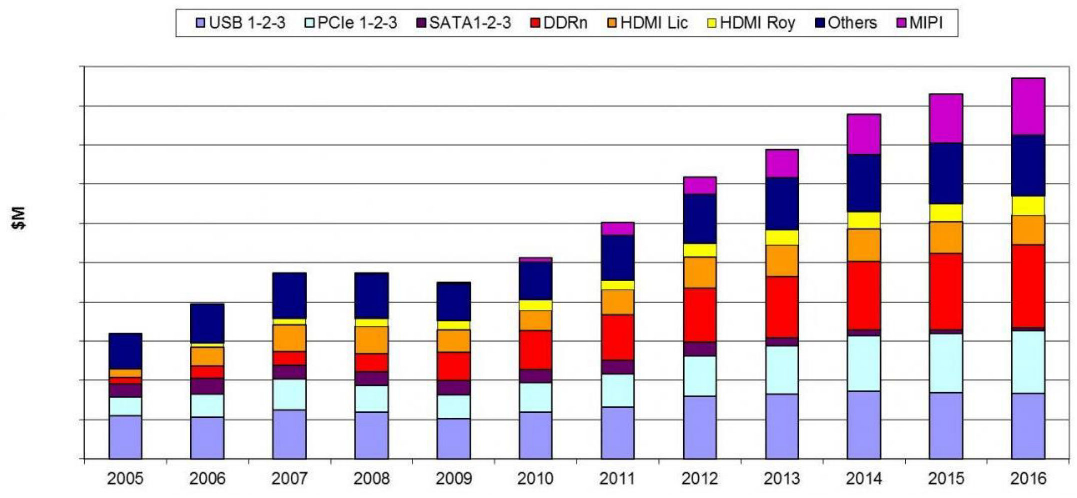

Figure 1.1: PCI Express market share (From [1])

1.1 Context

PCI Express stands for *Peripheral Component Interconnect Express* and defines a high-speed serial computer expansion bus standard used to interconnect different components of the same system. Since it was standardized in 2003, it has been used as a de-facto inside the box interconnect solution in a variety of systems.

Presented as a solution for interface IP, PCI Express shares the market with other protocols as shown in figure 1.1. While some protocols are dedicated to specific systems, PCI Express covers a wide range of applications. It is easily found in many motherboards, connecting the CPU and several peripheral devices such as video cards or external disk drives. In fact, this support for interoperability between different systems represents the main reason for the success of the standard.

The main consequence is a great demand for reconfigurability and adaptability. This has led to the release of new versions, each one with improvements in the speed of operation. This work is essentially centered in PCI Express generation 3.0.

The performance analysis may be implemented as part of the verification environment, or as part of the hardware. The distinction between these alternatives is of extreme importance due to the limitations carried by the latter in terms of additional logic, which leads to an increase in both cost and area.

1.2 Purpose

Given the range of applications in which it is used, adopters find the requirements for a PCI Express link an essential question to address. Nevertheless, deciding what the requirements are represents a challenging task. Link speed and link width need to meet application requirements, such as bandwidth, latency, power consumption and area, at a minimum cost.

For instance, the enterprise market needs high reliability and high bandwidth, whereas the mobile market needs low power consumption whilst still aiming at a high bandwidth. To follow the market needs, PCI Express has been expanding continuously in both Link speed and low power features. The result is a standard required to support several configurations, hence extremely complex.

The purpose of this work is the development of a system-level validation of a performance analyzer for a PCI Express device, aiming at the implementation of the solution into the IP Core at a later stage.

Unlike the majority of the performance analyzers, which are implemented in software and used in controlled verification environments, this document presents a study for an integrated solution. This means the developer must take into account the trade-off between the gain and the cost of the implementation, as the solution adds new logic and new registers to the IP Core.

It is therefore important to define what the most important features are, as well as if there is a reasonably low computational effort involved in implementing them.

1.3 Goals and scope

The topic treated throughout the document is part of a bigger project, whose goals are the following:

- Define a method to measure performance;
- Define a method to analyze performance issues;
- Add new functionalities to the IP Core for performance optimization;
- Provide the costumer with a guide for performance optimization.

The proposed set of goals associated with this document aims at modeling a system-level validation of the concept, performed in a behavioral fashion within a verification environment. The expectations are that the work done here enables determining the potential of the whole project, by defining what is worth implementing.

Given this, the following list presents the milestones for the topic of the project:

- Provide the existing IP Core with throughput measurement;
- Analyze the results obtained from the simulations;
- Find a bottleneck when the results are unexpected;
- Provide an infrastructure to test and tune the performance analyzer;
- Propose a physical implementation based on the above, to add new functionalities to the IP core for performance optimization.

The scope of the project is therefore an in-system solution for performance analysis at a device-level via the measurement of the throughput. No routing information or extensive configuration characteristics of the PCI Express standard are covered here.

Further statistics on the traffic pattern are collected, providing the user with a convenient way of identifying the type of traffic transferred during the simulation. Moreover, the contribution of each type of traffic to throughput can be determined. This may be useful when the costumer is interested in prioritizing the throughput of a specific type of data over the others.

1.4 Contributions

Due to confidentiality reasons, some information with respect to the implementation and the contributions is not exposed in this document. Notwithstanding, sufficient coverage of the involved concepts is provided so that understanding the whole solution remains possible.

That said, the work done in the topic resulted in the following contributions:

- Solution for throughput measurement;
- Performance analysis with bottleneck detection;
- Test infrastructure to allow the validation of the system and its use in verification;
- Documentation of the system and guide to add new functionalities.

The validation of the produced solution was carried out by a few techniques, namely the use of assertion checking. Not only are the results compared against theoretical values, but also the behavioral results are checked against the ones from the existing IP Core functionality.

1.5 Structure of the document

The document is structured in five chapters.

Throughout this chapter 1 an introduction to the topic comprised the motivation, goals and a summary of the contributions that resulted from the work carried out.

Chapter 2 provides a further reflexion on PCI Express and the characteristics that most enhance, or impact, performance. Two brief sections further cover the topic of performance analysis and its main foundations, and the state of the art for performance analysis in the scope of PCI Express.

The methodology used to complete the goals for the project is treated in chapter 3. A description of the implemented solution, the motivations and justifications for the decisions, and the strategy for validation are covered.

The project outcomes are presented in chapter 5, followed by the conclusions and future work in chapter 6.

Chapter 2

Background

This chapter covers the concepts required to understand the topic of the document.

A first section introduces the PCI Express specification and the characteristics that are directly connected to the work carried out. Describing the complete specification is not the intention of the section. Instead, it is mainly directed to the mechanisms that somehow have impact in performance.

A second section explores some techniques used in performance analysis. Its purpose is to give an overview of the strategies involved in performance analysis and support the motivation for the chosen methodology. It further expands its scope to the analysis of bottlenecks and to the bases on top of which the performance metrics and the validation of the concept are built. Since the scope of the project is not related to pure testing or verification, this section aims essentially at discussing the strategies utilized to ensure that the implementation and the results are coherent.

Finally, another section presents some examples of similar projects in the field. The main objective is to draw the attention to what has been carried out in the topic of performance analysis in PCI Express, mostly in a verification environment. Not many in-system solutions have been addressed in the literature so far.

2.1 PCI Express

The PCI Express bus is widely used in nowadays computational systems. But its history started two decades ago, and recalling some facts may help understand its motivation and functioning.

Conventional PCI, abbreviation for *Peripheral Component Interconnect*, is a bus standard released in 1992, capable of supporting 256 16-bit devices. With the introduction of 32-bit devices and need for faster interconnection busses, new alternatives were required and PCI-X, short term for *Peripheral Component Interconnect eXtended*, was released in 1998 [9].

Both Conventional PCI and PCI-X were parallel bus standards and PCI-X maintained compatibility with its predecessor. Nonetheless both suffered from the limitations associated with parallel busses, namely the speed limit and the great number of I/O pins, as well as limitations from their multi-drop parallel characteristics, that demanded medium access control [5].

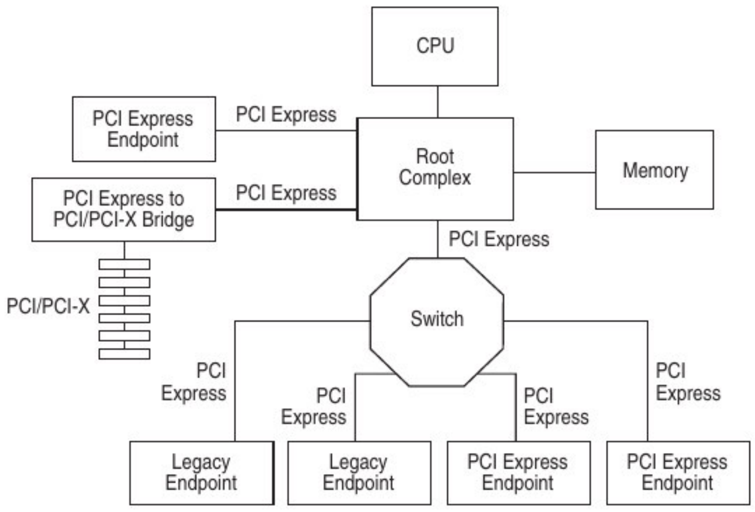

Figure 2.1: PCI Express general topology (From [2])

To avoid these limitations PCI Express was designed as a serial bus standard and makes use of point-to-point busses. Reduced number of I/O ports and increased data rate due to limited load on the link became possible. Switching devices maintained it suitable to interconnect devices and bridges permit connecting to its predecessors multi-drop busses, maintaining backward compatibility [4].

A general view of the PCI Express topology is shown in figure 2.1. The Root Complex connects the CPU and memory subsystem to the PCI Express switch fabric, and contains a set of PCI Express ports. The Switches and Endpoints can connect to these ports. Legacy Endpoints are out of the scope here, as they support few functionalities [4].

2.1.1 The PCI Express link

The interconnection of two PCI Express devices is carried out via a point-to-point bidirectional link, as shown in figure 2.2. It may be composed by up to 32 lanes, increasing bandwidth by allowing transmission in parallel through the lanes [5].

The lane is a pair of transmission and reception lines that operate in dual-simplex, meaning that transmission and reception is possible at the same time. It is essentially the same as full-duplex, except that each communication line has its own ground wire [4].

No clock signal is used, as it is recovered at the receiver side by detecting transitions in the bit stream. As a consequence transitions must occur in every symbol transferred, which is achieved by encoding the bit stream. Despite the overhead, this solves many issues that other standards faced. Clock skew resulting from using a common clock signal in a parallel bus and data skew are harmful characteristics, limiting the frequency of the clock cycle [10]. By mitigating these undesired effects PCI Express secures higher rates, fulfilling the need for compensating the fact that the data is serialized.

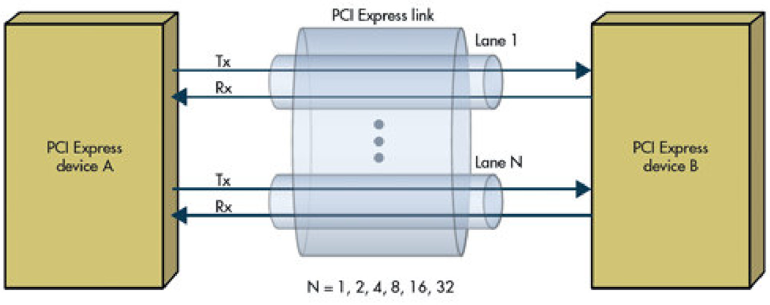

Figure 2.2: PCI Express link (From [3])

2.1.2 A layered protocol

Handling bit streams is a painful task and therefore data communication in PCI Express is defined in terms of transactions by transmitting and receiving TLPs, or *Transaction Layer Packets*. A transaction is a packet exchange required to complete a transfer of information between a requester and a completer. They are categorized into non-posted transactions - the completer responds back to the requester -, and posted transactions - no answer is returned by the completer [4].

TLPs are divided into four address spaces and they are shown in table 2.1. This document considers mainly the memory space and the configuration space. While most of the data transactions are related to the memory space, the configuration space is also important to the scope of the project further ahead.

Just a remark to state the role of the messages, introduced with PCI Express. In-band messages are exchanged in order to eliminate sideband signals related to interrupts, error handling, and power management. It would not be efficient to have a two lane link and a conjunct of sideband signals. Therefore the protocol replaces these signals with a set of in-band signals, some conveyed as TLPs and some as DLLPs, short form of *Data Link Layer Packets* [4].

A last type of TLP that does not fit into any of the categories, and therefore is not displayed, is the completion. It is part of a non-posted transaction and may refer to any of the address spaces, depending on the request type [2].

However, it is necessary more than a data path and several address spaces to provide a good service on a data bus and therefore data communication in PCI Express is defined within a layered protocol. Devices may not implement the exact layered architecture as long as the functionality required by the specification is supported [4]. In any case, it creates several levels of abstraction that hide the mechanisms that reside at the levels below, easing the process of adding functionalities or ensuring compatible solutions.

The layers that form the protocol are displayed in figure 2.3. The next subsections focus on each of the three layers, bringing up concepts that are closely related to performance.

Address space	Transaction types	Basic usage
Memory	Read Write	Transfer data to/from a memory-mapped location
I/O	Read Write	Transfer data to/from a I/O-mapped location
Configuration	Read Write	Device Function configuration/setup
Message	Baseline (including Vendor-defined)	From event signaling mechanism to general purpose messaging

Table 2.1: Transaction Layer Packets (Table taken from [2])

2.1.3 Transaction Layer

Starting from a top-down approach, the highest level of abstraction that directly interacts with the PCI Express protocol is the Device Core, or Software Layer. It connects to the PCI Express layered core via the Transaction Layer, whose main function is to assemble TLPs before transmission and to disassemble TLPs after reception [4].

Upon reception of a request from above, the Transaction Layer creates an header that contains information such as the type of TLP, source and destination addresses, and length of the data payload, among other information. The header size is 12, or 16 bytes, depending on whether the device address space is 32- or 64-bit. Additional information can be carried in prefixes before the header. In case there is data payload, it is appended to the end of the header [5]. An optional ECRC field, standing for *End-to-End Cyclic Redundancy Check*, can be added for error detection. At the receiver side, failure to pass the check results in dropping the TLP.

In addition, the Transaction Layer also takes part in the crucial process of flow control. It contains buffers to store outbound TLPs before transmission, and to store inbound TLPs upon reception. On the transmitter side, both the transmission and ordering of TLPs obey to rules defined in the specification. The flow control mechanism is further analyzed in subsection 2.1.7 [2].

Finally, the Transaction Layer is also important during the initialization and configuration functions, by storing configuration information generated by the processor or management device and also by storing the link capabilities generated by the Physical Layer hardware negotiation of width and operational frequency. In addition, the layer supports power management, controlled by software and hardware [2].

