Diplomarbeit

Design and Implementation of a Bus-Switching Unit for a High-Speed Peripheral Card

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von

O. Univ. Prof. Dipl.-Ing. Dr. techn. Dietmar Dietrich
und
Univ. Ass. Dipl.-Ing. Dr. techn. Thilo Sauter
und
Dipl.-Ing. Patrick Loschmidt
als verantwortlich mitwirkenden Assistenten

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Computertechnik, Inst.-Nr. E384

von

Felix Ring
Matr. Nr. 9925202
Bachackergasse 35, 2380 Perchtoldsdorf

Perchtoldsdorf, 7. Mai 2009

Kurzfassung

Diese Diplomarbeit befasst sich mit der Weiterentwicklung einer Ethernet Netzwerkkarte für hochgenaue Uhrensynchronisation. Um die Genauigkeit der Uhrensynchronisation bis zu Standardabweichungen im Subnanosekundenbereich verbessern zu können, soll die Eignung von neuen Übertragungsmedien für Uhrensynchronisation untersucht werden. Die vorliegende Arbeit beschreibt die Entwicklung einer dafür notwendigen Netzwerkkarte, die die Durchführung von Uhrensynchronisation sowohl in optischen als auch in kabelbasierten Gigabit Ethernet Netzwerken ermöglicht. Als Grundlage dafür wird eine bereits verfügbare Hardwarearchitektur für hochgenaue Uhrensynchronisation verwendet, die auf einer Evaluierungsplattform für optisches und kabelbasiertes Gigabit Ethernet implementiert werden soll. Da nur Evaluierungsplattformen mit PCI Express Anbindung erhältlich sind, muss die bisherige PCI basierte Hardwarearchitektur in eine PCI Express basierte Hardwarearchitektur übergeführt werden. Dieser Umbau erfordert die Entwicklung einer Einheit, die unterschiedliche Bussysteme miteinander verbinden und deren Pakete auf die verschiedenen teilnehmenden Knoten entsprechend verteilen kann. Es handelt sich dabei um mehrere AHB Knoten, die mit dem PCI Express Bus verbunden werden müssen.
Das vorliegende Dokument behandelt ausführlich die Anforderungen für eine solche Einheit und die dabei auftretenden Hürden, die es zu meistern gilt. Im Zuge dessen werden gut erforschte Techniken für vergleichbare Fragestellungen aus unterschiedlichen Anwendungsgebieten beleuchtet. Die dort gewonnenen Erfahrungen werden auf den vorliegenden Fall angewendet, und darauf basierend werden verschiedene Lösungansätze dargestellt. Desweiteren werden während der Entwicklungsarbeiten aufgetretene Schwierigkeiten beschrieben und analysiert, um daraus Hilfestellungen für zukünftige Projekte abzuleiten. Die aus den gewonnenen Erfahrungen erwachsenen Vorschläge für strukturiertes Vorgehen sollen helfen, mögliche Entwicklungsfehler zu vermeiden. Schließlich wird beschrieben, wie zwei auf grundsätzlich unterschiedlichen Ansätzen basierende Simulationsumgebungen verwendet werden, um durch die Durchführung automatischer und halbautomatischer Tests die entwickelte Einheit zu verfizieren. Als Ergebnis dieser Arbeit können durch den Einsatz mehrerer Einheiten der entwickelten Hardware Testnetzwerke aufgebaut werden, die ein tieferes Verständnis von hochgenauer Uhrensynchronisation in Gigabit Ethernet Netzwerken ermöglichen.

Abstract

This diploma thesis is about the evolvement of a high-precision clock synchronization enabled Ethernet Network Interface Card (NIC). To further improve the precision of the clock synchronization system to sub-nanosecond standard deviation, investigations on the suitability of new media for clock synchronization have to be made. The present work describes the necessary development of an evaluation network interface card for clock synchronization over copper- and optical fibre based Gigabit Ethernet. Therefore, an existing hardware design for high-precision clock synchronization has to be implemented on an evaluation board supplying Gigabit Ethernet physical layer connectors. As such evaluation boards are only available with PCI Express connectors, the existing Peripheral Component Interconnect (PCI) based system has to be transferred to a PCI Express based system. The transformation requires the development of a bus-switching and translation unit to interconnect multiple instances of Advanced High-Performance Bus (AHB) interfaces to the PCI Express connection interface. This document provides a thorough explanation of the requirements and challenges of such a unit, investigating theoretical approaches and utilising knowledge of similar problems in different, well understood application fields. Moreover, traps and pitfalls that came up during the design and implementation process are presented and analysed to derive hints and suggestions for future projects, in order to use the gained experience to optimise the design methodology and avoid mistakes. The design is verified by using two developed, fundamentally distinct simulation environments and applying a variety of test cases for automated and semi-automated verification. The resulting system is a powerful means to gain deeper knowledge of high-precision clock synchronization in Gigabit Ethernet based networks by setting up and measuring test networks using multiple instances of the developed hardware.

Acknowledgements

A lot of people contributed to this diploma thesis by encouraging words, helping hands and good ideas here and there, and even more people contributed to the long way it took me to get to this point of my studies. It is not possible to mention all but a small number I am grateful for. Still, honour, to whom honour is due. Most of all I am grateful that God, whom I may call my father, brought me this far. After him, I sincerely have to thank my parents for providing everything I needed to finish my studies. I also want to thank all my dear friends who supported and encouraged me during this long time, before all others my long-time study companion Lukas Riegler for almost adopting me in his family during the many hours spent studying together.

I am also very grateful for all of my colleagues at work, who also supported me during the development of this diploma thesis, and who helped me out with ideas and advice in one or the other fruitful discussion and conversation. For my supervisor Patrick Loschmidt, I would need an extra page to thank him for all he has done for me, not only during the writing of this thesis, but also for providing a superb working environment together with Georg Gaderer and Thilo Sauter, and for holding my back free to be able to concentrate on writing my thesis. I also want to specially thank my supervisors for correcting and evaluating the thesis on such short notice, even late at night. Further thanks go to my grandmother, Josefine Thurner, and my friends, Ilse und Andreas Schmaranzer, for letting me retreat to their houses for quiet and concentrated writing, and to my aunt Judith Thurner for lending me her car to get there. Finally, my special thanks for editing, go to my dear friend, J. Steven Ramey, my brother, Stefan Ring, and to the most wonderful person I have ever met, Leslie Kidd, who I want to thank for more than I can ever express.

Contents

Chapter 1

Introduction

Moore's law [1] has accurately predicted the development of the complexity of integrated electronic systems for approximately the last forty years [2]. Moore's prediction was the doubling of the complexity of Integrated Circuits (ICs) every two years, which proved to be a very good estimation of the past development. This rapid increase in complexity allows the integration of more and more functional components up to complete systems on a single chip. Not only because of the increase in complexity, but also because of more efficient and faster transistor and circuit designs computational power increases even faster than the mere number of transistors on a chip. In order to use the available high computational power, most applications demand high performance data connections to provide and receive the data that is to be processed.

Despite the possibility to integrate a high number of functions on a single chip, there will always be a need for additional peripheral components which have to be connected to the main unit. The advantages of peripheral components are rooted in the advantages of modularity, which are mainly flexibility, changeability, and reusability. To allow several modules to interact with each other and with the main unit, a communication system is required. In order to satisfy the needs for constantly increasing data rates, the interconnection systems for peripheral components have to be continuously adapted and improved. Therefore, scalability is a very important requirement to keep the need for architectural changes of the interconnection systems at a minimum. In recent years a transition from Peripheral Component Interconnect (PCI) systems to the faster and highly scalable PCI Express (PCIe) systems has taken place in the area of consumer PCs.