2.1.4 Data Link Layer

The most relevant purpose of the Data Link Layer to this project is to ensure data integrity at a link level. Upon reception of a TLP from above, the Data Link Layer attaches to it a sequence ID and an optional LCRC field, abbreviate for *Link Cyclic Redundancy Check*. The sequence ID

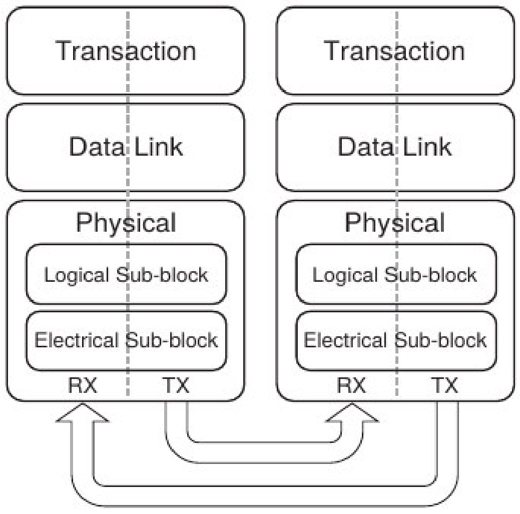

Figure 2.3: PCI Express Device Layers (From [2])

allows proper ordering of the data packets, while the LCRC field enables error checking and error reporting at every device through which the TLP passes until it reaches the completer [4].

Other functions of the Data Link Layer involve power management and flow control, by exchanging information with the link partner. Given the responsibility of the Data Link Layer in keeping the link operational, another type of packets is defined.

DLLPs are originated at the Data Link Layer and travel through the link to be consumed by the Data Link Layer of the link partner. Since it does not pass through switches, no routing information is needed and the packet remains small at 8 bytes. The packets can therefore be dropped upon errors as, even though these packets are not acknowledged, time-out mechanisms built into the specification permit their recovery [2].

As for TLPs, in the presence of an error and further reception of a *Not-acknowledged* DLLP (NAK), the transmitter resends the packet from the replay buffer. Packets are cleared from this buffer as soon as an *Acknowledged* DLLP (ACK) for that packet is received. Error indications for error reporting and logging mechanisms allow for re-training the link upon successive errors. The sequence ID is also tested on the received side to check for dropped or out-of-order TLPs [4].

From the above paragraphs one may conclude that the Data Link Layer plays an important role in the performance of the device, as both the timeouts and the replay buffer may be source of anomalies, thus affecting the traffic flow.

2.1.5 Physical Layer

The lowest layer in PCI Express is the Physical Layer. It isolates the upper layers from the signaling technology used for link data interchange, providing a level of abstraction that simplifies the implementation of functionalities.

As shown in figure 2.3, this layer is further divided in two sub-layers: the Logical Physical Layer and the Electrical Physical Layer [2]. The first processes packets before transmission and after reception, while the second is the analog interface of the Physical Layer that connects to the Link. The latter is therefore out of the scope of the project, as the main focus is at a higher level.

The Physical Layer receives DLLPs and TLPs from above and adds framing symbols to indicate beginning and end of packet. Furthermore, the data stream is processed per byte and striped across the available lanes on the link and scrambled with the aid of a Linear Feedback Shift Register type scramble, reducing the average *Electromagnetic Interference* noise (EMI) through eliminating repeated bit patterns [4].

The resulting bytes are encoded by $8b/10b$ or $128/130b$ encoding logic, depending on the link speed, to guarantee sufficient transitions to recreate the receive clock with a *Phase-Locked Loop* (PLL). This enhances the EMI noise reduction by eliminating the need for a clock signal [11]. This represents a rather constant overhead, as the system is not expected to often change speed.

Finally, a parallel-to-serial converter generates a serial bit stream of the packet on each Lane and transmits it differentially at $2.5GT/s$, $5GT/s$, or $8GT/s$, according to PCI Express generation 1.x, 2.0, or 3.0, respectively. This is shown in table 2.2, where the overhead associated with the encoding is considered.

PCIe architecture	Raw bit rate	Interconnect bandwidth	Bandwidth per lane per direction	Total bandwidth for x16 link
PCIe 1.x	$2.5GT/s$	$2Gb/s$	$\approx 250MB/s$	$\approx 8GB/s$
PCIe 2.0	$5.0GT/s$	$4Gb/s$	$\approx 500MB/s$	$\approx 16GB/s$
PCIe 3.0	$8.0GT/s$	$\approx 8Gb/s$	$\approx 1GB/s$	$\approx 32GB/s$

Table 2.2: PCI Express bandwidth example scenario per generation (From [5])

For PCIe 1.x and PCIe 2.0 the byte stream at raw bit rate is encoded by $8b/10b$. Thus 20% of overhead is taken into account in the interconnect bandwidth. For PCIe 3.0 the byte stream is encoded by $128b/130b$. For simplicity, this overhead is not being considered.

Other functions of the Physical Layer include link initialization, link training and clock tolerance compensation, all of which run through an automatic process without software intervention. To perform these functions, the Physical Layer is also capable of generating its own packets called PLPs, standing for *Physical Layer Packet* and also known as Ordered-Sets [4]. As with the DLLPs, PLPs are exchanged between link partners and therefore their size remains small because no routing information is needed.

Clock tolerance compensation is carried out via the periodic transmission of SKIP PLPs, allowing the receiver to compensate for frequency variations between the Rx Clock (derived from the bit stream and matches the Tx Clock) and the receiver Local Clock. The period by which SKIP PLPs are schedule is defined. Table 2.3 displays the minimum and upper limits for the transmission of two consecutive SKIP PLPs.

PCIe architecture	Lower bound	Upper bound
PCIe 1.x	$1180symbols$	$1548 + (MaximumPayloadSize + 28)$
PCIe 2.0		
PCIe 3.0	$370blocks$	$375blocks$

Table 2.3: SKIP PLP transmission latency upper and lower bounds (From [2])

For PCIe 1.x and PCI 2.0, the last 28 symbols consist of the header 16 bytes, the optional ECRC 4 bytes, the LCRC 4 bytes, the sequence ID 2 bytes and framing symbols 2 bytes. Moreover, the unit of measurement is the symbol, consisting of 10 bits. For PCIe 3.0, the unit of measurement is the block, consisting of 128 bits.

Link initialization, link training and link recovery are carried out by exchanging of other types of PLPs. While link recovery is motivated by the necessity of recovering the normal state of operation upon loss of symbols caused by anomalies in the SKIP PLP transmission, link training initialization and link training are used at the negotiation of the characteristics of the link and the important concepts to the scope of the project are listed below [5]:

- **Link width** — devices with different number of lanes per link may be connected, setting the link width to the minimum number of lanes between the two devices;
- **Link data rate** — training is completed at $2.5Gb/s$. Each node advertises its highest data rate capability and the link is initialized with the highest common frequency supported by the devices;
- **Lane-to-lane de-skew** — due to link wire length variations and different driver/receiver characteristics on a multiple-lane link, there is skew between lanes. This is compensated by adding/removing delays on each lane.

The link is not expected to enter link training very often, since it is mainly used after initialization and when recovery is required. Nonetheless speed changes and link width changes must also bring the link down for new negociation. Therefore it is taken into consideration in the document, as its effect generally lasts for a considerable amount of time.

At this stage, the general overview of the PCI Express layered protocol leads to some important notes. In addition to the processes acting on data before and after transmission, it brought up a few hints about mechanisms that impact performance. Summarizing, flow control acts at the Transaction Layer and may halt transmission when the receiver cannot handle such amount of data; the Data Link Layer adds futher overhead to the TLP packets and may halt their transmission upon the exchange of DLLPs or replay buffer use; the Physical Layer adds the encoding overhead to the bit stream and may bring the link down for re-training when issues arise.

Finally, just a note to recall the TLP packet structure and overhead when it travels down the layers and into the link. It can seen in figure 2.4 for PCIe 1.x and PCI 2.0. PCIe 3.0 introduces

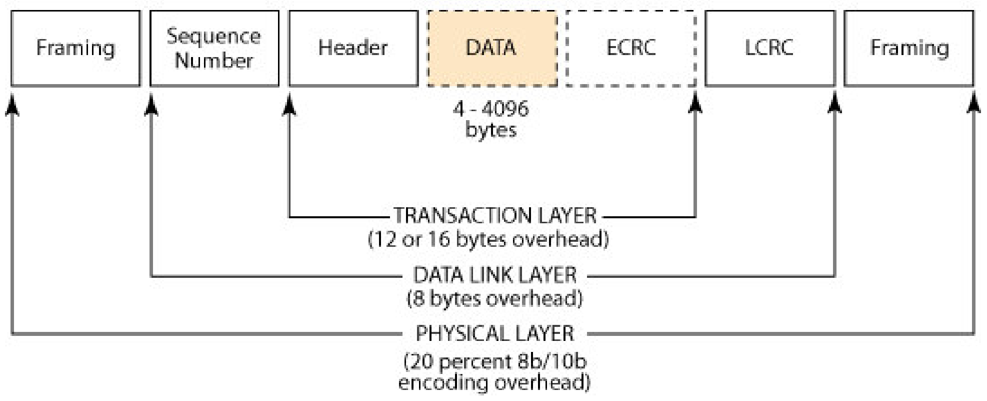
Framing
Sequence Number
Header
DATA
ECRC
LCRC
Framing
4 - 4096 bytes
TRANSACTION LAYER (12 or 16 bytes overhead)
DATA LINK LAYER (8 bytes overhead)
PHYSICAL LAYER (20 percent 8b/10b encoding overhead)

Figure 2.4: PCI Express TLP packet format (Adapted from [4])

a few modifications, namely the deletion of the framing at the end of the TLP. The framing at the beginning of the TLP contains the TLP length, enabling the detection of the end of the packet.

2.1.6 Split transactions

It has been mentioned that PCI Express transactions may be classified as posted or non-posted, according to the need for a completion. While its predecessor busses used active waiting until the reception of the completion for a request, PCI Express separates the non-posted transaction into two complete, and independent, TLPs. This mechanism is known as the *Split Transaction Protocol* and its illustrated in figure 2.5.

The completer will only generate the completion TLP once it has data and/or status ready for delivery, resulting in a free link during the time between a request and a completion. On the other hand, it requires additional overhead as two complete TLPs must be generated to carry out a single transaction [2]. Therefore posted transactions are available. Since they add uncertainty, write posting to memory is acceptable, whereas writing to I/O and configuration space may change the behavior of the devices and consequently is not allowed [4].

2.1.7 Flow Control overview

Each port at an end of a link in PCI Express must implement flow control. It is essentially a method for communicating the receive buffer status of a receiver to a transmitter, providing ways of preventing buffer overflow, but also underflow. A special note to say that flow control is point-to-point and not end-to-end [2].

In practice, flow control is implemented by using the buffers mentioned before in section 2.1.3. These buffers store non-posted requests, posted requests and completions in separate FIFOs, short form for *First-In-First-Out*. Furthermore, a credit-based protocol makes it possible for devices to keep track of the space available at the receiver buffers. Credits are divided into header and payload credits. The first category defines the DWORD (4 bytes) as the base unit, whereas the

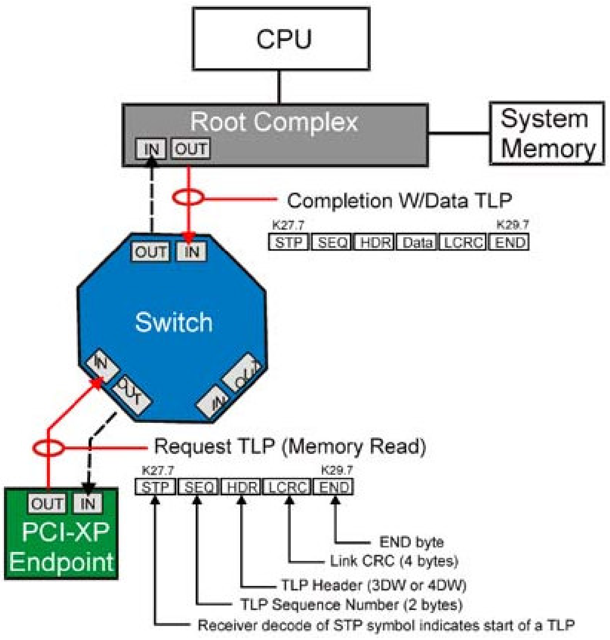

Figure 2.5: PCI Express transaction request and completion TLPs for a memory read (From [4])

latter defines 4 DWORDs (16 bytes) as the base unit [4]. Therefore, 6 buffers are utilized per virtual channel and are shown in figure 2.6. More on this in section 2.1.8.

This information is afterwards updated periodically with the transmission of flow control DLLPs between link partners [5]. This represents an extra overhead, but its impact is minimized considering the small fixed size of DLLPs. Despite of assembling and disassembling DLLPs, the Data Link Layer actually shares the responsibility of flow control with the Transaction Layer, as the buffer utilization counters are contained at the Transaction Layer [2]. The overall mechanism involves the following:

- **Devices report available buffer space** counted in units called credits. Flow control DLLPs update the information periodically;

- **Reveivers register credits** in the transmitter-side transaction layer complete the transfer from one link partner to the other;

- **Transmitters check credits** before transmitting a TLP. If there are not enough credits, it halts transmission for the given type of transaction until more credits are available.

With this, the main mechanisms within PCI Express that are directly connected to performance were covered. The work accomplished throughout the project is essentially built over the study of these mechanisms and their impact in the behavior of the system.

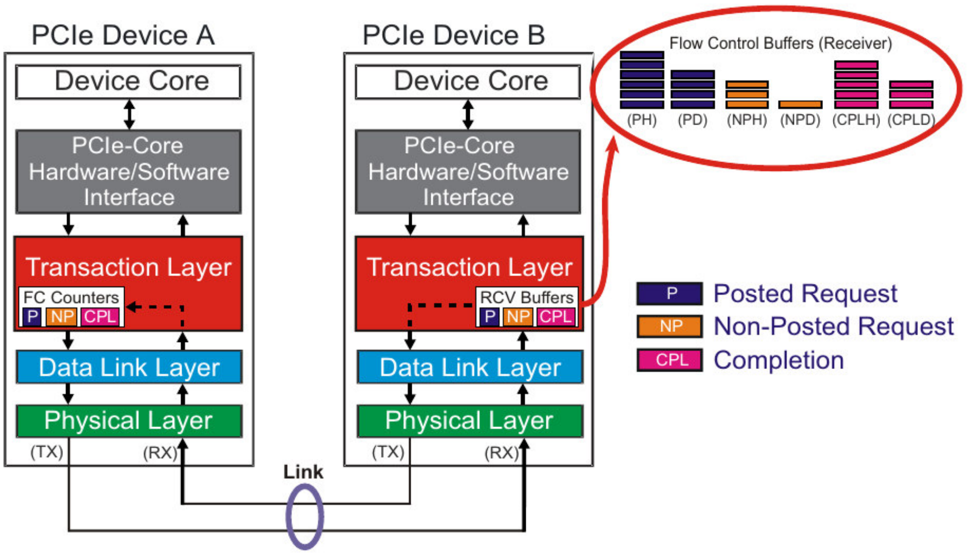

Figure 2.6: PCI Express flow control buffer organization per virtual channel (From [5])

2.1.8 Some remaks on Quality of Service

However, before moving on to the next section, a few remarks on Quality of Service are important to further comprehend the potential of the standard. Quality of service is a generic term that refers to the ability of an entity to provide predictable latency and bandwidth [4].

In PCI Express it is related to the concept of traffic classes, among others. Up to 8 virtual channels (VC) allow establishing priorities between up to 8 traffic classes (TC), as displayed in figure 2.7. The motivation for the implementation of such mechanism is in part associated with isochronous traffic, when a constant bus bandwidth is necessary as with audio or video data.