The steady development of more complex ICs with higher processing power and higher interconnection data rates continuously straightens the path to new application fields. The improvement and evolvement of industrial electronics, especially together with the trend to distributed computing and control, leads, amongst others, to the requirement for high-precision clock synchronization. In networking systems a strong trend towards packet-switched networks can be observed. Therefore, during the last years many efforts were taken to provide high-precision clock synchronization over packet-switched networks,

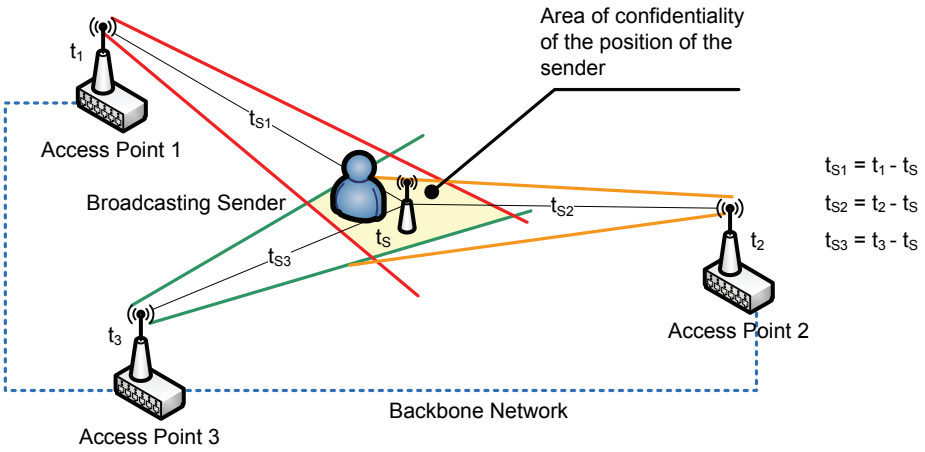


Figure 1.1: Basic localisation concept

e. g. over Ethernet. A dedicated working group was created at IEEE to define a standard for clock synchronization, the IEEE 1588 standard [3].

1.1 Motivation

All of the above considerations together with the desire for wireless communications are driving key factors for the FIT-IT project Embedded Position Determination and Security in Wireless Fidelity networks (ε-WiFi). The aim of the project is to offer position information to services in Wireless LANs (WLANs) without the need to modify the node to be located.

To show the basic idea of localisation in wireless networks, figure 1.1 depicts a standard approach to localisation by measuring the signal propagation delays from a mobile client node to geographically distributed, positionally defined access points. With these measured propagation delays and the knowledge that WLAN packets travel at the speed of light the distance of the mobile node to each Access Point (AP) can be calculated. With the known distances, in the two-dimensional case the mobile node can be localised via triangulation.

Position Detection

In ε-WiFi a more sophisticated approach for localisation has to be used, but the basic idea stays the same. The ε-WiFi project aims at localising any standard mobile node with a WLAN interface. No specific hardware, software or drivers are installed on the mobile nodes. Therefore, the transmission time of the packet originating at the mobile node cannot be known, since no clock synchronization can be implemented with appropriate accuracy. And even if the clock was accurate, without a dedicated software the node does

not include the transmission times in its transmitted packets. To still be able to perform localisation the Time Difference of Arrival (TDoA) of the WLAN packets has to be used. The mathematical background for a technique using TDoA for localisation is described in [4].

The concept is based on the idea that all wireless signals are inherently broadcast transmissions. Therefore, all access points receive the same signal at a different instant of time, depending on the distance of the mobile node to the respective access point. As the access points can communicate with each other, the time difference, and only the time difference of the arrival of a packet at two distinct access points is known. With two access points this calculated difference corresponds to the difference of the distance of the node to the first access point ($d1$) and the distance of the node to the second access point ($d2$). This known difference ($d2 - d1$) is the only position determining information available. Mathematically, in two-dimensional space, a function with a constant difference of the distance of a variable point to two distinct focal points is a hyperbola, as depicted in figure 1.2. In the three-dimensional space from the time difference of arrival of the packet at the two different nodes a hyperboloid with the access points being the focal points can be drawn. The mobile node can be anywhere on the hyperboloid, as the difference of the distance of all points on a hyperboloid to the focal points is constant. When a third access point is added, the node can be determined to be on the intersecting ellipses of the hyperboloids. A fourth access point is needed to limit the node's location to two points, and five access points are needed to get a unique point (as long as the access points are not all on the same plane).

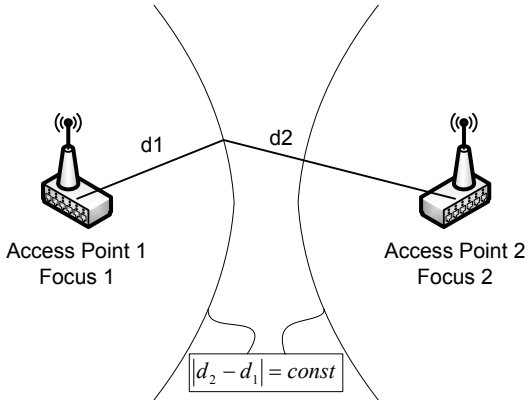


Figure 1.2: Hyperbola defined by two access points

To get accurate localisation results, the measured differences of time have to be very precise. In order to achieve such high precision, the clocks of the access points have to be well synchronized. The clock synchronization of the access points is performed over the wire-based backbone network. Therefore, the project consortium of the ε-WiFi project aims at further improving the precision of this wire-based synchronization. As previous projects led to a deep understanding and remarkable results in the field of clock synchro-

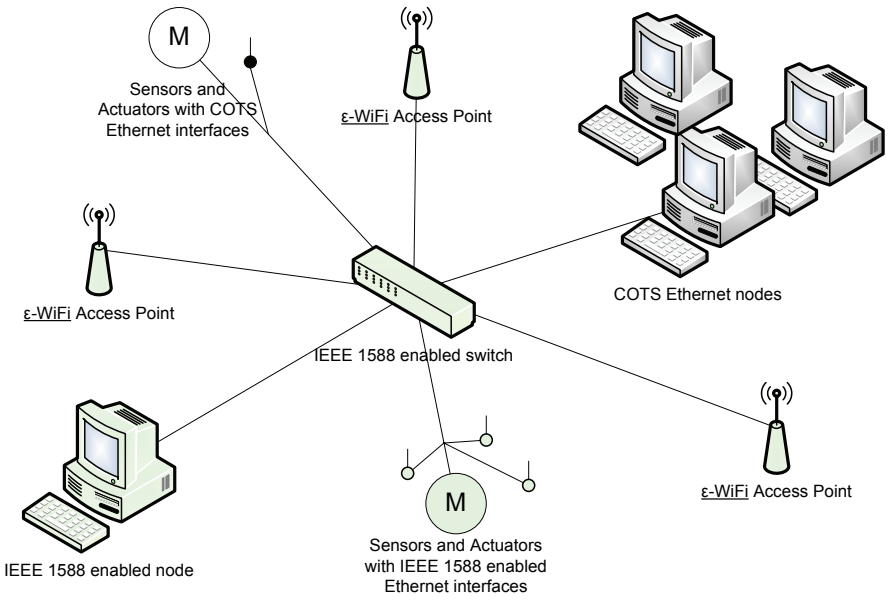


Figure 1.3: Example network for clock synchronization

nization over Fast Ethernet (100 MBit/s), new transmission media and technologies are investigated for their suitability as base-networks for high-precision clock synchronization.

High-Precision Clock Synchronization

Figure 1.3 shows several Ethernet nodes which are interconnected in an example network. The nodes in light green are equipped with Ethernet Network Interface Cards (NICs) that are specifically designed for high-precision clock synchronization, while the other nodes have Commercial Off-The-Shelf (COTS) Ethernet network interfaces. The interconnecting Ethernet network is a standard network with no limitations. The clock synchronization enabled NICs are capable of participating in normal Ethernet activities, and additionally use standard Ethernet frames for communicating their specific messages needed for clock synchronization. Those messages are part of the Precision Time Protocol (PTP), which is a protocol for high-precision clock synchronization defined in the IEEE 1588 standard. Therefore, the specific nodes in the figure are also labeled "IEEE 1588 enabled". The PTP uses Ethernet as a base-network to transmit specific messages for clock synchronization. It is described in more detail in section 2.1.5. Undisturbed Ethernet traffic and high-precision clock synchronization can co-exist even in an existing COTS Ethernet network.

For Ethernet-based clock synchronization, a crucial factor is the accurate delay measurement of packets as they travel from the source node to the destination node. The high-precision clock synchronization enabled network nodes perform such measurements at the Media Independent Interface (MII) level (the connection between the Ethernet Physical Layer (PHY) and the Ethernet Media Access Control (MAC) unit during normal operation). Any jitter of these measurements, as well as any asymmetric, direction

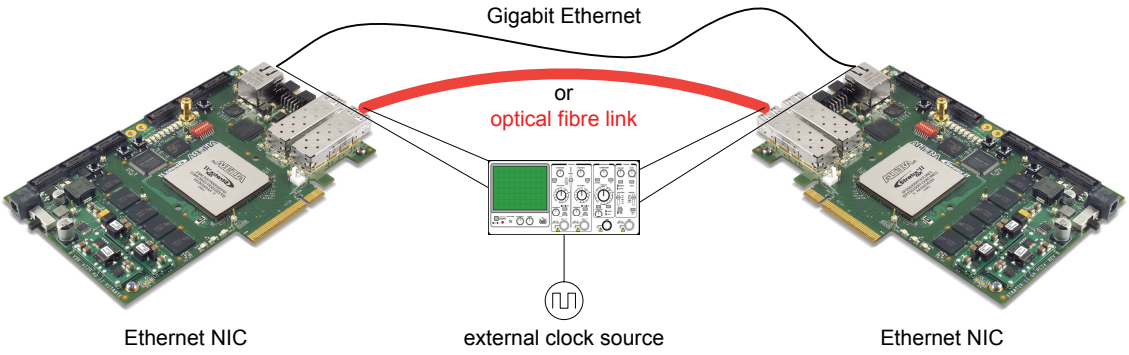


Figure 1.4: Example setup for delay measurements

dependent delay difference deteriorates the precision of the clock synchronization system. To get an impression of the precision which can be expected, the jitter and asymmetric delay properties of Gigabit Ethernet over copper and optical fibre links have to be investigated.

Required Measurements

Figure 1.4 shows an example setup for direct-link measurements concerning the mentioned delay and jitter properties. The used Ethernet NICs are not required to support clock synchronization, any network card with physically accessible MII signals can be used. However, it is advisable to measure the properties of the Ethernet PHYs which will be used later for clock synchronization, as different PHYs have different jitter properties.

To minimise the number of influencing factors, such as added jitter by switches or high traffic load, first measurements have to be done on a direct-link basis. With direct-link measurements it is possible to focus on the influence of the clock recovery circuitry and the clock transitions of the PHYs, which are the main sources for delay jitter. Asymmetric line delays also deteriorate the achievable precision of clock synchronization and can be observed by the direct-link measurements as well. As Gigabit Ethernet over copper uses all available four twisted-pair lines for both directions of data transmission, in contrast to Fast Ethernet which uses two dedicated pairs for each direction, a reduction of the asymmetry is expected for Gigabit Ethernet. The measurements are needed to verify or falsify this expectation.

The small depicted oscilloscope in the figure symbolises the measurement equipment. In contrast to Ethernet-based clock synchronization, the measurement equipment can sample the transmission time and reception time based on an external reference clock. Therefore, jitter of the one-way delay, as well as an asymmetry of the delay in one direction compared to the delay in the other direction can be measured. To get meaningful results, care has to be taken to use absolutely identical measurement cables to not distort the measurement by unequal signal propagation delays from the nodes to the measurement equipment.

The just mentioned measurements are required to estimate the theoretically achievable precision of a clock synchronization system using the measured components. However, measurements of an operational test network performing clock synchronization over Gigabit Ethernet have to be made to get real numbers of the standard deviation of the clocks of different nodes. Additionally, the nodes can be used to evaluate clock synchronization for systems with long distances between the individual nodes, e. g. measurement stations at the particle accelerator at CERN. Therefore, it is important to provide network nodes that are capable of participating in clock synchronization over Gigabit Ethernet copper and optical fibre networks, which leads to the task requirement of this diploma thesis.

1.2 Task

The task of this diploma thesis is to develop an evaluation system that enables clock synchronization related tests and measurements with Gigabit Ethernet over copper and optical links. The work of this thesis is to be understood as part of the ε-WiFi project[1] by providing the necessary hardware to perform the tests and investigations.

To be able to perform tests and measurements with the new media an appropriate test platform has to be created. In order to keep the development effort for such a test platform as low as possible, commercially available evaluation boards shall provide the base framework. An FPGA based evaluation platform shall be chosen to allow tests with the following transmission standards [5]:

- 10BASE-T (two pairs of twisted-pair copper, data rate is 10 MBit/s)
- 100BASE-TX (two pairs of twisted-pair copper, data rate is 100 MBit/s)
- 1000BASE-T (four pairs of twisted-pair copper, data rate is 1000 MBit/s)
- 1000BASE-LX (point-to-point link over one single-mode optical fibre, data rate is 1000 MBit/s)

Evaluation boards providing connectors for these standards are only available with a PCI Express connection for host PC communication. To be able to use the existing hardware design for the clock synchronization enabled Ethernet NIC, which is the outcome of previous projects, the modules responsible for host communication have to be re-developed. Figure 1.5 sketches out the basic function blocks and supported communication standards of the existing NIC. It also shows the desired new NIC system with the unchanged legacy hardware core, the additional interfaces and the required change from a PCI to a PCI Express based system.

[1]The work presented in this thesis is partly funded by the FIT-IT Project ε-WiFi -Embedded Position Determination and Security in Wireless Fidelity Networks, grant number 813310

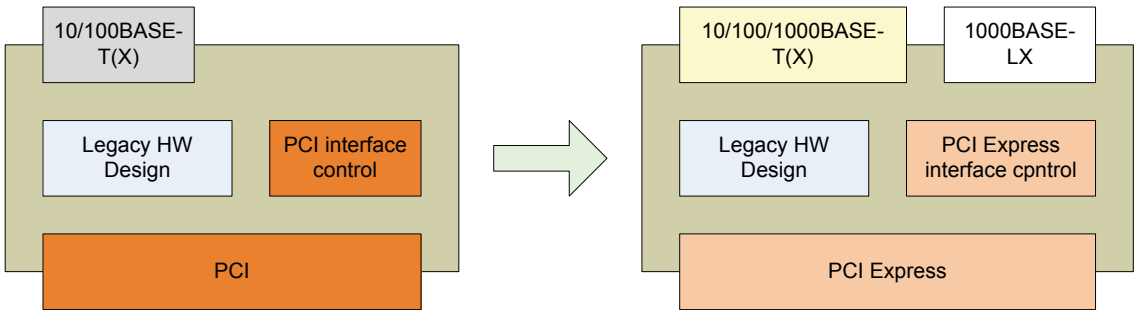


Figure 1.5: Evolution of hardware

The legacy hardware core is maintained by a third party and must not be altered, even though the source code is available. The interface on the legacy hardware side which has to be connected to the PCI Express system is an Advanced Microcontroller Bus Architecture Advanced High-Performance Bus (AMBA™AHB) [6] system. The bus system consists of two AHB slave interfaces and one master interface. The interfaces do not have to be connected to each other, only communication with the device driver in the host PC is required. For this communication transfers with a data width of 32-bit—a Double Word (DW)—have to be supported. The existing device driver requires the hardware to support the reception and handling of single DW read and write requests, as well as the independent generation of burst read and write requests for Direct Memory Access (DMA) with a configurable length in multiples of a DWs. Furthermore, the interface has to be able to generate an interrupt signal for hardware-to-software communication triggering.

A simulation environment shall be provided to be able to simulate the data-flow and operation of the whole NIC architecture. The simulation environment is intended to be a means of basic verification in the development phase. For the intended usage of the evaluation board in an IEEE 1588 clock synchronization test network, it is not required to optimize the design for high performance and high throughput.