Traffic is assigned to a TC, which is mapped to a VC upon reception at the receiver side. The maximum number of VCs is negotiated between link partners and more than one TC can be mapped to the same VC. This brings up the discussion about the flow control buffers mentioned in subsection 2.1.7, given that each virtual channel contains a separate conjunct of these buffers [5].

On the other hand, a few considerations where left aside and the PCI Express standard does not define a strict way of dealing with such cases. For instance, there are no ordering relationships required between different TCs nor between different VCs. PCI Express provides a set of rules for transaction ordering, but this only applies to transactions of the same TC moving through the fabric at the same time [5].

Arbitration is the solution used to provide the method for managing transactions and differentiated services, reordering TLPs of the same type and VC before transmission. This prevents inadvertent split transactions timeout [4]. Mechanisms for TLP arbitration are available via software intervention: strict priority, and Low- and High-priority VC arbitration, illustrated in figure 2.7; round robin, and weighted round robin (WRR) [2].

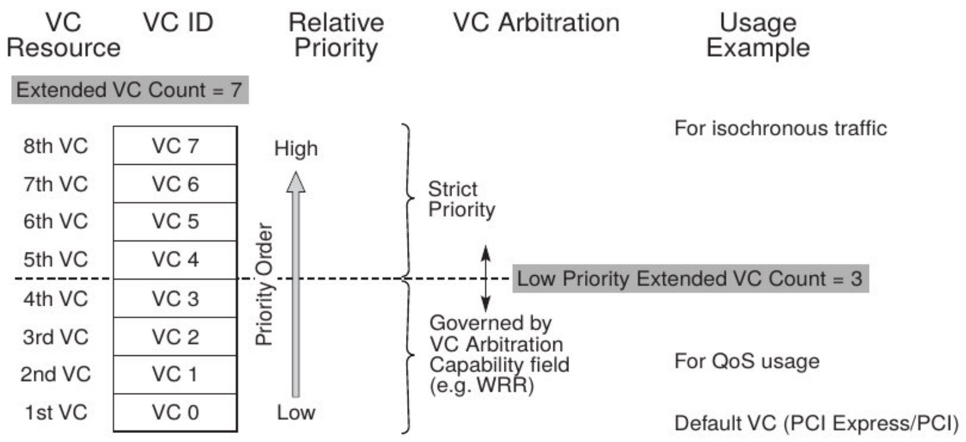

Figure 2.7: PCI Express VC arbitration example (From [2])

Wrapping things together, Quality of Service is achieved in PCI Express through the combination of several mechanisms and inner schemes. Not only it defines isochronous transactions to guarantee the requirements on bandwidth and latency for a given device, but it also comprises a set of features to configure the switching fabric according to what is desired. It is important to recall that there is not a perfect configuration scheme, and different devices/applications require different schemes.

2.2 Techniques for performance analysis

Throughout section 2.1 several mechanisms that impact performance in PCI Express were covered. Starting from the analysis of their impact, it is possible to analyze performance and determine the most critical bottleneck in the system. But before moving on let's present some literature related to performance with the intent of gathering important information about performance techniques and associated limitations.

Performance analysis starts with the utilization of one or more performance metrics. These need be chosen according to the characteristics of the system and the scope of the analysis, and latency and throughput are the most common metrics used. The first describes the time spent to complete an operation, whereas the latter defines the rate of work performed [8].

In the view of Agilent Technologies in [12], link *throughput* is the most interesting performance metric for PCI Express since its architecture is mainly directed to prioritize throughput over latency. It represents the number of bytes being transferred over the link per unit of time. Or, in a different perspective, it is the amount of time needed to successfully transmit payload bits, defined in [13].

Motivated by the high complexity and reconfigurability of the PCI Express architecture, throughput has been chosen as the metric utilized in the scope of the project. Not only measuring latency is a limited task considering a device-based analysis and the existence of posted transactions, but also throughput is directly related to the link speed and bandwidth. In fact, the values displayed in table 2.2 act as reference values for the maximum throughput possible, easing the analysis.

On the other hand, latency has been made worse in PCI Express compared to its predecessors by the trending focus on throughput and the effect of serialization, as TLPs need to be stored upon reception so that the LCRC, and optional ECRC, check may be applied to the whole packet. Similar logic occurs in the transmission side, since TLPs need to be buffered at the replay buffer [14].

When focusing on throughput, there are some concepts that need to be addressed. The first concept is the link *utilization* [12], and is determined by the following relation:

$$Utilization = LinkActiveSymbols/TotalSymbols$$

It essentially records the fraction of the total time during which the link is effectively active for transfer of data. It acts as the upper bound on the raw link bandwidth, if encoding overhead is not considered.

But even when the link is active, one must count with the *efficiency*. It defines the amount of symbols transmitted through the link if the complete link time is used. In other words, it addresses the weight of the several sources of overhead in the possible bandwidth. It is evaluated with the equation [12]:

$$Efficiency = PayloadSymbols/LinkActiveSymbols$$

The amount of link active symbols can then be expressed by [12]:

$$LinkActiveSymbols = OverheadSymbols + PayloadSymbols$$

From these concepts, a first performance estimation is applied to the link throughput in the following way [12]:

$$Throughput = MaximumThroughput * Utilization * Efficiency$$

2.2.1 Analysis of bottlenecks

The main outcome of the performance analysis carried out in this document is to indicate the main bottleneck on the transmission side and on the reception side. A bottleneck basically represents a resource, device component or underlying mechanism that limits performance [15].

The strategies to detect bottlenecks vary and depend essentially on the nature of the system. To motivate what was mentioned at the introductory section 1.2, identification and analysis of performance issues can be performed in either *offline* or *online* modes. While the first is performed after the run, the latter is performed during run-time. Considering the characteristics of the performance analysis in this project, the offline mode is utilized [8].

Having said this, the simulation run starts with a measurement phase prior to the analysis. Further commentaries on the motivations behind the chosen methodology are given in chapter 3.

2.2.2 Validation of the analysis

Another important concept to recall is the validation of the performance analysis. [8] also addresses this by presenting a list of different validation techniques. In a verification environment, performance analysis can make use of all the data that allows making assumptions about the behavior of the system. On the contrary, analyzing performance of a running system demands for extra logic within the Core, increasing area, power consumption, and consequently, cost.

As consequence of this, the designer is forced to simplify the analysis while keeping the accuracy at satisfactory levels. Hence strategies to validate the analysis are required [8]. These strategies include statistical analysis of both the measurements and the results from the analysis, techniques of machine learning and clustering, or theoretical baselines [15].

Summarizing, there many different techniques applied in performance analysis. What motivates the use of a specific metric and validation mechanisms is in great part the nature of the system, the conditions under which the performance analysis is done, and possible limitations that arise from it. This makes a large number of different approaches possible. Not all of them are indicated to the same scenarios and therefore the motivation needs to be addressed properly.

The next section closes the background chapter and goes through the work performed in the field, hence representing the State of the Art by the time of the work.

2.3 Aspects of PCI Express performance measurement

In the previous section, 2.2, some performance-related mechanisms and reference baselines have been covered. Throughout this section, these mechanisms are addressed once again with the presentation of some examples of work done in the field of PCI Express performance. There are not many experiments on performance analysis of PCI Express under the conditions defined in this document, but several studies target the topic of performance analysis and therefore many methods may still be considered [13].

The theoretical values for the bandwidth, shown in table 2.2, represent the main reference for throughput analysis and validation of the concept. In reality, the effective data rate is lower than the values displayed due to overhead, as treated before in section 2.2 [15]. Thus counting the overhead associated with each involved mechanism is essential to carry out the identification of the most critical bottleneck upon detection of an undesired throughput.

2.3.1 Data transfer overhead

Contrary to the approach to the PCI Express layers, starting at the section 2.1.2 onwards, a bottom-up approach is used here instead.

First of all, every data transfer in PCI Express includes additional overhead with the introduction of framing and by encoding the resulting data stream at the Physical Layer. This is a system requirement and cannot be adjusted. Nevertheless, PCI 3.0 doubled the speed when comparing with the older releases by reducing the encoding overhead from 20% to approximately 1.5% [15].

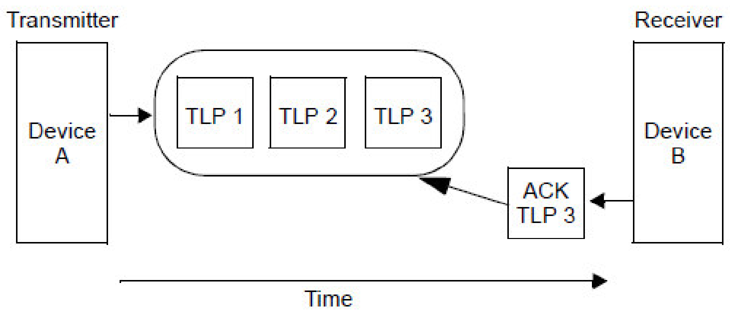

Figure 2.8: Three ACK DLLPs collapsed into one (From [6])

The second factor that contributes to the data transfer overhead is the TLP overhead, as seen in figure 2.4. In order to minimize its impact, a larger maximum payload size (MPS) can be programmed. This enables larger multiple TLP transfers of a desired size, which increases link efficiency [15].

2.3.2 Link protocol overhead

Every TLP transferred over a link is to be acknowledged, or not, by the receiver. Furthermore, the ACK/NAK protocol enables collapsing several ACK DLLPs into one, as illustrated in figure 2.8. Acknowledging the last TLP received acts as acknowledging all the previously received TLPs [15]. This is interesting in the sense that it introduces a trade-off between DLLP overhead and timeout issues.

In addition, receiving an ACK or NAK DLLPs is not an instantaneous action, and therefore must be counted when dealing with performance analysis. Besides this, there is a delay between the reception of an NAK until the replay of the TLP [12].

2.3.3 Flow control overhead

The flow control is both a crucial solution to enable reliability and performance, but is also a factor that affects performance. For instance, it mitigates the effect of discarding packets due to receive buffer overflow [15].

In fact, flow control DLLPs sent by each link partner continually update the status of the receive buffers so that the other device does not transmit too many packets. This prevents buffer overflow by adding its own overflow. A trade-off of the frequency by which this updates occur is essential and must be configured by keeping in mind the system and its application [13]. Guidelines for the frequency of flow control credits update are made available in the PCI Express specification [2].

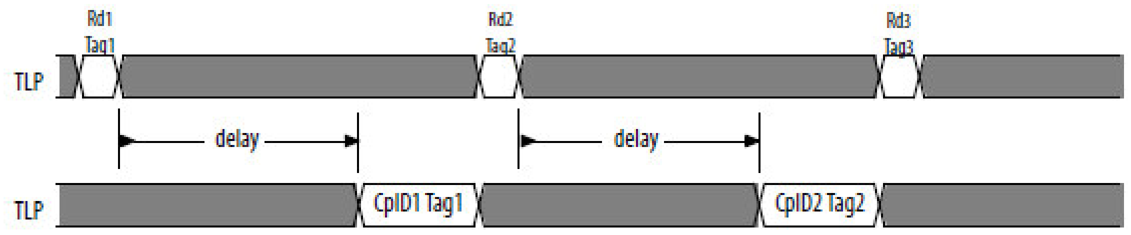

Figure 2.9: Low non-posted rate of requests (From [7])

2.3.4 A latency study

In [14] a different approach to performance analysis in PCI Express focuses on latency. Serialization delay and marshaling delay were defined and while the first is taken as purely a function of the link speed and link width, the second comprises the latency introduced by the factors mentioned in the previous subsections.

The article covers the measurement of the round-trip time (RTT) of non-posted transactions and embraces the limitation of measuring the links directly by estimating the measurement of the round-trip time at the devices on either end of the link. The minimum round-trip time is taken as the minimum value within a sample of one million measurements, as cumulative functions show there is a well defined value [14].

Also relevant for this work is the validation technique used, which is based on the comparison between the software implementation by the user and the kernel implementation. The document finishes by concluding that PCI Express has traded latency for throughput. In fact, considering the introduction of split transactions as described in subsection 2.1.6, PCI Express does not ensure that the link is free by the time the completion is ready to be returned to the requester. While the previous busses focused on achieving a small latency at the cost of a low throughput, PCI Express does the opposite. Since modern computational system grow larger and more complex, connecting more and more devices, it is preferable to prioritize throughput.

Another paper further addresses performance analysis in PCI Express by actually relating latency to throughput. In [7] it is admitted that the throughput for non-posted transactions is lower than the throughput for posted transactions. Secondly, the round-trip time for a non-posted request is analyzed by checking the non-posted request timing. According to the latency for completion and the payload size of the completions, throughput may be affected.

To exemplify this behavior, figure 2.9 shows idle periods on the receiver side, as the rate of non-posted requests is low. On another scenario, a high rate of non-posted requests originate high throughput of completions as shown in figure 2.10. In any case, the effect on the latency to completion works the other way around, as the size of the completions introduces cumulative latency.

Just a quick remark to inform that the *TAG* term used in the figures refer to an identifier which purpose is to keep track of non-posted transactions [5]. Hence the maximum number of *TAG value* limits the maximum number of simultaneous non-posted requests originated by a PCI

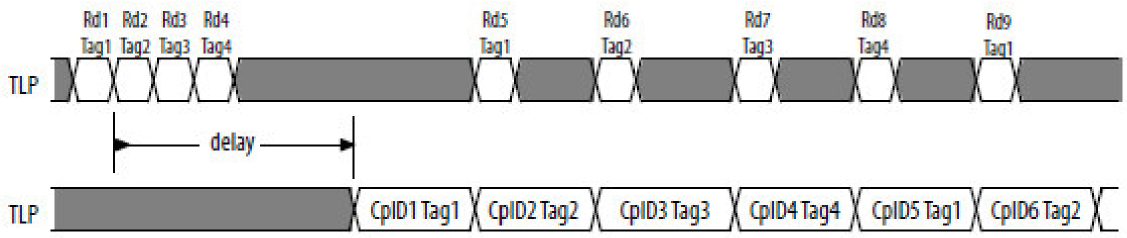

Figure 2.10: High non-posted rate of requests (From [7])

Express device. This is considered in the bottleneck analysis, as lack of TAGs may result in a loss of throughput on the opposite direction if only completions are to be transmitted.

2.4 Introduction to SystemVerilog

Before moving on to chapter 3, in which the strategy used to achieve the proposed goals is presented, it is important to bring up some aspects of SystemVerilog. However, this section does not cover all the aspects of the language; instead it aims at exposing the main characteristics of the concept. More information is given throughout the rest of the document as necessary.

SystemVerilog is both a hardware description language and a hardware verification language. It derives from Verilog, a hardware description language used to model electronic systems, essentially digital circuits. The main motivation behind SystemVerilog is to ease the process of verifying these electronic systems by introducing new features on top of Verilog [20].

Amongst the several new features of the language, two are especially useful for the project: new data structures and verification tools.

Within the first, SystemVerilog enumeration data type is widely used in this project. It essentially allows assigning meaningful names to numeric quantities, hence providing a cleaner code and enhancing readability. When the complexity of the system grows large, it is very useful to address elements in arrays.

In terms of verification tools, the introduction of classes expand the support for verification. Not only does inheritance represent an important characteristic towards achieving a proper architecture for the verification system, but also creates several levels of specialization that are useful to structure tests. Moreover, polymorphism enables handling inherited classes in a more appropriate way, by allowing the parent class to handle instances of child classes. More about this topic is covered in 4.0.1.1.

Besides the two features that were mentioned, SystemVerilog also introduces interfaces and assertions [20]. The first are useful because they allow defining the border between modules, classes or between both. Typically grant a mean of quickly defining the characteristics of an interface in terms of signals that are visible from outside the element to which it is attached. Hence it is not only useful when changes on the I/O of the element are needed, but also enable using it to connect to the testbench when the element itself or the other elements that would connect to it are not yet defined.

Assertions are useful to verify the behavior of some features of the system. They can be defined to check a given property at a single moment, or to check for sequences over a period of time. In practice, acts as a conditional statement that triggers a message when the expression being tested does not verify. The output messages can be associated with different levels of severity, which the user may use according to the impact of the failure in the behavior of the system.

To sum up, SystemVerilog acts as an extension for Verilog when hardware design is concerned, whereas on a verification level its object-oriented programming techniques are closer to that of Java or C++, by carrying typical characteristics of higher-level programming languages.

At this point, the protocols, mechanisms, concepts and the state of the art were covered and represent the background for the rest of the document. Whenever there is need for background support, a reference to this chapter is made.

Chapter 3

Methodology

This chapter covers the methodology adopted in the implementation of the performance analyzer.

The section 3.1 aims at providing further contextualization by taking a small tour through the working environment and the specification requirements for the project.

The main strategy and the justifications for the main decisions are covered in section 3.2. This is supported by opposing the adopted strategy against the alternatives and showing the motivation behind the final decision.

3.1 Contextualization and overview

The starting point for the proposed work in this document comprises the following working environment:

- PCI Express release 3.0 Proprietary Core;
- System Verilog Testbench verification environment.

Following the brief introduction to the project in chapter 1, the first subsection introduces a system level overview of the proposal and the validation environment used to implement the behavioral architecture for the proof of concept.

3.1.1 System level overview

In a general view, the possible result of the implementation of the performance analyzer is represented in figure 3.1:

A CPU, short form for *Central Processing Unit*, on the top of the hierarchy is responsible for the overall functioning of the system.

It is connected to the PCI Express topology through the Root Complex using an interface that is transparent for this specification. Furthermore, the Root Complex is connected to a series of Endpoint devices through PCI Express links and switching elements.

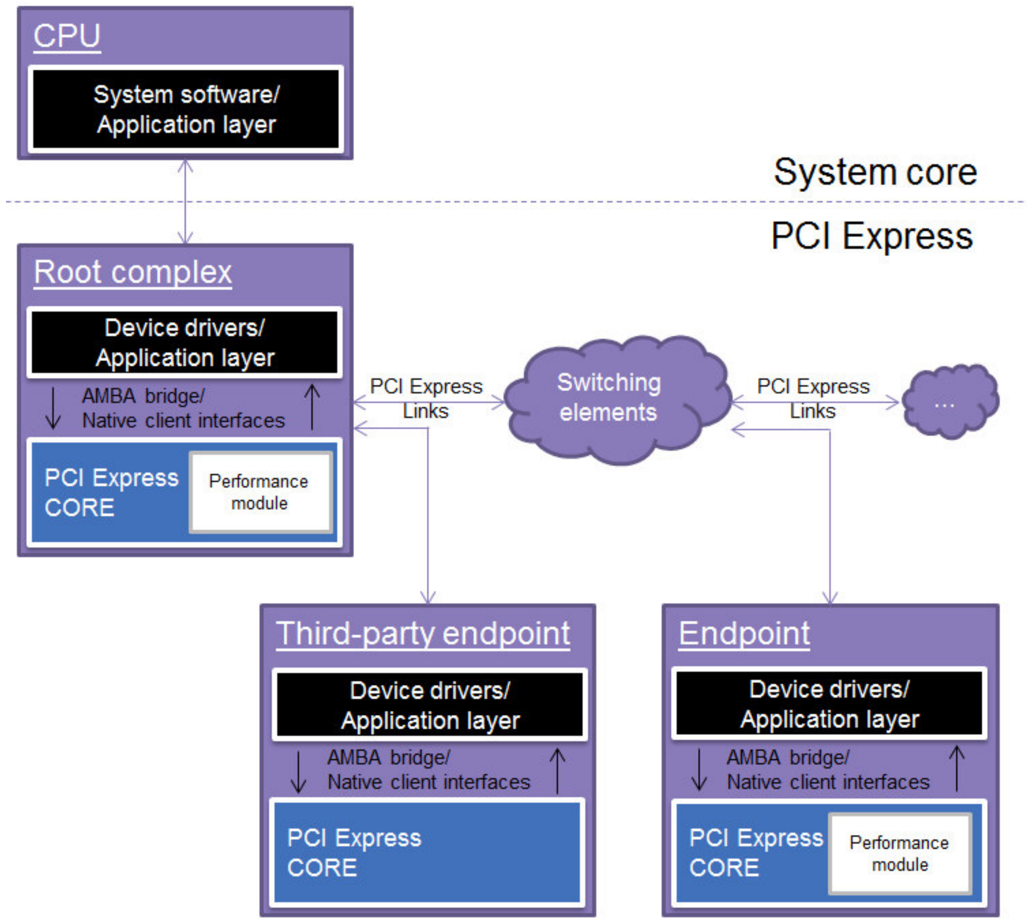
CPU
System software/ Application layer
System core
PCI Express
Root complex
Device drivers/ Application layer
AMBA bridge/ Native client interfaces
PCI Express CORE
Performance module
PCI Express Links
Switching elements
PCI Express Links
...
Third-party endpoint
Device drivers/ Application layer
AMBA bridge/ Native client interfaces
PCI Express CORE
Endpoint
Device drivers/ Application layer
AMBA bridge/ Native client interfaces
PCI Express CORE
Performance module

Figure 3.1: System level representation of the implementation proposal

Each device contains a firmware/application layer where a controller for that component resides. The interface may be left as native or implemented through a bridge. Even though it does not represent any PCI Express specified structure, this interface may have influence on performance and therefore must be considered.

Finally, to be able to analyze the performance associated with a given component, it must have its own performance module. Orders from the CPU travel through the topology in the form of PCI Express transactions, until reaching the intended device's core. These requests are directed to the configuration space of the device, allowing controlling the functionality of the performance module.

3.1.1.1 AMBA bridge and DMA

Before moving on to the validation environment in 3.1.2, it is important to take some brief notes on AMBA bridge and DMA. AMBA is the short form for *Advanced Microcontroller Bus Architecture* and essentially represents an open-standard used in the interconnection of blocks within a system-on-chip design [16]. The scope of the project covers mainly the AMBA AXI protocol.

The AXI protocol is burst-based and defines the following independent transaction channels: read address, read data, write address, write data, and write response. It works under the principle of a handshake process between master and slave, using *VALID/READY* signals, respectively [17]. A burst consists of a stream of data that is transferred without being separated into different transactions. This means the master begins each burst by driving control information and address of the first byte in the transaction. As the burst progresses, the slave must calculate the addresses of subsequent transfers.

The DMA stands for *Direct Memory Access* and defines a feature in computer systems that allows access to memory without intervention of the CPU. The motivation is to minimize the latency due to the quicker response times of the dedicated hardware, compared to the utilization of interrupts. This is usually required to achieve maximum data transfer speed, and furthermore releases pressure on the CPU [18][19].

Both technologies are usually found in PCI Express devices and may have impact in performance, reason why they are mentioned here. While the AMBA bridge data path width and transfer maximum burst size may force halting of the data traveling through the IP Core, the DMA aims at reducing the latency associated with normal memory access involving the processor.

However, these will not be considered as pure sources of bottlenecks for the system. The reason is that no access points are present at the AXI bridge, since the bridge is optional. It is true that there is an associated loss in the precision of the analysis, but on the other hand, including this access point would represent a big cost. The logic for the analysis would also have to be done differently for both situations.

3.1.2 Validation environment

Given the system level overview of the proposal, the validation of the concept for the performance analyzer is implemented within a SystemVerilog Testbench verification environment. Its representation is illustrated in figure 3.2.

The verification environment runs traffic-based simulations for the described PCI Express system according to a set of test cases, thus allowing testing for different traffic patterns. This generates traffic that is transferred through the PCI Express link. The traffic stimuli however does not need to travel through to the PCI Express topology, as in a real system both devices may initiate communication.

Configuration stimuli for the performance module is also delivered directly at the application layer of the PCI Express device. This workaround does not represent relevant impact in performance and enable surpassing some limitations. This will be brought up again when discussing the measurement phase of the proposed solution.

Considering both the point-to-point characteristics of the PCI Express link and the device-level performance analysis, a Root Complex and an Endpoint, both stimulated by the test case, share a PCI Express link. Only the device under test contains the performance module during a test case like suggested in figure 3.2. In this situation, the Endpoint is the device under test and the Root Complex acts as the link partner. The opposite situation is also possible.

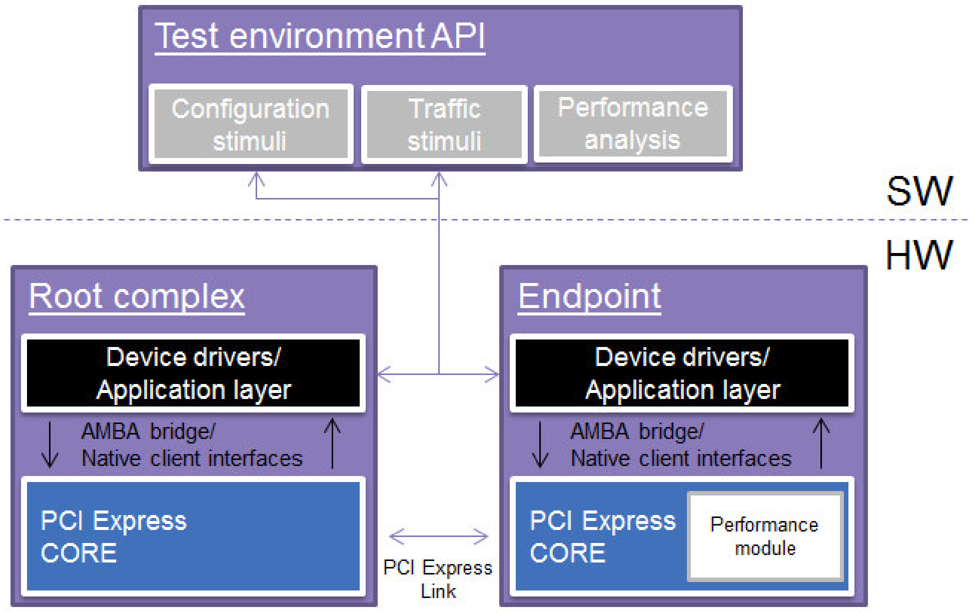

Figure 3.2: Validation environment for the proposal

Finally, the test environment API contains a set of test cases that allow testing and covering different scenarios of operation. This way one may test a particular behavior and/or force a given bottleneck to stand out. This is also important to verify the correctness of the system.

3.1.3 Operating flow

For a given test case, the test environment initiates the performance analyzer by starting the performance module of the device under test at the intended moment within the simulation. A local posted write from the application layer to the configuration space writes into the performance module configuration registers to initiate the measurement phase.

At the end of the simulation time, the test case performs another write request to terminate the analysis and further read requests get the results from the measurements.

The results are stored and used in the performance analysis.

3.2 Strategy for the implementation

This section represents the Core section of the document, expanding the solution and justifying the decisions that were made in order to accomplish the goals. Starting from the purpose of the project, described in section 1.2, the validation of the concept must put effort into implementing a solution that resembles as much as possible the behavior of a implementation in hardware.

The main limitation here is to keep the logic involved in the measurements at a low computational effort and the number of memory registers within a reasonable number. Furthermore, the

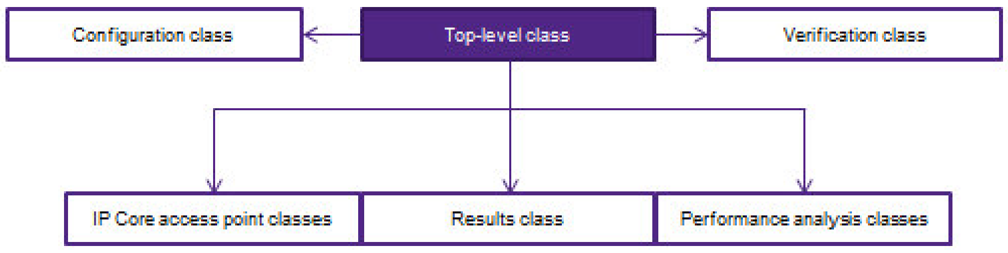

Figure 3.3: SystemVerilog infrastructure of classes for the proposed solution

implementation of an in-system performance analyzer forces the developer to admit the utilization of approximations to the metrics evaluated. For instance, in opposition to most of the studies referred in section 2.3, in which performance analysis is mainly carried out in a verification environment and evaluated during run-time, here instead, performance analysis is completed after the performance module run-time. Given this, storing all the information about traffic and timing is not scalable and therefore is not considered.

This motivates three phases of an operation run of the proposed solution. These are covered in subsection 3.2.2, after the main architecture of the solution be covered in section 3.2.1.

3.2.1 SystemVerilog architecture solution

The implementation of the behavioral solution consists in a SystemVerilog architecture based in classes. SystemVerilog is based on extensions to Verilog, a hardware description language, and expands its functionality to hardware verification with the introduction of an object-oriented architecture and SystemVerilog Assertions [20][21].

Even though bringing up the discussion to SystemVerilog is not the purpose of the section, some details may be addressed when a particular characteristic of the language is taken to allow some specific feature.

Having said this, the infrastructure created to implement the solution contains a hierarchy of classes, that in a simplified view is represented in figure 3.3.

A top-level class is responsible for both the instantiation of the other classes and the workflow of the solution. It makes use of a configuration class that loads the IP Core relevant configuration details into the solution, plus extra configurations that are specific to its implementation.

The measurement phase is performed by the execution of the IP Core access points classes logic, probing a set of signals and performing some statistics. Since the IP Core already contains some statistics that are useful not only for the solution itself but also for its validation, the access points are synchronized with this functionality.

After the run-time, the results are read from both the IP Core statistics and the IP Core access point instances and stored into the results class. Here logic applied in the results from the measurement phase give origin to other statistics that are used in the performance analysis.