1.3 Related Work

The given task addresses a couple of known problems which are treated in scientific literature and papers. Addressed topics are e. g. packet-based clock synchronization in Gigabit Ethernet over copper and optical fibre networks, PCI Express to AHB interfacing, on-chip packet switching and buffering techniques for packet-switching. This section gives an overview of approaches found in literature regarding the mentioned topics and shows where experiences and results from different application fields can be used.

Clock Synchronization

Concerning the main system topic, the high-precision clock synchronization for packet switched systems, a large number of publications is available. On the hardware side for

example in [7] an Ethernet NIC together with a dedicated switch for clock synchronization is presented. A novel concept for enhancements to the IEEE 1588 version 2 standard [3] master/slave based clock synchronization for fault tolerance is extensively discussed in [8]. The main influences on jitter in timestamping and resulting theoretical limits for high-precision clock synchronization accuracy over packet-based networks are discussed in [9]. The ε-WiFi project embodies the knowledge and considerations of the mentioned work. Additionally, to be able to calculate the theoretical limits presented in [9] the delay jitter of the physical layer has to be known. In [10] delay measurements and delay jitter measurements for 10BASE-T and 100BASE-TX PHY devices and cables are presented. Precise delay jitter measurements for 1000BASE-T and 1000BASE-LX physical layers currently are not available and have to be made to be able to adjust the clock synchronization algorithm, as described in [9], to get as close as possible to the theoretical accuracy limit.

PCI Express

The diploma thesis of Nassar [11] deals with the generally connection of some registers to a host PC using the PCI Express connection technology. Though this diploma thesis deals as well with the connection of hardware using the PCI Express interface, the work of Nassar has a completely different objective. It describes the PCI Express architecture in more detail with an emphasis on helping users with their first experiences with PCI Express. An example design is also included, demonstrating read and write operations of single DWs. In contrast, this diploma thesis focuses on the design and the theoretical principles of a more complex communication model in the context of an existing hardware environment. While in the work of Nassar the registers can be directly connected to the PCI Express protocol core, in this diploma thesis the PCI Express packets have to be switched and translated to AHB nodes. Further, this diploma thesis has a focus on the practical design and implementation work with all the occurred traps and pitfalls.

Packet-Switching

The problem of packet-switching is inherent to all direct-link, packet based communication systems with more than two participants. Therefore, it is a very well understood topic with a vast number of available optimisation proposals for different cases of application. For example, [12] proposes a general packet switching methodology for intra-chip architectures. Although this and similar approaches have proved to be useful for systems with a large number of inputs and outputs at very high data-rates, it is not reasonable for this diploma thesis, as the general problem here appears in a very simplified environment with only one input and two output ports. Still, the basic requirements stay the same. The incoming packets arrive on one port, they carry some kind of address information and they have to be forwarded to an outgoing port according to the address information. Therefore, similar techniques as in the well known field of Ethernet switching can be used.

Hein [13] presents fundamental considerations concerning the switching fabric and the switching technique. Two fundamentally different approaches for the switching fabric are discussed, Space Division Switching (SDS) and Time-Division Switching (TDS). SDS uses geometrical structures to interconnect all input ports to all output ports. Those geometrical structures need space, hence the name. A crossbar switch, which connects m inputs to n outputs with a $m \times n$ matrix of crossing points is a well known example for space division switching. In contrast to SDS, with TDS not all inputs have a direct connection to an output. The switching rather is realised by using a shared resource. Circuit switched networks where a fixed slot time corresponds to a specific route from a specified input port to a specified output port employ TDS as well as packet-based switches with shared memory. With these switches the switching algorithm has to take care that each input can write to the shared memory when a packet arrives on a port, and to prevent that two inputs get simultaneous access to the same memory, as the data would be destroyed. Depending on the destination, the respective output port gets read access to the shared memory and thus can forward the packet to its intended destination.

For Ethernet switches Hein [13] also distinguishes between cut-through and store-and-forward switching. Cut-through switching takes advantage of the fact that the destination address of Ethernet packets is at the very beginning of a frame. Therefore, a switching decision can already be made as soon as the destination address has been read and thus the introduced delay of the packet transmission is kept low by the switch. Furthermore, less resources are needed as only a small part of a frame has to be stored at a time. Store-and-forward switching stores a complete frame upon reception, and only after the frame is verified to be correct by calculating the frame check sequence of the received frame and comparing it with the Frame Check Sequence (FCS) field of the frame it is forwarded to the intended destination. This switching technique requires more resources in the switch, as each arriving frame has to be stored and verified. It is especially useful for low performance, low quality networks with a high amount of broken frames. By sorting out the broken frames and not forwarding them, unnecessary network load is avoided.

The specific characteristics of the mentioned techniques of Ethernet packet switching are compared to the needs for the application of this diploma thesis, considering the comparably low throughput and the low number of ports that have to be interconnected.

Buffer Strategies

Switching almost always leads to the need of buffers. Buffers might only be omitted if a specific application scenario enables the switching fabric to always perform non-blocking packet forwarding, and if not more sources address the same destination port simultaneously than the destination port can handle at once. As these very specific requirements are seldom met, various buffering methods exist and are well understood.

Hluchyj [14] describes and compares four different buffering techniques, namely input queuing, input smoothing, output queuing and completely shared buffering. With input

queuing, each input of the switch has a separate buffer. If the addressed output port of a specific packet temporarily is not accessible, the incoming packet is stored in the input buffer. Input smoothing takes advantage of the statistical distribution of packet destinations by buffering all input during a certain time in all input buffers. All received packets are then released simultaneously from all input ports into the switching fabric (which is enlarged, compared to input buffering), thus smoothing the needed paths through the switching fabric over all available paths. With output queuing all packets are routed immediately to the intended output port and are queued there if more than one packet is routed to the same port at the same time. Completely shared buffering basically uses output buffering as well, but all outputs share one bigger buffer. Additionally, the switching fabric has to be enlarged to allow packets to be routed to the shared buffer if the intended output temporarily is not accessible.

Theoretical considerations lead to the conclusion that pure input queuing by using First In First Out (FIFO) structures reduces the maximum data throughput. As a packet is queued when its destination cannot be reached, a consecutive arriving packet might also be queued behind the first packet, although the destination of the second packet would be reachable. The paper also concludes that the use of completely shared buffering is most efficient in terms of required buffers to achieve a given error rate of dropped packets. This efficiency in terms of buffers is traded off with a higher complexity of the switching fabric, which has to include additional output ports to the buffers.

1.4 Document Structure

This document is intended to provide a thorough presentation of the work that was done to develop a system which fulfils the requirements of the given task. This diploma thesis is embedded in a long term project of packet-based clock synchronization. Although a clear focus is set on the description of the actual work, a brief overview of the Syn1588 clock synchronization project is provided as well.

The basic structure of the main part of this document, chapter 3 to chapter 5, can be compared to a top-down approach, while chapter 2 builds the knowledge base for the work. Beginning with the requirements arising from the surrounding and previously existing hardware units, the challenges are identified and theoretical approaches are discussed and evaluated for the use in the system. Then the design decisions for the actual hardware are presented and discussed in detail. The top-down nature of the description can be found in the gradually increasing level of detail, according to the more general presentation of the required functionality in the beginning, down to the very detailed description of the individual realisation of the functional requirements in the end. In the following, a more detailed description of what can be expected in each chapter is given.

Chapter 2 gives an overview of the already existing technologies which were used and needed for the design of the required hardware. From the task description of this diploma

thesis it becomes clear that protocols and their respective differences are of great importance for the actual work. Therefore, this chapter first gives a brief overview of the used protocols, but also describes the specific details which are important to know for the development of the bus-switching unit, the unit which combines the different bus interfaces. The section about the protocol description might be skipped by experts on the field of the mentioned protocols. Furthermore, a brief description of the Precision Time Protocol (PTP) is given, as well as a short presentation of an application example of the developed bus-switching unit together with the clock synchronization enabled NIC design.