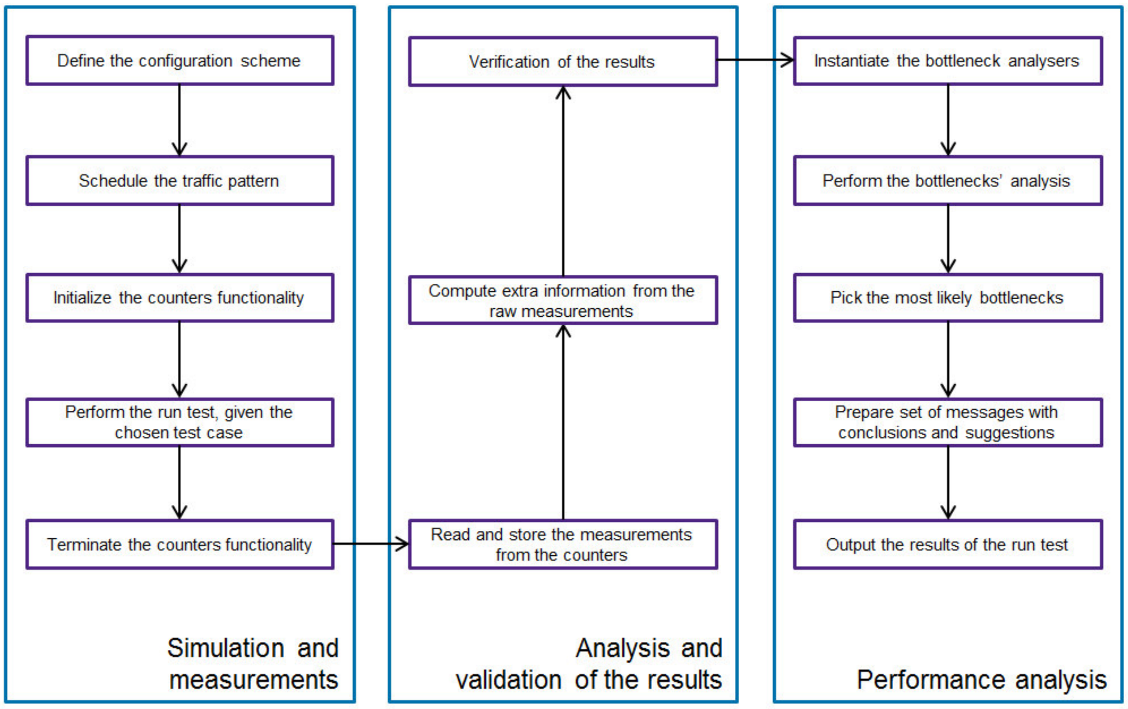
Define the configuration scheme
Schedule the traffic pattern
Initialize the counters functionality
Perform the run test, given the chosen test case
Terminate the counters functionality
Simulation and measurements
Verification of the results
Compute extra information from the raw measurements
Read and store the measurements from the counters
Analysis and validation of the results
Instantiate the bottleneck analysers
Perform the bottlenecks' analysis
Pick the most likely bottlenecks
Prepare set of messages with conclusions and suggestions
Output the results of the run test
Performance analysis

Figure 3.4: Simulation workflow used in the validation of the concept

At this stage, the verification class is called to perform the analysis of the results. A set of techniques are used to verify the correctness of the implementation.

Finally, if no harmful imperfections are found during verification, the performance analysis classes, one per direction of traffic flow, evaluate the presence of a list of bottlenecks and their impact in the current run. A final decision returns the most likely bottlenecks for the transmission path and for the receiving path.

Not represented in figure 3.3 are the classes responsible for auxiliary tasks such as the output of results and the custom test cases that were created to test both the solution and different traffic scenarios.

3.2.2 Simulation workflow

This subsection covers the simulation workflow used in the validation of the concept. It essentially consists of an extension to the description of the operating flow in the previous subsection 3.1.3 and is represented in figure 3.4.

Within the simulation workflow three essential phases are defined. The next three subsections aim at covering the main features of each phase, supporting the decisions with references to the background chapter 2.

3.2.2.1 Simulation and measurements

Given the highly configurable PCI Express IP Core, the execution of a test case starts with the definition of the configuration details. Since PCI express is backward compatible, it is possible to set the maximum generation speed possible and the number of lanes used. In addition, the interfaces and details such as maximum payload size are also considered at this stage.

The test case is also configured given a set of input parameters that include the amount of data to be transferred on both directions. By defining different traffic patterns and/or the introduction of errors in the traffic, it becomes possible to force a given bottleneck to stand out.

The next step is to start the traffic transfer and the performance module to start probing data from the IP Core. Considering the settlement period of the test case, during which the device under test and the link partner negotiate the terms for communication and initialize the link, the performance module start instant is parametrized. This allows discarding this window of time that, if considered, would have influence in the performance analysis by impacting the throughput of data and the traffic pattern recorded.

Two methods for defining the end of the performance measurement phase were made available. If on one hand the measurement phase may continue until the end of the run, i.e., upon detecting that all traffic was transferred, measuring the activity at the IP Core during a fixed and pre-configured window of time is the alternative. The motivation here is in part similar to what was addressed for the start moment of the measurement phase. Considering that one device may be finished with the transmission of traffic sooner than the other, that direction of the link will contain a significant window of idle time. This idle time makes it impossible to measure the maximum possible throughput.

By defining the start and end instants of the measurement phase, the scenario further resembles the possible physical implementation of the solution. The two definitions of throughput presented in section 2.2 are each one related to one of these alternatives of operation.

The measurements themselves are divided into types of counters, given their purpose. Three main types are defined, namely event counters, data counters and time counters. Each type of counter is further divided into groups, each one centered in a specific mechanism of PCI Express. The groups are listed in section 4.

Event counters aim at recording events of any nature, by defining error event counters and non-error event counters distributed through several groups. These take into consideration not only the number of transactions but also the events created on the behalf of the inner management protocols.

Data counters are specific to the traffic of TLPs and record the amount of data and overhead for each type of TLP at several access points. Throughput and TLP overhead can be derived from these counters.

Finally, time counters are implemented to perform two concurrent tasks. The first task makes use of a designed FSM, standing for *Finite State Machine*, at the Physical Layer, that records the amount of time the link spends in each state, for each direction. The time is actually counted in terms of bytes so that in the end the total measurement time is both computed in a time unit and in a data unit, in terms of percentage.

Through this it is possible to check the throughput result against the theoretical baselines, shown in table 2.2, at the same time it outputs a relative value for it. This relative value may provide an easier way of identifying the performance of the system and the margin for progressions. The second task is to record blocking time and halting time, enable quantifying the impact of the inner management protocols in performance, validated by the event counters.

3.2.2.2 Analysis and validation of the results

The intermediate phase of the simulation workflow performs a set of read requests to the device under test configuration space registers where the results from the measurement phase are stored. These results are read into the results class mentioned before in section 3.2.1.

Extra information is computed from the raw measurements at this stage. These include the computation of performance metrics like the relative throughput and the relative impact of the mechanisms involved in the data communication.

The verification of the results is applied after the last stage is completed. It is important to refer right from the start that the verification is limited when the measurement scenario implies a fixed window of time for measurements that is smaller than the window of time during which traffic is being transferred. This is due the techniques applied in the validation and that are covered in subsection 3.2.3.

3.2.2.3 Performance analysis

The performance analysis represents the last phase of the simulation workflow and comprises the logic through which the impact of the bottlenecks is measured. A bottleneck analyzer is instantiated for each direction of traffic flow and performs the analysis for the bottleneck comprised within its scope. The bottleneck analysis is performed in two stages.

A first stage exposes the individual logic applied to each possible bottleneck. This is done via a two-step analysis. During the first step, a maximum value for the presence of the bottleneck is determined. This value represents the percentage of the probing time during which the bottleneck may be present, and results mainly from the simulation counters. It acts as an upper bound for the bottleneck impact. The second step makes use of more information to evaluate the actual impact of the bottleneck in the test case.

This two-step analysis is necessary due to the superposition of the impact of some bottlenecks and the definition of bottlenecks directed to a specific type of data. For instance, detecting lack of posted flow control credits does not necessarily mean this bottleneck really has impact in throughput as other types of TLPs may still flow. However it does impact the throughput of posted TLPs.

The second stage of the bottleneck computation performs another analysis, starting from the results of the previous stage. It uses an algorithm especially designed for this matter, which establishes precedences between the bottlenecks and aims at finding the most critical bottleneck for the given test case. The result from this analysis is the main, or most critical, bottleneck in each direction when the simulation returned an unexpected throughput of data. As mentioned before, confidentiality reasons do not allow to expand much this part of the solution.

Finally, the last stages of the simulation prepare and output the results, the bottleneck analysis and a set of messages with conclusions about the performance of the system and possible suggestion for performance enhancement. It is in these stages that latency results are analyzed, starting from the impact of directly related bottlenecks to output a set of messages that show the behavior or a particular type of data and/or mechanism. This is addressed further ahead in subsection 4.0.1.6.

3.2.3 Validation of the concept

As mentioned throughout the document, the proof of concept for the new feature is performed in a behavioral way. Not only this prevents the introduction of bugs in the existing core, but also enables testing new features without major concerns regarding the production of synthesizable hardware description, or its representation in terms of area and power. Nevertheless, the solution is expected to take these notions into consideration to be able to prove the concept and to make a possible future implementation simpler.

Consequently, testing new features before real implementation through the use of behavioral description is faster and cleaner. The behavioral part of the system can nevertheless be connected to the existing physical components, providing ways of simulating the overall functionality of the system, even before effectively implementing all of its parts.

However, if on one hand the utilization of behavioral code is useful for debugging of the physical implementation through the process known as test benching, on the other hand, when designing new features in a behavioral fashion, test benching is also needed. Problems arise when the complexity of the system, or its nature, does not make that task simple.

In fact, the PCI Express standard is so complex and keeps expanding itself at such a high pace that typical test benching does not apply here. Having said this, throughout this section a set of strategies is defined in order to verify the correctness of the new features.

3.2.3.1 Correctness of the behavioral counters

The first step towards the addition of the performance analyzer feature and identification of bottlenecks in the PCI Express Core is the implementation of several new counters that store information on behalf of the access points during the measurement phase. These parameters complement the information available from the existing functionality, in order to enable reaching valid conclusions about the performance of the system.

The results from the counters are therefore an essential part of the whole new functionality and checking their correctness, or absence of it, is extremely important.

The first set of corrections that shall be done upon the introduction of new behavioral code is related to the code syntax. This is however the simple part, as the compiler returns some messages that help identifying where the errors are. Once the entire code is compiled and the simulation is finally able to run, a whole set of challenges regarding correctness arise, and the way of determining the correctness of the system needs to be defined.

Given this, the following list displays the guidelines used in order to validate the correctness of the behavioral counters in the proposed solution:

- Defining a set of test cases that allow testing each counter, by consisting of well-known traffic that drives the intended counters;
- Correlating the data from the behavioral counters to the data from the core counters, whenever it is possible;
- Using theoretical bounds to find situations where a given counter returns a result that is not compatible with the theory behind the protocol;
- Modeling the same counter logic with a different access point/probing strategy that, in theory, shall return the same result.

The use of SystemVerilog Assertions is important at this stage. Assertions are checkers that return a message associated with a defined level of severity when the tested condition is not verified. The condition shall be defined in such a way that it is flexible enough to accommodate the results from different scenarios, but also strong enough that allows identifying any type of error.

By following these guidelines it is expected that every behavioral counter is verified to all its extent. Ideally the verification should cover all the scenarios of test, especially since some conditions may be verified under most scenarios but still fail in some corner case. However when the measurement phase does not wait for the completion of all the scheduled transactions, the full verification is not possible as desired. For instance, without all the traffic recorded in the given counters, the correlation between behavioral counters in different access points does not pass the assertions checks.

To allow minimizing this undesired effect, a further list of assertions performs an in-system partial verification of the counters logic and the FSM, extending the verification to the access points and critical events.

3.2.3.2 Correctness of the bottleneck analyzers

Considering the bottleneck analysis feature of the solution, defining correctness in this situation is rather complex, as the performance of each bottleneck analyzer is not measured in terms of its functionality being correct or wrong. On the contrary, the process of achieving valid results from the bottleneck analyzers is based on tuning.

Based on what is known from the PCI Express standard and correlating that information with test cases specifically made to drive a certain bottleneck, it is possible to tune the bottleneck analyzer so that the results are considered valid. This is a process that may have to be performed in parallel for every bottleneck analyzer, considering that a bottleneck at a given point in the IP Core may force other bottleneck to stand out in the results.

Given this, the following points are the guidelines used in order to tune the performance of the bottleneck analyzers:

- Defining a set of test cases that allow testing each bottleneck analyzer, by consisting of well-known traffic that is expected to drive the intended bottleneck;
- Correlating the data from every bottleneck analyzer, in order to help identifying interferences between them;
- Establishing, when and if necessary, an order of precedence in which some bottleneck analyzers have priority in respect to others by the time of determining the most critical one.

This analysis is independent of the verification of the results from the counters, which is performed in the analysis and validation phase of the simulation flow. In this case, the correctness is defined in terms of the coherence of the bottleneck metrics used and the bottleneck decision at the end of the simulation.

In order to facilitate the tuning, the conditions used to determine the impact of the bottleneck and which bottleneck is the most critical contain parametrized fields. Through this it is possible to easily perform small adjustments when needed.

Chapter 4

Implementation

Chapter 3 covered the decisions made on the planning phase of the project and presented the justifications that support those decisions. The working environment and the simulation workflow were also explored.

This chapter completes the previous one by presenting the most relevant implementation aspects and the main limitations of the concept. Its purpose is to sustent the previous chapter with concrete information about the implementation of the proof of concept.

4.0.1 Main characteristics of the implementation

In sections 3.1 and 3.2 the description of the problem and the strategy that allowed its realization covered the techniques for measurement and analysis of the results.

However the main aspects of the implementation were left aside to be treated in this section. Having said this, the next four subsections give more detail about the implemented solution.

4.0.1.1 Some useful SystemVerilog characteristics

Before covering the main subsections within the implementation section, first it is important to refer some useful SystemVerilog characteristics that were considered in the implementation. An introduction to the architecture based on the language was provided in section 3.2.1, but the implementation details were left unmentioned. This subsection goes through the most interesting characteristics of the language that proved to be useful in this context. [20] is the main source of information for this part.

Firstly, being SystemVerilog an object-oriented language it allows developers to easily generalize functionality, which is not possible with Verilog modules, used mainly in design to cover a given functionality. The main characteristic of the language used in this project is inheritance. It essentially enables the possibility for the creation of new classes that extend the functionality of another class.

Figure 3.3 shows in a general fashion how the infrastructure of classes is implemented. Two main groups of classes can be identified, namely the IP Core access point classes and the performance analysis classes. Within these groups some classes are defined to take care of a specific set of tasks, which differ from the tasks carried out by the other classes within the group. Nevertheless some features rely on the same principles and therefore inheritance becomes useful.

As a matter of example, all IP Core access point classes must start/finish their functionality, as well as access a register address space to store the results from their measurements. This means this common functionality can be described in a common class, which is not instantiated as an access point itself, but instead acts as a base class for the specific access point classes, defined ahead in section 4.0.1.2. By defining this inheritance, it is possible to quickly make changes in the common functionality, without accessing each extended class individually.

On top of this, SystemVerilog further enables handling the extended classes in a very comfortable way by allowing polymorphism. It basically defines the possibility of using an array instance of the super class, the class from which others are inherited, to handle the extended classes altogether. The following piece of code demonstrates a simple example where two child classes extend a parent class, and an array instance of the parent class is used to handle them.

```
program class_inheritance_polymorphism_example;

class parent;
  // Methods prototypes
  function void start_action;
  function void stop_action;
  virtual task action;
endclass

class child1 extends parent;
endclass

class child2 extends parent;
endclass

// Program main
initial begin
  parent handler [2];
  handler [0] = child1.new ();
  handler [1] = child2.new ();
  // Main body
  foreach (handler[i])
    handler[i].start_action;
  foreach (handler[i])
    handler[i].stop_action;
end

endprogram
```

By analyzing the example program, one may easily notice that handling a big set of child classes becomes much easier if this strategy is used. However one must also take into consideration that there is an extra detail to be concerned about: parallel computing. In fact, the example code shown above does not ensure a correct behavior if parallel computing is not used.