In chapter 3 a functional overview of the complete NIC is given. Based on the functional requirements the main challenges are identified and listed. For all identified challenges the theoretical background is discussed and possible approaches are presented. Well known techniques for generalised problems are considered, and it is investigated whether they can be applied to the present problem or not. Approaches and experiences of similar challenges in different system and application environments are also considered, and conclusions are drawn whether those approaches are suitable for the design of this diploma thesis or not.

Chapter 4 presents all of the existing hardware that surrounds the bus-switching unit. First, the hardware platform itself, the chosen evaluation board is described. Then the required interconnection of the different units with their individual interfaces is presented.

Chapter 5 describes the actually used methods to develop the required functionality of the evaluation board. The transition from the old to the new hardware platform is described, mentioning some difficulties which came up during the transition as well as some hints to prevent unnecessary errors. Then the bus-switching unit is reviewed, describing each module in detail with regards to the functionality, the respective realisation and the reasons for the chosen realisation. This chapter also gives an insight into the detailed information flow during operation of the bus-switching unit. Finally, some pitfalls that occurred during the development are described, reasons are given why they could occur and hints state how to avoid similar errors in the future.

Chapter 6 describes two distinct simulation environments which were developed for testing and verifying the hardware design. The simulation environments are very different in nature, and therefore they are both explained. Reasons are given why two different environments are needed after all, and what their respective strength and weaknesses are, as well as for which test cases which environment was needed and used.

The last chapter finally concludes this document, recapitulating the major challenges and issues of the design process. Results are presented in a sense of lessons learned about a hardware design which is embedded in an existing system environment. Also, perspectives are given on how the resulting hardware evaluation board can be used, as well as which enhancements and improvements can be performed in the future.

Chapter 2

Technology Overview

This chapter will present a knowledge base of the technologies in the hardware system that were used. The bigger picture of clock synchronization will also be drawn to understand the intended application of the developed hardware. The bus-switching unit is the main part which has to be developed. It concerns itself about the interconnection of the units with different bus interfaces, translates the protocols and controls the information flow of the interfaces. To understand its operation and the challenges of the development, the protocols in use must be known and understood. Therefore, this chapter presents an overview of the used protocols, giving details only when necessary to understand the hardware design.

2.1 Protocol Overview

In the task description three main protocols were listed which are involved in the system: Ethernet, Advanced High-Performance Bus (AHB) and PCI Express. All three are of different nature, as they are designed for different purposes. Despite their differences they have to be interconnected and data has to be passed from one protocol to the other. This

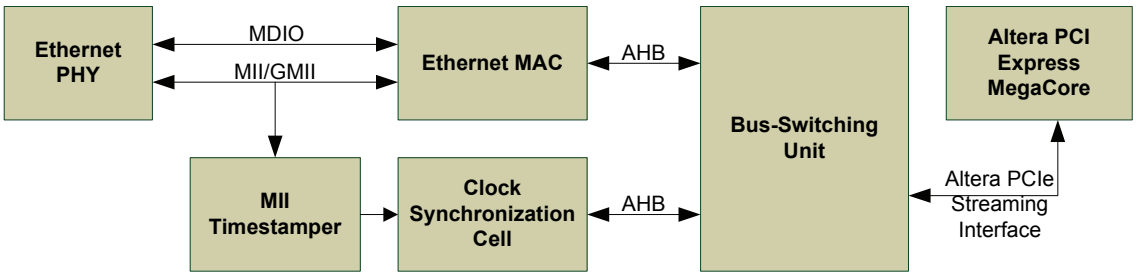


Figure 2.1: Overview of the NIC architecture

raises some questions and challenges which are discussed in the following chapters. Therefore, a description of the protocols in use is given here. Figure 2.1 shows the individual connections of the different units and components of the hardware architecture.

Although the Ethernet protocol is not directly involved in the design of the bus-switching unit, it is still an important part of the overall system and therefore is described here. As the system itself is an Ethernet network interface card, the main task is to process Ethernet traffic. The MAC unit of the existing hardware core processes the Ethernet frames and translates them to AHB transfers and vice versa. Therefore, the bus-switching unit is not directly involved in the handling of Ethernet traffic. Still, the data it must process results from Ethernet frames.

The MII is the data connection between the Ethernet PHY and the MAC. This interface is important for the hardware support for clock synchronization as well as for the simulation environments. The MII management interface is a simple two wire serial interface for the management connection between the Ethernet PHY and the MAC. It consists of the Management Data Clock (MDC) and the Management Data Input/Output (MDIO) signals. It is described because some problems arose with this interface during the hardware transition to the new evaluation board.

The AHB protocol is a high-speed intra-chip bus protocol which is used by the Clock Synchronization Cell (CSC), the unit for the hardware support for clock synchronization, as well as by the MAC. Those two units have to be connected to the PCI Express link, which is depicted on the right side of the figure. The corresponding traffic from and to the PCI Express protocol core interface must be switched and translated. This is the task of the bus-switching unit, and therefore these two protocols are laid out in more depth. Because of the three layered architecture and the complex structure of PCI Express, not the whole PCI Express protocol is explained; only the transaction layer, the highest layer of the PCI Express protocol stack, is reviewed in more detail.

2.1.1 Ethernet

Ethernet, as defined in the IEEE 802.3 standard [5], is a standard for connecting devices in a Local Area Network (LAN) or Metropolitan Area Network (MAN) over a shared medium using the Carrier Sense Multiple Access/Collision Detection (CSMA/CD) access method. Though most office networks use Ethernet in full duplex mode with point-to-point connections where no contention of the shared medium can occur and thus the CSMA/CD mechanism is not used, it still is an important and integral part of the Ethernet protocol. On the one hand many requirements are derived from the necessities for the CSMA/CD access method, e. g. the minimum frame length, and on the other hand classical CSMA/CD implementations are still used in industrial environments.

CSMA/CD access method means that all participants on the shared medium observe the carrier for activity. If there is no activity sensed, each participant is allowed to start the transmission of a packet at any given point in time. If an activity is sensed, the

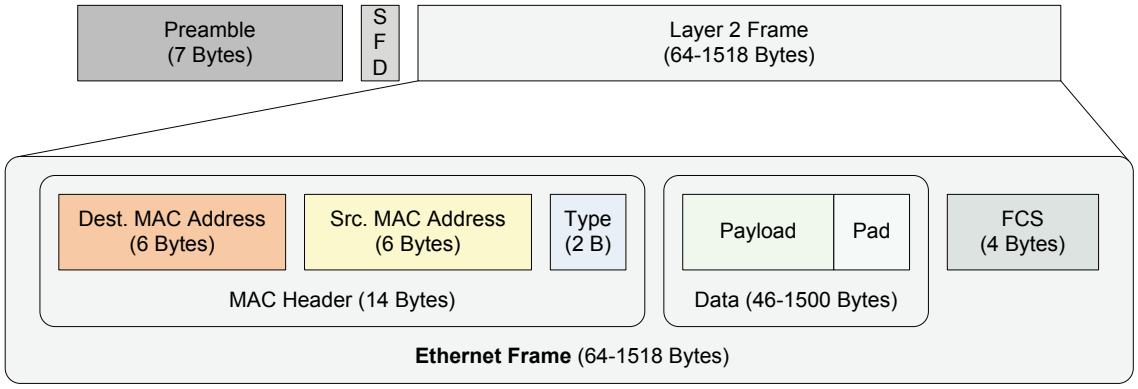


Figure 2.2: Structure of a standard Ethernet packet neglecting extensions

participant has to wait until the shared medium is found to be idle again. After sensing the transmission line to be idle, it has to wait for the time specified by the interframe gap, which is 96 bit times long. Because of the signal propagation delay it might happen that two participants correctly start the transmission of a packet which will collide when they have propagated over the network. In this case, all participants back off and try to retransmit the frame after a random backoff time, with the length of the backoff time being generated randomly, with the maximum duration depending on the number of previously failed transmissions.

2.1.1.1 Ethernet Packets

Ethernet is a packet-based transmission protocol defining two layers according to the ISO Open Systems Interconnection (OSI) reference model [15]. An Ethernet packet on the physical layer consists of a layer two frame plus the preamble and the Start-of-Frame Delimiter (SFD). The structure of a packet and a frame is depicted in figure 2.2. A packet starts with the preamble, which is a sequence of seven times an octet of the hexadecimal value 0x55, which is used for the PHY to synchronize on the received packet. The SFD is an octet of the hexadecimal value 0x5D and indicates the start of a frame.

The SFD is followed by the actual frame which starts with the destination MAC address. The destination address being at the beginning of the frame allows for a quick decision whether the frame is intended for the actual receiver or not, as well as for quick routing decisions in a switch. Moreover, because of the network byte order of Ethernet being big endian and the multicast identification being contained in the first byte of the destination address, a quick decision for multicasts can be performed. The following source address indicates the sender of the frame. The length/type field specifies either a registered Ethernet frame type (values $\geq$ 0x600), or it specifies the length of the payload (values $\leq$ 0x5DC). The payload field holds the data and is followed by an optional padding field to extend the length of the payload field to the minimum amount of 46 bytes if necessary. The four bytes of the FCS are the last bytes belonging to a frame and contain the value

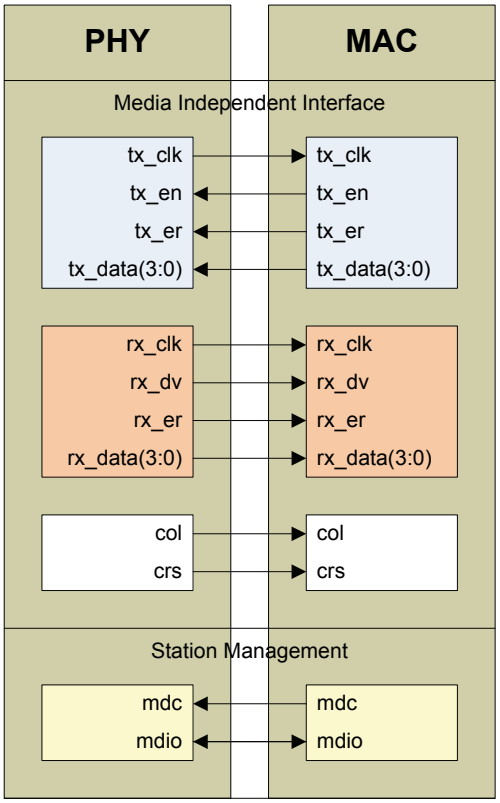


Figure 2.3: MII signals

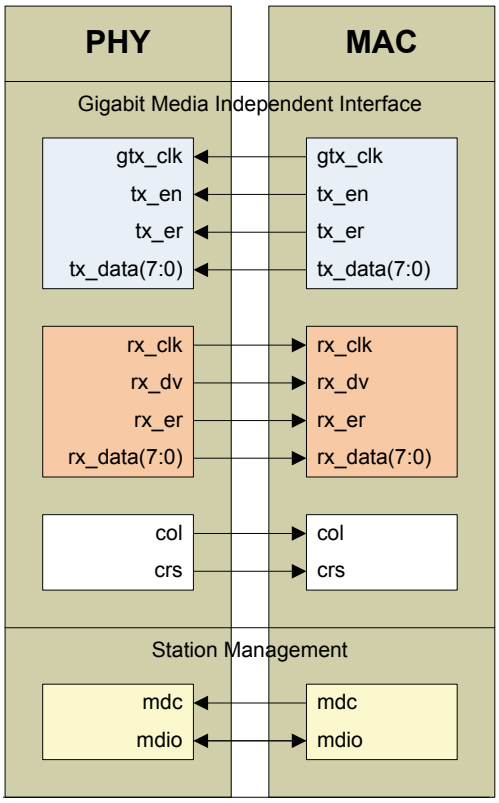


Figure 2.4: GMII signals

of a Cyclic Redundancy Check (CRC), which is computed as a function of all frame fields except for the FCS field itself, thus containing all fields from the destination address to the optional padding field.

2.1.1.2 PHY to MAC

The Ethernet PHY is on one side connected to the physical medium, the cable. On the other side it is connected to the MAC unit. For this connection two different interfaces are used, the Media Independent Interface (MII) and the MII management interface. All Ethernet data is transmitted over the MII link. The MII basically consists of parallel data lines for the actual Ethernet data, as well as of clock signals and some control signals, while the MII management interface consists of a dedicated clock line and a bidirectional data line, which is called the MDIO signal line.

Media Independent Interface

Figure 2.3 shows the various signals of the MII connection between the PHY and the MAC. The data lines (`rx_data` and `tx_data`) always carry four bits of data in parallel. While the coding and clock frequency of the physical link from one Ethernet node to

another depends on the used medium, the protocol of the MII is independent of the used transmission medium, hence the name. Because of the four parallel data lines the clock frequency for MII is the fourth part of the bit rate. For a 10 Mb/s link the MII clock rate is 2.5 MHz, for 100 Mb/s links the clock rate is 25 MHz. Both, the receive (`rx_clk`) as well as the transmit clock (`tx_clk`) are sourced by the PHY. While the receive clock is derived from the received data, the transmit clock is sourced by a local oscillator. The rest of the signals are control signals which are used to indicate the reception and transmission of a frame (`rx_dv` and `tx_en`), for error propagation and detection at the PHY level (`tx_er` and `rx_er`) and to propagate a busy line (`crs`) and the detection of a collision (`col`).

Gigabit Media Independent Interface and Further Derivatives

For Gigabit Ethernet the MII is no longer used, but the definition is enhanced to the Gigabit Media Independent Interface (GMII) protocol which is very similar to MII. Figure 2.4 shows the signals of the GMII connection. In comparison to MII, only the data lines are doubled to eight data lines per direction, and the transmit clock (`gtx_clk`) is not sourced by the PHY but by the MAC. Because of the increase of the parallel data lines the clock frequency is no longer the fourth part of the bit rate, but the eighth part, thus being 125 MHz for Gigabit Ethernet.

Further derivatives of the MII and GMII definitions with reduced pin counts exist, though they are not covered by the IEEE 802.3 standard specification. Reduced Media Independent Interface (RMII) [16] instead of MII, and Reduced Gigabit Media Independent Interface (RGMII) [17] instead of GMII are de-facto standards proposed by special working groups. Both use less sideband signals and use only half of the data pins than the original definition, but operate at a doubled clock rate. For RMII this means only two data lines at a clock frequency of 5 MHz for 10 Mb/s operation and 50 MHz for 100 Mb/s operation. RGMII uses four data lines and keeps the original clock frequency of 125 MHz, but reacts on both, rising and falling edges of the clock signal and thus doubles the actual data rate.

MII Management Interface

The MII management interface is common to all versions of MII and its derivatives. It consists of two wires, the MDC and the Management Data Input/Output (MDIO), as defined in IEEE 802.3 clause 22. It is a serial interface to transport management frames between the MAC and the PHY. The PHY is requested to have a basic management register set which can be written and read using the MII management interface.