Prior in the document it has been stated that each access points must perform its task while traffic is flowing. This means each access point main task, defined above as *action*, shall run in a separate thread, while the main thread of the program is responsible for the traffic flow and simulation control. By defining so, it is now clear why the parent class implements *start_action* and *stop_action*. However the modifications made in the following code are necessary to the overall good functioning of the example program. Notice that all the code shown before is not duplicated here for simplification.

```
class parent;
  bit action_on;
endclass

class child1 extends parent;
  function void start_action;
    this.action_on = 1;
    fork begin
      this.action;
    end join_none
  endfunction

  function void stop_action;
    this.action_on = 0;
  endfunction

  task action;
    while (this.action_on){
      // Method logic
    }
  endtask
endclass
```

With the modifications presented above, the method *start_action* calls a new thread with the command *fork...join_none*, which runs the statements within it on a separate thread while the parent process continues its logic without waiting for the completion. At some later stage, the main process calls *stop_action* to stop the execution of *action*. A flag *action_on* is used in the process.

This strategy is used to both control the access point classes and the bottleneck analyzers. Handling a great number of classes involves less effort and adding new functionalities becomes simpler as shown ahead in 5.2.1.

Finally, another important SystemVerilog characteristic that is widely used throughout the implementation is the possibility to define *singleton* classes. These specify classes that can only be instantiated once, at the first call, and every further call will return the instance previously instantiated. In a complex system, composed by a great number of classes, where many classes may need access to a given class, using this strategy eliminates the need for complicated argument passing between classes and/or methods.

The most relevant example of this on the current implementation is related to the results class. This class, amongst other features, contains the array that is used to store the information collected during the measurement phase. It is accessed by the IP Core access points as each one contains a set of counters. In order to prevent each IP Core access point to instantiate an individual instance of the results class, it is the top-level that performs the first call. Each IP Core will later make a similar call, but the same instance is returned. The following example code illustrates this with a simple example program.

```
program singleton_class_example;

class singleton;
  singleton _instance;
  int id;

  function singleton get_instance ();
    if (_instance == null)
      new ();
    return _instance;
  end

  function new ();
    _instance = singleton::new ();
    this.id = 1;
  end
endclass

// Program main
initial begin
  singleton A, B;
  A = singleton::get_instance ();
  B = singleton::get_instance ();
  // A.id = B.id = 1
end

endprogram
```

Using this methodology, the *singleton* class contains a member that is a variable of its own type. The function *new* is only called when this variable, here called *_instance*, is yet to be initialized. In the example, two instances *A* and *B* of the class *singleton* are called by the program.

However, when *B* is instantiated, the method returns the previous instance. The result is that both *A* and *B* contain a handler for the same instance, hence the last comment.

4.0.1.2 IP Core access points

The first phase of the simulation workflow, treated in 3.2.2.1, presented the necessity of accessing the IP Core through a series of access points. This is especially directed at recording enough information so that executing the performance analysis in the later phases is possible.

Initially, addressing the measurement phase on a counter basis may be considered adequate. It allows an independent control over each measurement logic. However when things are scaled to more complex scenarios, handling each counter individually becomes inefficient. As a matter of fact, the counters that were implemented have in many cases similar logic and therefore it is more adequate to handle them together, reducing the computational redundancy. For this matter it is important to recall that each counter needed to be run in its own thread so that the behavioral implementation in classes resembles a possible physical implementation. This needed be considered as the sequential characteristics of a software flow would not suit the problem.

This led to the decision of designing the measurements in a access point basis, inside which a set of counters are updated by different logic for different measurements. This eliminates a lot of redundancy while keeping the possibility of controlling the activity of the counters or the activity of the access points. The addition or modification of features is also facilitated by this approach, as this modularity makes the implementation scalable.

The created access points are illustrated in figure 4.1. Each access point contains the logic for a given set of counters according to the following responsibilities:

- **XAPP** and **RAPP** — application layer throughput in both directions, further detecting the DMA transactions if active;
- **XTL** and **RTL** — transaction layer access points, responsible for keeping track of the flow control credits utilization for both directions of communication;
- **XDL** — data link layer access point, observes the state of the replay buffer and pending of DLLPs in the Tx side;
- **XPL** and **RPL** — physical layer access points, responsible for most of time counters logic and in which the FSM runs. The detection of TLPs per type is also performed in these access points.

From the list above, it is clear that there are two groups of access points designed to measure throughput: one at the interface between the IP Core and the application layer, and another at the Physical Layer. This enables probing the data traffic at the two ends of the PCI Express layered architecture within the device. The redundancy in computing the throughput at both groups provides a mean of detecting drop of packets within the IP Core, as well as packet retransmissions and packets originated or targeted to the configuration space of the Core. It also enables validating the

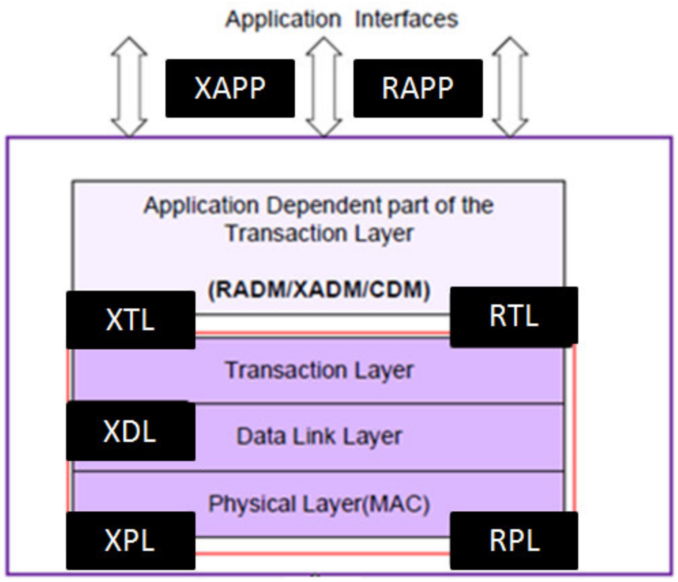

Figure 4.1: IP Core access points used at the measurement phase

correctness of the system using simple test cases that do not contain any of the events mentioned in the previous sentence.

Even though the detection of DMA transactions does not necessarily represent useful information for the performance analysis in terms of throughput measurement, it is considered here for statistic matters as the current functionality of the IP Core does not contain this.

The other three access points main purpose is to record events that somehow have impact in performance, and quantify their impact by using time counters. This strategy allows storing enough information to realize the performance analysis at the same it keeps the logic rather simple.

4.0.1.3 Physical Layer Finite State Machine

One of the most distinctive features of the implementation is the creation of a Physical Layer Finite State Machine to quantify the relative amount of time during which the link spends in each state, for both directions. The states are displayed in table 4.1.

The FSM runs on a byte basis, checking each lane sequentially. In fact, it was not mentioned before in chapter 2 but when several lanes are active, sequential data is striped over by the available lanes, from lane 1 to lane X. Based on this it is possible to identify the correct sequence of data as if only one lane was used. The result is that the logic for the FSM remains unaltered when the number of lanes is changed.

While the rest of the implementation of the proof of concept remained as simple as possible, aiming at validating a physical implementation with reduced cost, the FSM is rather complex. The motivation here is that the functionality of most IP Core signals available to probe was not clear and, especially at the Physical Layer, did not provide sufficient information. The introduction of the FSM at the Physical Layer allowed identifying the utilization of the link on a byte basis. This

FSM state	Description
TLP	The link is occupied with the transfer of Transaction Layer Packets
DLLP	The link is occupied with the transfer of Data Link Layer Packets
PLP	The link is occupied with the transfer of Physical Layer Packets
IDLE	The link is in IDLE state
Unknown	The link state is unknown before the FSM is synchronized

Table 4.1: Physical Layer Finite State Machine

can easily be simplified for a physical implementation by discarding the detection of PLPs, among others, since their occupation of the link was concluded to be typically small.

Nevertheless for this proof of concept, it was decided that the entire link time be identified and assigned to a data type and/or mechanism. Having said this and based on table 4.1, the detection of the TLP state includes the identification of the DLL and PL overheads, prefixes, header and payload; the DLLP state includes the identification of the framing tokens and DLLP type; the PLP includes the detection of the PLP type to further identify link training or simply SKIP transmission; and finally, the IDLE state is simply the time during which the link is not being used.

Through this, it is possible to analyze the impact of the bottlenecks as their effects will in most cases have impact in the results of the FSM. For instance, a great amount of time spent in the transmission of PLPs for link training is translated into the conclusion that link training is impacting the system by the given factor.

The FSM starts in the unknown state and remains there until it synchronizes, after which it shall not return to the initial state unless the link is brought down and the FSM restarts. This leads to the necessity of specifying a minimum window of time, such that the relative value of the Unknown state is minimized and its impact in the overall measurement phase minimal and therefore not relevant. In any case, the bottleneck analysis considers the relative time of the Unknown state, returning a warning if this value exceeds a certain threshold.

4.0.1.4 Groups of counters

In subsection 3.2.2.1, the types of counters utilized were covered, as well as the motivation for their use. Nevertheless, their further division into groups of counters was left uncovered and is now brought up.

Like mentioned before, event counters represent the most basic unit of measurement used in the proposed solution. They are grouped into 9 groups, which are displayed in table 4.2.

Most of the IP Core existing functionality associated with the measurement phase is within this type of counters and therefore they are important for the validation of the results from the measurement phase. In the real system, these counters are mainly important to record critical

Layer	Group	Description
PL	Per-Lane error	Physical Layer per-lane error event counting
PL	Common-Lane error	Physical Layer common-lane error event counting
DLL	Data Link Layer error	Data Link Layer error event counting
TL	Transaction Layer error	Transaction Layer error event counting
PL	Per-Lane non error	Physical Layer per-lane non error event counting
PL	Common-Lane non error	Physical Layer common-lane non error event counting
DLL	Data Link Layer non error	Data Link Layer non error event counting
TL	Transaction Layer non error	Transaction Layer non error event counting
APP	DMA non error	DMA non error event counting
APP	APP non error	APP non error event counting

Table 4.2: Event counters groups

events such as errors that should not occur given any circumstances, or provide an easy, initial view over what was captured during the measurement phase.

In terms of the throughput measurement and bottleneck analysis, the event counters do not act directly in the involved logic but are used in some places as a mean of validating the conclusions. For instance, a difference on throughput between Physical Layer and Client Interfaces at the Rx side may seem immediately resultant from dropped TLPs. However, TLPs may also be consumed by the IP Core if they target its configuration space or use an alternative interface, directed for configuration and messages, to access the Application Layer. Upon this situation, the event counters can help identify which situation is more likely to have happened.

The data events are focused in the Transaction Layer traffic, realizing the necessary data measurements to allow measuring throughput and overhead associated with the different types of TLPs and the access points. The groups of data counters are shown in table 4.3.

These counters record the amount of bytes recorded per TLP type, divided in prefix, header and payload. Once again, this is mainly useful for statistic purposes as most of this is not considered for the bottleneck analysis. In fact, most of the bottleneck analysis could rely on three data counters: prefix, header and payload. The decision of expanding this to all the TLP types and DMA was based on costumer's requests.

The time counters are stored in four groups and listed in table 4.4. These counters address

Layer	Group	Description
PL	TLP data	Physical Layer TLP detection and data counting per type
APP	DMA data	DMA data counting
APP	APP data	APP data counting through the client interfaces

Table 4.3: Data counters groups

the link utilization and the encoding overhead, carried out by the FSM group. The blocking time counting is performed by another group of counters and create the measurements that allow connecting an unexpected link utilization with the source of the problem.

The next subsection finally lists the considered bottlenecks and exposes a concise explanation of the logic involved in their identification.

4.0.1.5 Bottlenecks and identification methodology

Arisen from the fact that PCI Express is extremely complex, a lot of mechanisms have direct, or indirect, influence in performance even if at different scales. For the scope of this document, the bottleneck analysis is limited to the main bottlenecks, i.e., the ones that impact performance in a severe way. If, on one hand, these bottlenecks are the only ones whose identification may be possible given the characteristics of this implementation, on the other hand, limiting the analysis to these bottlenecks represents a sufficient approach for most situations.

In reality, the infinite possibilities for different scenarios of test flow make it necessary to first address the main bottlenecks and find ways of minimizing their effect or the conditions under which they become present in the system. The following table, 4.5, shows the possible bottlenecks that the analysis may return for the Tx side and for the Rx side:

Most bottlenecks are common to the Tx side and the Rx side, even though the logic used for the identification may differ, as it is the case of the lack of FC credits. As a matter of fact, the analysis for the Tx side is simpler and more accurate than the analysis for the Rx side. First of all, the DUT does not have access to the link partner's signals and therefore can only estimate the reason why the throughput at the receiving side is unexpected.

Starting precisely with the lack of FC credits, the DUT contains information about the number of available credits at the link partner by knowing the initial values and by receiving DLLPs that update these values. Upon the intention of transmitting a TLP, the DUT checks whether it has got enough FC credits for that TLP type that allows it to transmit the TLP over the link.

As mentioned in subsection 2.1.7, the FC credits are divided into header credits and payload credits. Furthermore, considering the wide range of payload length possible for a TLP, the number of necessary credits may also differ a lot. This means that the number of available credits by itself

Layer	Group	Description
PL	FSM time	FSM time counters for each state
PL	Encoding overhead	Encoding overhead time counters for $8b/10b$ and $128b/130b$
DLL/TL	Blocking time	Blocking time counting for the involved mechanisms
n/d	Latency	Latency counting for non-posted RTT and latency through Core for TLPs

Table 4.4: Time counters groups

does not alone indicate whether the device is lacking of enough credits to transmit. The solution was to implement logic that identified when a given TLP was requesting access to the link and checked for halting due to lack of FC credits.

At the Rx side, this is not possible and the workaround consisted in recording and updating the average payload length of the incoming packets of a given type and comparing it against the available FC credits the DUT advertised. A parametrized threshold was used to define the ratio that indicated there could be that the link partner was lacking of a given type of FC credits. During the bottleneck analysis, the value of the presence of this bottleneck can be reduced considering the link utilization and the throughput for that type of TLP.

Turning to the Tx side, another important bottleneck is related to the replay buffer. Here two situations may occur: either the buffer gets full and all TLP traffic is halted from entering the link, or the buffer is being used and TLPs from the upper layers are halted at the arbiter.

The first situation typically results from timeout issues related to the reception of ACK/NAK DLLPs from the link partner. While the DUT does not received any of these, it will continuously add new TLPs to the buffer as they are sent over the link. Eventually the buffer gets full and the transmission of TLPs needs to be halted. The second situation arises from the reception of NAK DLLPs, meaning that there was a problem with the reception of TLPs at the link partner. Here the reasons may be failure when checking the LCRC, for example.

Given the complexity of the current IP architecture, it was not possible to access the replay buffer and identify the moments when it is granted access to the link. Therefore the identification for its use relies on the difference between throughput at the Client Interfaces and throughput at the Physical Layer, supported by event counters that indicate the number of replied TLPs and the blocking time counters that indicate halting at the upper layers.

In the Rx side, this bottleneck is not considered directly. In fact, it can be associated with the bottleneck *TLPs dropped*, as some of the reasons why the TLP is dropped may lead to retransmission, such as LCRC errors. In any case, it is sufficient to the DUT to consider the TLPs dropped, as it does not represent throughput at the receiving Client Interfaces. The way of identifying the

Tx side	Rx side
Replay buffer full Replay buffer use	TLPs dropped
No relevant bottleneck	
Excessive TLP overhead	
Lack of Posted FC credits	
Lack of Non-Posted FC credits	
Lack of Completion FC credits	
Link training	
All TAGs used in the opposite direction	
Not sufficient TLP data	

Table 4.5: Possible bottlenecks

loss of throughput is again by comparing the difference between throughput at the Physical Layer and throughput at the Client Interfaces, supported by event counters.

On a normal simulation, some bottlenecks do not stand out if no errors to the specification are introduced. In fact, this occurs for the following cases: Replay buffer full/use and TLPs dropped. To surpass this condition, there are methods within the verification environment that allow introducing some specific errors on traffic, to precisely test their possible effect in a real running system. This technique is used to make the mentioned bottlenecks appear.

Not mentioned until this stage, the impact of TLP overhead and the link training on performance is directly identified from the results returned from the measurement phase. This means that the post-simulation processing for the associated bottlenecks is almost nonexistent.

Finally, when no evident bottlenecks are encountered and yet the throughput is unexpected, the only reason is that both devices are not scheduling an enough amount of data. This is validated by looking at the simulation time during which the transmitting Client Interfaces lack of TLPs to transmit and the FSM state for the Rx side is IDLE.

Like mentioned before, latency is not considered in terms of bottleneck. It is nonetheless taken into consideration in output messages, thus proving the user with more information about the behavior of the system under the given circumstances. The next subsection explores the latency measurements carried out in this solution.

4.0.1.6 Latency measurement

Given the characteristics of the implementation, the main concern when measuring latency is to make the measurements scalable and their results sufficient to make assumptions about the performance of the system. While this is trivial in a verification environment because recording latencies for individual TLPs is possible, in a physical implementation this is not scalable. The strategy defined here involves therefore a set of statistics such that it becomes possible to understand the behavior of the system, at the same time the cost of the implementation remains low and relatively constant.

But first of all, it is necessary to define which latencies need to be measured. Ideally one would be interested in measuring the latency between all the layers within the IP Core, for each TLP traveling through it. However, this solution involves a reduced number of access points, thus making this impossible. In addition, the previous paragraph automatically discards this possibility. Nevertheless, considering that halting occurs mainly at the interface between the Transaction Layer and the Client Interfaces, it is possible to record the following latencies per TLP type, Posted, Non-Posted and Completion:

Tx — Latency between Client Interfaces and Physical Layer;

Rx — Latency between Physical Layer and Client Interfaces.

Furthermore, the latency between the moment when a non-posted request is issued and the moment when the correspondent completion is returned is important to define the responsiveness of the system.

Once again, let's recall the information provided in 2.3.4. Considering that only non-posted transactions are requested in one direction, it shows that the latency between a request and its completion depends on factors such as the rate of non-posted requests and the size of the completions. This may lead to the introduction of idle periods between consecutive completions when the rate of non-posted requests is low, lowering throughput. However latency is rather constant as the idle period ensures there is immediate available bandwidth for transmission. The effects are reverted when the rate of non-posted requests is increased.

On the other hand, the size of the completions also have influence in latency, because it will take longer to transmit them when the size increases. Once again, a greater throughput and better TLP efficiency is ensured when bigger completions are issued, whereas latency is compromised as completions need to wait for available bandwidth to have access to the link.

Based on these premises, the following latencies are measured at both the Physical Layer and the Client Interfaces:

Tx — Latency between outbound non-posted request and correspondent inbound completion;

Rx — Latency between inbound non-posted request and correspondent outbound completion.

On the transmission side, measuring latency at the Physical Layer between XMLH and RMLH aims at identifying the round-trip time of the transaction from the moment when it leaves the IP Core until the moment when it arrives; measuring latency at the Client Interfaces allows identifying the complete round-trip time and further identify the latency introduced by the IP Core itself.

On the receiver side, measuring latency at the Client Interfaces captures the time spent by the Application to process an inbound non-posted request and provide a completion, whereas measuring latency at the Physical Layer aims at identifying the round-trip time of an inbound non-posted transaction from the moment it arrives at the IP Core until the moment the completion is sent over the link.

Now that the latency measurement are defined, it is time to address the chosen statistics, which are shown in the following list:

- Total latency time — sum of all the entries for a given latency;
- Maximum latency time — maximum latency time for a given latency;
- Minimum latency time — minimum latency time for a given latency;
- Median latency time — median latency time for a given latency.

Assuming straight away the limitations derived from this solution, the mentioned statistics allow keeping a reasonable record about the behavior of each latency.

From the total latency time it is possible to derive the average latency. Even though this does not give a real perception about the distribution of the measurements, especially when the distribution is not linear, it acts as a reference value. Similarly the maximum and minimum latency become references for the best- and worst-case values, at the same time the range defined by these two values carries information about the range of latencies.

The most important statistics in such conditions is the median latency. It provides a better knowledge about the distribution of the measurements in a more robust way compared to the average latency. The main obstacle here is that all the measurements are necessary to compute the exact value for the median. Since this would go against the premises mentioned before, the solution was found in [22].

In this article it is provided an algorithm for the computation of quantiles without storing all the observations, in fact, only stores a predefined constant number of last observations. A p-quantile defines the value on a distribution under which $100p\%$ of the distribution lies. In such sense, the median is the 0.5p-quantile, which means it returns the value under which 50% of the distribution lies.

By being able to compute the median latency, it is possible to determine if most latency measurements were closer to the minimum latency or to the maximum latency.

4.0.2 Limitations

As mentioned throughout the document, the solution described here embraces a strategy that enables a possible future physical implementation. Hence the limitations that affect a performance analysis in an hardware implementation with the mentioned characteristics are adopted in this proof of concept.

The first limitation is related to the necessity of maintaining the number of counters at a reasonable number so that the number of gates in a real implementation does not represent a huge overhead to the system. By keeping this values low, a small area is ensured and with it reduced cost. The main problem with this is that the information recorded must be well selected so that a reduced amount of logic still enables the analysis.

Nonetheless, the analysis on the receiving side will always be the worst-case scenario, since information local to the link partner is not visible to the DUT. Therefore the analysis of bottlenecks must be different in each direction.

Other limitations arise from the environment itself. These comprise the lack of control on the ordering of TLPs, motivating the decision made at 3.1.2 associated with the configuration stimuli. Also no control in ensured on the collapsing of ACK DLLPs, making the analysis of the margin for progression difficult to address.

Chapter 5

Project outcomes

Given the detailed description of the strategy and implementation aspects, respectively in 3.2 and 4, this chapter presents the main outcomes of the solution. Hence it expands the introductory section 1.4 into four sections.

5.1 Throughput measurement and bottleneck computation

The implemented solution accomplishes the goals referred in the first chapter, by defining a mean of measuring throughput and other statistics and further analyzing the data from the measurements to carry out the performance analysis. This section makes use of three examples to show the type of outcome of the implemented system. The tools used are displayed in the following list:

- **Working environment** — Unix with IceWM and Windows 7;
- **Code development** — Vim;
- **Waveform analysis** — Discovery Visual Environment (DVE).

Having said this, the conditions of the test cases for the first example are the following:

- Traffic pattern consisted of 20 posted and 20 non-posted transactions in each direction;
- Maximum payload size and TAG are set to the maximum available (4096 bytes and 256) and the size of the transactions are randomized;
- One test per PCIe 1.0, PCIe 2.0 and PCIe 3.0;
- One test per number of lanes: x1, x2, x4, x8 and x16;
- A window of time of $50ms$.

PCIe Gen	x1 Link	x2 Link	x4 Link	x8 Link	x16 Link
PCIe 1.0	1.92	3.76	7.04	10.32	8.08
	1.84	3.68	6.72	10.16	11.28
PCIe 2.0	3.76	6.88	9.04	6.96	8.08
	3.76	7.60	7.68	12.96	9.28
PCIe 3.0	7.36	8.96	7.60	10.00	11.52
	6.88	7.60	7.76	8.04	9.92

Table 5.1: Throughput results for the first example (in Gb/s)

This example comprises a general test case where random traffic composed by non-posted, posted and completion TLPs flow in both directions. In such conditions, the results obtained for throughput at the Physical Layer are shown in table 5.1 in the format Tx/Rx.

By analyzing the results, it is easily visible that the throughput is expected when x1 link is used, when comparing to the theoretical references shown in table 2.2. Since this is the least demanding situation possible for both ends of the link, the receiving queues never get full. In fact, the rate at which the devices process the incoming data can increase with the rate at which data is received. Furthermore, the flow of DLLPs is enough to replace the flow control credits.

It is when the number of lanes increases that problems begin to arise. More lanes mean more theoretical bandwidth available through the link, but does not necessarily mean that the devices can process the incoming data at such rate. Moreover, according to the arbitration rules within the IP Core the flow of DLLPs may be compromised and the replacement of flow credits may not be fast enough.

For a better idea of what happened in this example, the table 5.2 shows the bottleneck decision for the test runs.

PCIe Gen	Tx	Rx
PCIe 1.0	No relevant bottleneck for x1 and x2; Lack of P credits for x4, x8 and x16.	No relevant bottleneck for x1 and x2; Lack of P credits for x4, x8 and x16.
PCIe 2.0	No relevant bottleneck for x1; Lack of P credits for x2, x4, x8 and x16.	No relevant bottleneck for x1 and x2; Lack of P credits for x4, x8 and x16.
PCIe 3.0	No relevant bottleneck for x1; Lack of P credits for x2, x4 and x8; No sufficient TLP data for x16.	No relevant bottleneck for x1; Lack of P credits for x2, x4 and x8; No sufficient TLP data for x16.

Table 5.2: Bottleneck results for the first example

Looking at the bottleneck decisions, it is clear that the analysis done previously is correct. No relevant bottleneck was found when x1 link is used, independently of the PCI Express generation. When enough traffic is scheduled in order to fill the window of time for the measurement phase, throughput shall not be impacted by anomalies when such random case is tested.

However, as the number of lanes increases, the bottleneck analysis return different results. Let's analyze the results for each generation.

For generation PCIe 1.0 and PCIe 2.0, increasing the number of lanes results in lack of Posted credits. This means that as the transmission of TLPs is faster, the receiving side of each device cannot process the TLPs quickly enough and must at some point indicate to the link partner that it needs time to do it before receiving more TLPs.

However, looking at the PCIe 3.0, when x16 link is used, insufficient number of scheduled TLPs is the reason why throughput is not higher. In fact, this case continues facing the same issue as the other generations, with lack of Posted flow control credits, but the higher available raw bandwidth ensures that most traffic be transferred before the end of the simulation time. In such case, the analyzer detects that the lack of flow control credits cannot justify by its own the loss of relative throughput, and detects an IDLE period in the link that can only be explained by the lack of TLP data to transfer.

The second example explored in this section aims at identifying the impact of different values for the maximum number of TAGs, as well as the impact of different payload sizes of the completions on latency. The conditions of the test cases are the following:

- Traffic pattern consisted of 200 non-posted transactions on the Tx side;
- No prefixes allowed, no ECRC;
- Request size set constant to 128 and 4096 Bytes;
- PCIe 2.0, x1 link, 64-bit address space;
- Maximum number of simultaneous outbound TAGs set to 32 and 256;
- Simulation time: waiting until all traffic is transferred.

Before analyzing the results, it is clear that the number of TAGs set to 256 does not affect this scenario, since only 200 non-posted requests are issued. On the other hand, by making this value 32, it is expected to see an impact in throughput and/or latency. This further depends on the request size, as shown in the latency study, at section 2.3.4.

The bottleneck analyzer results for this example are shown in table 5.3. This only comprises the bottleneck analysis for the Rx side, as no throughput of data is transmitted on the Tx side:

By looking at the results, when the number of maximum TAGs is set to 256 and request size is set to 4096 bytes no relevant bottleneck is found, which is in accordance to what was expected. When the request size is reduced to 128 bytes, the result is different due to the ratio between header bytes and payload bytes.

		Request size	
		128	4096
TAGs	32	All TAGs used on Tx side	No relevant bottleneck
	256	TLP overhead	No relevant bottleneck

Table 5.3: Bottleneck results for the second example

In fact, each completion carries a 12-byte long header and therefore the associated overhead when the payload equals 128 is $12/128 = 9.38\%$. On top of this, the additional overhead from the Data Link Layer and Physical Layer makes the total TLP overhead equal $(12+2+2+4)/128 = 20/128 = 15.6\%$. In reality, this means that the link utilization, as defined in chapter 2, is expected because the Link is being used to transfer TLPs; however it also means that the link efficiency has been affected considering that the overhead associated with TLPs is considerable.

Now, if the maximum number of TAGs is reduced to 32, the results are also interesting. When the request size is 128 it is easily understandable that there is an associated loss of throughput on the Rx side. This actually resembles the situation shown in figure 2.9. A combined low rate of non-posted requests on the Tx side with a small request size introduces idle periods on the Rx side. However, when the payload size is increased to 4096 bytes, even though the rate of non-posted requests stays low, the scenario actually resembles the situation shown in 2.10.

The reason why this happens is because the large length of the completions compensates the small rate of non-posted requests. However it is important to recall that if the rate of non-posted requests decreased even further to lower values, idle periods would start to occur.

In respect to latency, this scenario is interesting because it clearly shows that an overall good throughput does not necessary result in good responsiveness. The table 5.4 contains the results associated with the latency measurements from the simulations.

		Request size	
		128	4096
TAGs	32	Small, constant latency	Great, constant latency
	256	Wide range of latencies	Wide range of latencies

Table 5.4: Latency results for the second example

When TAGs equal 32 and payload remains small, the idle periods introduced between consecutive completions make it possible for completions to have immediate access to the link. If the payload size is increased, then at the beginning of the simulation there is a settling period during which the round-trip time of consecutive non-posted transactions keeps increasing due to the length of the payload of the completions, which prevents the following completions from having access to the link. Since the maximum number of outbound non-posted requests is rather low, this settling period quickly gives origin to a permanent state where the round-trip time stabilizes and stays constant at a large value compared to the latency for the first transaction. Typically, the first transaction sets the minimum latency possible.

Moving on, when TAGs equal 256 and size remains small, the distribution of the latency values is measured over a wide range. Since the number of scheduled non-posted transactions for this test case was set to 200, they are all issued at once by the DUT. This results in that the link partner receives these requests in a small period of time and then needs to answer them back. Since the size of the completions, even at 128, is greater than the size of the non-posted requests, the latency measurements have an increasing distribution with time. Basically, this resembles the settling period mentioned in the previous case. Just a note to recall that running this case with x1 link has influence. If the number of lanes was increased to x16, each completion would take only two cycles to transmit over the link, as each cycle can be used to transmit $8bytes * 16lanes = 128bytes$. Considering the overhead from the header and lower layers, two cycles would be enough. In such conditions, the queue at the link partner to transmit the completions would be clear much faster and the maximum latency would be decreased.

Finally, in the last case, an even wider range of latencies is recorded due to the increase in the payload length. In this case, not even by increasing the number of lanes would the improvement in latency be noticeable.