The MDC signal is an aperiodic clock source for the management data frames, which has a minimum period of 400 ns, corresponding to a frequency of 2.5 MHz. The duration of the high and low times must be at least 160 ns each and have no upper bound. The MDIO signal is a bidirectional line which requires both sides to have tri-state drivers and needs a pull-up resistor to pull the bus high in idle state. The structure and timing of the

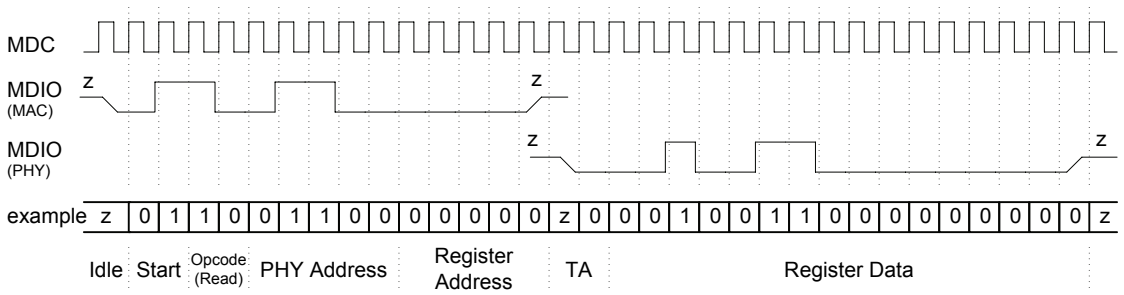


Figure 2.5: Timing and structure of a MDIO frame

management frames is depicted in figure 2.5. Special to a tristated bus is the turnaround phase. When a read operation is performed, first the MAC drives the MDIO line to initiate the read transfer. In the turnaround phase control is handed over to the PHY to transmit its response using the same MDIO line that the MAC used before. Therefore, the MAC shall change to high impedance state during the first bit time of the turnaround phase while the PHY still stays in high impedance state. During this first bit time both drivers are disabled to prevent a short circuit of the drivers. Only during the second bit time the PHY drives the MDIO line with a zero bit, and thus takes over control of the MDIO signal line.

2.1.2 Advanced High-Performance Bus

The Advanced High-Performance Bus (AHB) protocol [6] is part of the Advanced Microcontroller Bus Architecture (AMBA) specification and is designed as a high-speed intra-chip bus for the interconnection of different hardware cores. It is specifically designed for synthesizable, high performance and high clock-rate designs. It is a master-slave architecture, using a non-tristate implementation, supporting pipelined burst transactions and variable data widths from 8 to 1024 bits. Out-of-band control signals together with pipelining enable high performance with very low control overhead. Two different transfer types are supported, non-sequential transfers and burst transfers.

The AHB master slave architecture is designed to support multi-master multi-slave configurations with an address decoder, a dedicated bus arbiter module and the support for split burst transactions to allow optimal resource usage of the shared bus medium. In the course of this diploma thesis only a single-master system is used with no need for an AHB bus arbiter or the support of split transactions. Such a system is also called an AHB lite system [18]. The master and slave bus interfaces are depicted in figure 2.6.

2.1.2.1 Control and Slave Response Signals

The control signals are sourced by the master interface to control the individual AHB transfers. The `trans` signal indicates the type of transfer and encodes either one out of

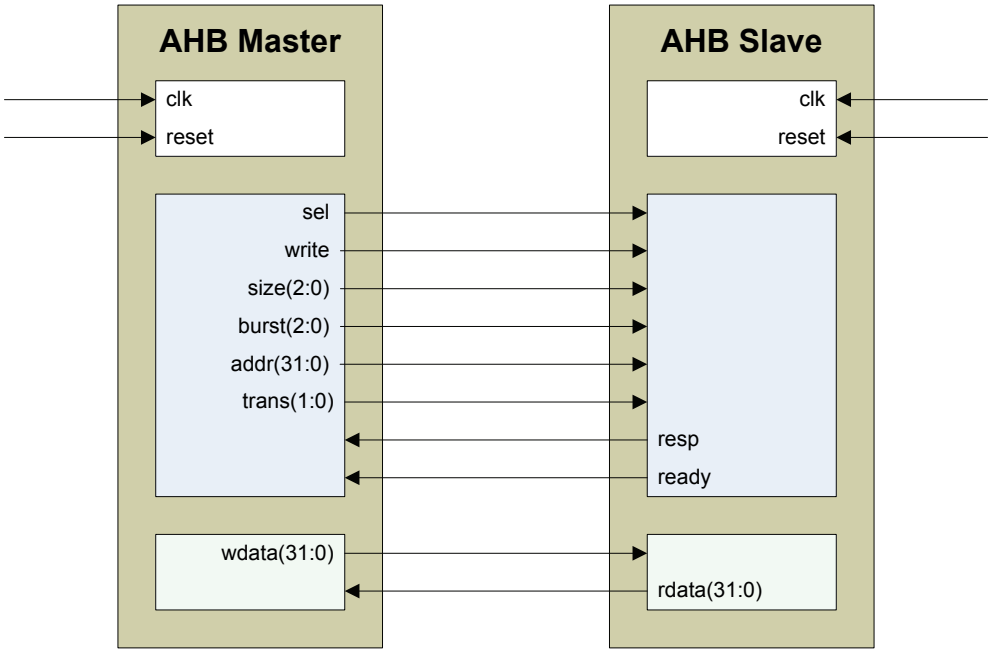


Figure 2.6: AHB lite master and slave interface signals

four possible transfer types, *idle* (00), *busy* (01), *non-sequential* (10), or *sequential* (11). The `write` signal indicates the direction of the transfer. If it is asserted, data is written from the master to the slave. If it is deasserted, data is read from the slave to the master. The `size` signal specifies the used data width with a valid range from 8 to 1024 bits, where the three bits of the `size` signal encode an offset of a power of two. An encoded zero of the `size` signal corresponds to $2^3 = 8$, an encoded one corresponds to $2^4 = 16$, and the highest encoded value seven corresponds to $2^{10} = 1024$. The `prot` signal is an optional signal for protection control and is not used in the design of the diploma thesis. The same holds true for the `lock` signal, which can be used in multi-master systems to indicate indivisible transfers.

The address is a byte address (a change of the least significant bit changes the addressed memory by one byte) and is encoded by the 32 available address bits. Their timing is exactly the same as the timing of the control signals, which is depicted and explained in the following sections. An address decoder, usually a simple combinatorial decoder of the high-order address signals, takes care of selecting the addressed slave by asserting the corresponding `sel` signal. As the minimum memory space that can be mapped to an AHB slave is 1 kB, no burst must cross a 1 kB address boundary in order to prevent transfers to exceed the address range of a slave.

The slave response signals are the `ready` signal and the `resp` signal. The `ready` signal is used to indicate the readiness of the slave. For write transfers this means that the slave indicates by the assertion of the `ready` signal that it is ready to receive data in the next clock cycle. For read transfers the assertion of the `ready` signal indicates valid data on the

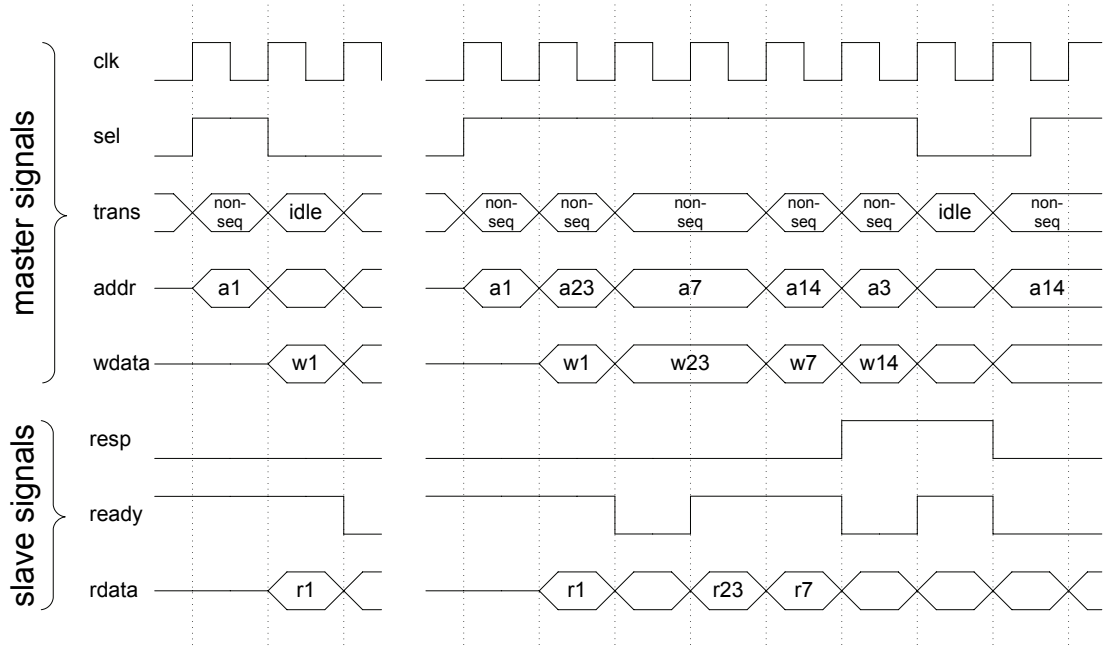
slave's `rdata` lines. When the master issues an *idle* transfer, the slave must respond with the `ready` signal asserted. Therefore, the slave usually can not prevent the master from initiating a transfer, but it can insert wait states until the next data is presented. The response (`resp`) signal for an AHB lite slave can be either one of two possibilities, *okay* (0) or *error* (1). On any kind of error the slave is expected to respond with an error response, which is a two-cycle response. In the first cycle the `resp` signal has to be asserted to indicate an error, while the `ready` signal has to be deasserted. In the subsequent cycle the `resp` signal still stays asserted and the `ready` signal is asserted as well. A two-cycle response is needed because of the pipelined nature of AHB, to allow the master to cancel a potentially new transfer that has already been issued. A transfer with an error response is shown in figure 2.7b.