The third and last example addresses the reception of TLPs that contain errors, namely LCRC errors. As mentioned previously, this case needs the utilization of extra functionality from the verification environment to enable introducing errors into the transferred traffic. The test case then consists in simulating a set of tests that introduce LCRC errors with different probability. The conditions for the test case are the following:

- Traffic pattern consisted of 200 posted transactions on the Rx side;
- No prefixes allowed, no ECRC;
- Payload size set constant to 128 Bytes;
- PCIe 2.0, x1 link, 64-bit address space;
- Probability of error: 0%, 10%, 40% and 90%;
- Simulation time: $30ms$.

In this situation, the identification of a performance anomaly on the Rx side cannot rely only on the throughput measured at the Physical Layer. In fact, the throughput at the RMLH should not verify any substantial loss. On the contrary, throughput at the Client Interfaces is expected to verify a significant loss, as the LCRC checker drops TLPs that do not pass the check.

Upon a transmission of a NAK DLLP on the Tx side, the Link Partner proceeds with the retransmission of the TLP that contained errors. A further mechanism defines that the Link shall enter retraining when the same TLP cannot be successfully transferred in a few tries.

Given this initial analysis, the results from the simulation are displayed in tables 5.5 and 5.6. The throughput results are shown in *PL/APP* form and as a percentage of the total simulation time.

Error probability	Tx	Rx
0%	0	0.83
	0	0.76
10%	0	0.84
	0	0.50
40%	0	0.83
	0	0.20
90%	0	0.81
	0	0.00

Table 5.5: Throughput results for the third example

From the throughput measurements, it can be seen that the main impact of the introduction of errors on the inbound TLP traffic is that the throughput at the Client Interfaces drops consistently. Considering the scheduled traffic, such discordance between the throughput measured at the two locations acts as the first hint for a situation where TLPs are dropped upon reception. The analysis also comprises the evaluation of flags that indicate the reception of errors, thus allowing the validation of the diagnosis of the bottleneck *TLPs dropped*.

Error probability	Tx	Rx
0%	No sufficient TLP data	TLP overhead
10%	No sufficient TLP data	TLP overhead
40%	No sufficient TLP data	TLPs dropped
90%	No sufficient TLP data	TLPs dropped

Table 5.6: Bottleneck results for the third example

Looking at the throughput results, no meaningful changes are verified at the Physical Layer. On the other hand, throughput at the Client Interfaces is clearly affected when the probability of receiving TLPs with LCRC error increases.

Analyzing the bottleneck conclusions, some interesting facts are clear. When the probability of introducing LCRC errors on TLPs is 0%, the conclusion is that no relevant bottleneck is affecting the system. In fact, there is some TLP overhead resultant for the small constant TLP payload size, but it does not compromise the throughput in a very harmful way.

Once the probability is set to 10% some TLPs are already dropped and the throughput at the Client Interfaces is slightly affected. However, the uncertainty associated with the Rx side does not allow the algorithm to relate the loss in throughput with the loss of TLPs. In such case, the algorithm also detects a rather excessive TLP overhead, which combined with the loss of TLPs by LCRC error results in a considerable loss in throughput and therefore assigns TLP overhead as the main bottleneck.

When the probability rises even further the algorithm is able to associate the loss of throughput with the LCRC errors. As a final remark, when the probability of LCRC error on the Rx side is 90%, the Link actually enters retraining and some output messages indicate this action. However this is not shown as the final conclusion because it is mainly a consequence of the high number of LCRC errors.

The presented examples shown in this section are not meant to show the entire functionality of the system. Instead of doing so, they are meant to explore three test scenarios that one may easily comprehend from the background presented in chapter 2. Both scenarios are rather simple to analyze and tuning the bottleneck analyzer for these situations was not complicated.

The next sections, 5.2 and 5.3, intend to mention what the contributions in terms of created infrastructure and support for configurations are. A last, brief section 5.4 shows some details about the dimension of the created solution and the tools used for the implementation.

5.2 Created infrastructure

The infrastructure created for the solution includes the implemented SystemVerilog class architecture and the test cases that were created to treat the bottlenecks considered previously. A set of different configurations in which the test cases shall run is also added so that the results of the test cases are coherent.

This relation between the test cases and the configurations is necessary because changing the number of lanes and/or the speed of operation can lead to completely different scenarios of test. Since it was not practicable to create a test case to test a given bottleneck for every single configuration, some example pairs are used to exemplify the scenario and validate the results.

To support this, let's recall the results returned in the first example case shown in the previous section. Using the given test cases with the same configuration resulted in that the maximum throughput will never be reached by the faster releases, as the amount of traffic is not enough for the given configuration. Considering this, either the test case is made in accordance to the configuration, or the configuration is modified.

Nevertheless a custom test case is also available to quickly launch a simulation of a new scenario of test. This test case allows defining the following conditions:

- Characteristics of the measurement phase: 1) Fixed window of time; 2) Entire window of simulation;
- Pick from an existing test case (may still modify some parameters), or create a custom test case;
- Specify amount of traffic and size of traffic per type and direction;
- Specify maximum/minimum payload size;
- Introduce errors in the traffic, by defining the probability of occurring the error.

This test infrastructure aims at easing the process of creating new scenarios as well as adding new functionalities. Its modular characteristics in terms of classes make it easy to add new counters for the measurement phase, new access points and new analyzers. This subject is expanded in subsection 5.2.1.

Furthermore, the documentation provided with the solution covers all the implementation details and adds a guide for the introduction of new functionalities. It comprises a specification of the solution, divided into three main chapters: functional specification, implementation specification and verification methodology. Some ideas for a possible implementation in hardware are also mentioned within the specification. This does not intend to be a strict decision; instead its purpose is to present the conclusions derived from the results of the proof of concept.

5.2.1 Guide to add new functionalities

Taking into consideration that PCI Express is continuously involving, essentially meaning that the IP Core needs to evolve accordingly, special attention was given to the infrastructure of the system. Not only is it expected to comply with the current requirements, but also it must be versatile enough to be adapted to newer characteristics when required.

Some of the characteristics of the implemented infrastructure were already covered in the previous chapters. However no information on the addition of new functionalities was provided. The purpose of this subsection is to give a brief overview of the actions to be performed in that direction.

In a general way, the infrastructure in SystemVerilog classes is designed so that the addition of new access points and/or new counters, as well as new logic for the performance analysis be a simple and fast process. It takes advantage of the SystemVerilog potentiality covered in section 4.0.1.1, which mostly enables handling a great number of class instances in a simplified, adequate way.

Given this, the following abstract steps define guide for the addition of new functionalities into the solution:

- Create and code new class, if new IP Core access point is needed, or add new methods into the existing bottleneck analyzer classes for intended functionality;
- Add new entry on the configuration files in the form of *define* or *enumeration*;
- Add configuration parameter to allow enabling/disabling feature;
- Add entry to the respective common class handler at the top-level class of the solution.

From the list of steps defined above it is clear that only a small number of steps are required to add new functionality. In reality, a template for new IP Core access point class is available, containing the inheritance characteristics and the prototypes for the main methods.

In order to make the code legible, IP Core access points and their counters are defined in terms of *enumeration typedef*. This basically allows addressing each location within an array by using a string, hence easing the process of reading code. For debug purposes a set of configuration parameters, in the form of *define* statements, are used to determine which functionalities are enabled/disabled.

Finally, the top-level class is responsible for controlling the whole functionality of the solution and contains the class handlers. If a new class is to be added, it must be instantiated here. IP Core access point classes can simply be added to the common handler and their functionality is automatically controlled by the previously mentioned mechanism.

5.3 Configuration support

Given the highly reconfigurability of the PCI Express standard and the further necessity for configuration support brought by the previous section, one extra focus of the implementation was to ensure support for as many configurations as possible. A proof of concept for a performance analyzer for use in PCI Express must, up to a certain extent, aim at making its use possible through the various configuration scenarios.

This represented at some points a very challenging task, as the Core signals, Core width, frequency of operation and Core modules suffer changes when different configurations are selected.

Considering this, the following characteristics are supported:

- PCIe 1.x, 2.0 and 3.0 link speed;
- Link width x1, x2, x4, x8 and x16;
- AMBA AXI bridge or Native Client Interfaces;
- DMA module presence or absence;
- Speed changes and Link width changes partially supported.

The referred configurations are high priority and therefore were addressed in the scope of this project. Further configurations are not top-priority and therefore may be left out of the contributions. They may be addressed at a later stage if needed.

5.4 Some implementation statistics

Before closing this chapter, this last section shows some statistics related to the implementation of the solution. On one hand, this intends to show the complexity associated with PCI Express, even when many aspects of the specification were left uncovered in the scope of the project; on the other hand, this also makes it possible to have an idea of the computational effort involved in the implementation.

Given this, the following list displays a first set of statistics related to the implemented solution. This is mainly directed to the size of the implementation in terms of code.

- Number of SystemVerilog created files: 23 class files, 2 configuration files and 1 test case file;
- Number of lines of code: $\approx$ 10000;
- Number of further edited files: 10 VTB API files, 1 testlist file.

Furthermore, the time spent for simulation vary a lot depending on the window of time of the measurement phase and the amount of traffic scheduled. Based on the first example shown in section 5.1, and as a matter of example, the time spent for simulation is around 30 minutes with the proof of concept active and around 22 minutes with it inactive, for Gen1 and x1 link. It is however important to consider that the jobs are run remotely at a farm and therefore the time spent depends on the utilization of the server.

In any case, the computational effort of the solution is increased to such extent mainly due to the exhaustive byte-based analysis during the measurement phase, especially heavy due to the FSM at the Physical Layer. For instance, for a given simulation window of time, a x1 link test case runs faster than a x2 link test case because of the ratio $1/2$ in the number of simulation bytes to consider.

By turning these access points off, the simulation time is reduced and the computational overhead of the solution is minimal. The motivation for this exhaustive implementation was related to the fact that many approximations had already been introduced when assuming a hardware-resembled implementation of the proof of concept. Given this, the measurement phase relies essentially in probing the data path, making sure the timing and accuracy of the measurements are preserved to their maximum extent.

Another alternative can easily be implemented if necessary by making use of sideband signals present at the core. Initially this was not done because some timing issues between them seemed to give origin to imprecise measurements. Approaching the end of the implementation it was concluded that they do not have a big impact on the overall functionality of the system. Some of the modifications are currently being done at the time of the release of this document.

Chapter 6

Conclusions

This document was expected to demonstrate a concise description of the work performed during the semester, in an enterprise environment.

The topic of performance analysis in PCI Express was covered in a consistent way and supported by the overview on the PCI Express system architecture. Due to the extent of the PCI Express standard, many other topics were left aside and others were mentioned here but not explored deeply. The main key point however is that the proof of concept for the performance analysis of the PCI Express standard using a device-level approach was carried out.

The performance analysis is a field that faces a trade-off between the margin of gain and the cost of the analysis and enhancements to reach this gain. Moreover, many technologies such as PCI Express expand truly fast, which carries further difficulties when performance is to be analyzed. In fact, not many studies have been carried out in the topic of in-system solutions for performance analysis, as most are merely used in verification environments or with the aid of external devices that perform the analysis in a controlled environment. Having said this, there are many concepts that represent limitations in this field.

Nevertheless, it is believed that the contents most related to performance were covered and, in the end, the main factors and parameters of measurement of PCI Express performance were explored. The main mechanisms that impact performance are considered in the bottleneck identification and the final decision and therefore a further analysis would only be necessary if identifying smaller bottlenecks at a different scale was part of the scope.

In this case, the scope left unaccounted the bottlenecks originated by failures in the link negotiation or by the PCI Express topology for the given system in which the device is located. Nonetheless, with the perspective for a future implementation in a real system, the amount of PCI Express devices that contain the performance module completed with the proposed solution increase the knowledge about the system and enables analyzing the performance at a system-level as well.

Even though the proof of concept developed here accounts for both the performance analysis and the measurement of statistics such as DMA traffic and TLP detection per type, introducing extra counters, a real future implementation can discard these so that the implementation cost is minimized. Of course there are some fundamental counters in the system and these cannot be left unaccounted.

The future work that can be performed on top of the implementation described here is listed below:

- Further validation/verification of the concept;
- Continuous tuning of the bottleneck analyzer for more accurate results;
- Possibility of the implementation of machine learning techniques to enhance the previous point;
- Model and code a solution based in this proposal;
- Add the new functionality to the IP Core.

A further validation of the concept is necessary to ensure the system does not contain any undesired effects. This also contains tuning the bottleneck analysers so that better accuracy is achieved, which can be explored via machine learning techniques.

Finally, a proposal for the implementation in hardware shall pick between the proposed logic and select what is in fact necessary for the implementation.

References

[1] Eirc Esteve. IPnest: Interface IP Survey 2005-2016, September 2012.

[2] PCI-SIG. *PCI Express® Base Specification Revision 3.0*, November 2010.

[3] Daniel Gallant. Choose The Optimum Clock Source For PCI Express Applications. *Electronic Design*, May 2012.

[4] Inc. Mindshare, Ravi Budruk, Don Anderson, and Tom Shanley. *PCI Express system architecture*. Addison Wesley, first edition, 2008.

[5] Inc. Mindshare, Ravi Budruk, and Mike Jackson. *PCI Express technology - Comprehensive Guide to Generations 1.x, 2.x, and 3.0*. First edition, 2012.

[6] Freescale Semiconductor Inc. *MSC8156 and MSC8157 PCI Express Performance*, November 2011.

[7] Altera. *PCI Express High Performance Reference Design*, December 2014.

[8] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. Performance Anomaly Detection and Bottleneck Identification. *ACM Computing Surveys*, 48(1), July 2015.

[9] Inc. Mindshare, Don Anderson, and Tom Shanley. *PCI system architecture: Fourth edition*. Addison Wesley, fourth edition, 1999.

[10] P. Ramanathan, A.J. Dupont, and K.G. Shin. Clock distribution in general VLSI circuits. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 41(5):395–404, May 1994.

[11] M. Dinesh Kumar. Implementation of PCS of Physical Layer for PCI Express. Master's thesis, National Institute of Technology, Rourkela, 2007-2009.

[12] Agilent Technologies. *PCI Express performance measurements*, September 2006.

[13] *Modeling and Performance Analysis of PCI Express*, 2014.

[14] David J. Miller, Philip M. Watts, and Andrew W. Moore. Motivating Future Interconnects: A Differential Measurement Analysis of PCI Latency. ANCS'09, October 2009.

[15] Xilinx. *Understanding performance of PCI Express systems*, October 2014.

[16] ARM. *AMBA AXI and ACE Protocol Specification*, 2011.

[17] Xilinx. *AXI Bridge for PCI Express v2.5*, November 2014.

[18] National Instruments Corporation. *DMA Fundamentals on Various PC Platforms*, April 1991.

[19] Jonathan Corbet, Alessandro Rubini, and Gre Kroah-Hartman. Memory Mapping and DMA. *Linux Device Drivers*, January 2015.

[20] Accellera. *SystemVerilog 3.1a Language Reference Manual*, May 2004.

[21] Doulos. *SystemVerilog Golden Reference Guide*, August 2012.

[22] Raj Jain and Imrich Chlamtac. The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms Without Storing Observations. *Communications of the ACM*, October 1985.