2.1.2.2 Non-Sequential Transfer

The basic transfer type of AHB is the non-sequential transfer. Its duration is two clock cycles, and it consists of two phases, one phase per clock cycle. The first phase is the address phase, where all control signals are asserted according to the intended transfer. A non-sequential transfer consists of exactly one data phase with the duration of one clock cycle. During the data phase the data is presented on the respective data bus, either on `rdata` for read transfers, or on `wdata` for write transfers. By using two distinct data buses a two directional bus can be implemented without the need for tristate drivers. The maximum transmittable data during a non-sequential transfer is limited by the data width. Figure 2.7a shows the timing of a typical non-sequential transfer.

For non-sequential transfers one clock cycle is needed for the address phase and one clock cycle for the data phase. This results in an overhead of 50%, which is not acceptable for a high-speed, high-performance bus architecture. Therefore, AHB provides the possibility for pipelining transfers. The first transfer needs a separate address phase, indicating the address and the transfer type for the first data transfer. During the next clock cycle the data is presented, but the address and control lines are not required anymore for the first transfer. They can already be used to initiate the next transfer, with the corresponding data being presented in the subsequent clock cycle. In other words, the data phase of a transfer can simultaneously be used as address phase for the next transfer. Figure 2.7b shows an example for pipelined, non-sequential transfer. The address numbers are chosen arbitrarily to indicate the arbitrary, independent nature of addressing with non-sequential transfers. By pipelining the only remaining protocol overhead is one clock cycle at the beginning, reducing to zero for all subsequent transfers, which makes the protocol perfectly suitable for high-performance applications.

To be able to utilise the full benefit of pipelining both endpoints of the bus have to be fast enough to handle the data within one clock cycle. Not all endpoints are at all times capable of such a high-speed data processing. Therefore, both sides of the AHB protocol are allowed to insert wait states. If the master is not ready to immediately initiate a new



(a) Non-sequential transfer **(b)** Fully pipelined non-sequential transfer

Figure 2.7: Non-sequential AHB transfers

transfer it can simply issue *idle* transfers until it is ready for the next transfer. If the slave is not ready to immediately handle new data it can deassert the `ready` signal and thereby force the master to remain in the current state until the `ready` signal is asserted again, as also shown in 2.7b.

2.1.2.3 Burst Transfer

A burst transfer is very similar to a series of fully pipelined non-sequential transfers, it even starts with a non-sequential transfer. As the name indicates, the important difference to non-sequential transfers is the sequential nature of addresses in a burst transfer. In contrast to non-sequential transfers, where each transfer can have an arbitrary address, independent of the previous transfer, burst transfers have to follow a strict addressing scheme.

The `burst` signal specifies the nature of the burst. It can be either incrementing with an unspecified length, incrementing with a specified length or a wrapping burst of specified length. Specified length transfers can be either one of 4, 8, or 16 beat length, while a beat corresponds to a data cycle. Thus, the number of beats has to be multiplied by the data width to get the actual amount of data which is transferred during a burst transfer.

All addresses within a burst transfer must be aligned to the address boundary which is specified by the `size` signal. Therefore, all 32 bit transfers must be aligned to 32 bit

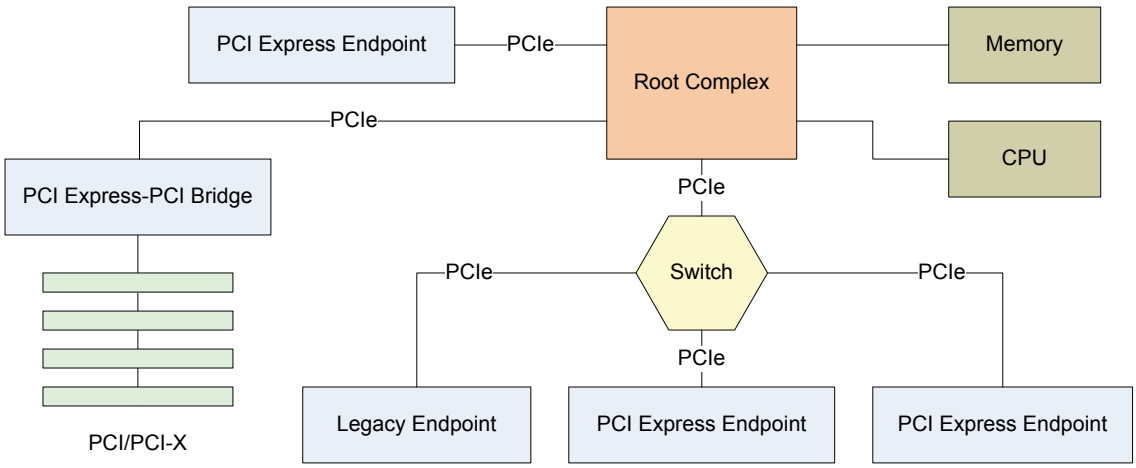


Figure 2.8: Example topology of a PCI Express architecture

boundaries (the two least significant bits of the address have to be both zero), all 64 bit transfers must be aligned to 64 bit boundaries (requiring the three least significant bits to be zero). For all incrementing bursts, the address in a subsequent clock cycle must be the address of the previous clock cycle incremented by the size transferred at each beat, specified by the `size` signal. Incrementing bursts with an unspecified length have no further restriction than not to cross a 1 kB address boundary. Wrapping transfers are basically of incremental nature as well, but if they are not started at an address aligned to the total number of bytes transmitted in the burst, they wrap at the boundary of this total number of bytes.

Burst transfers also allow both, slaves and masters to insert wait states. The master can insert *busy* transfers, while the slave again uses the `ready` signal to stall the master. The advantage of burst transfers is the fixed addressing scheme. Many applications and devices need more time to handle a random access transfer than a sequential transfer. Examples are Random Access Memory (RAM) modules which can handle sequential transfers on the same memory page much faster than if the page has to be switched, but also applications which can pack sequentially addressed data into a single container structure.

2.1.3 PCI Express

The PCI Express protocol specification [19] describes PCI Express as a third generation I/O interconnect. It is an advancement of the PCI protocol, intended to stay compatible with PCI concepts such as the load-store architecture and the enumeration process, allowing legacy PCI drivers to work unmodified with a PCI Express design.

PCI Express is a packet-oriented, switched point-to-point protocol, using a highly scalable, serial differential electrical interface, providing means for e. g. power management, quality of service, data integrity and error handling. Figure 2.8 shows a typical architecture of

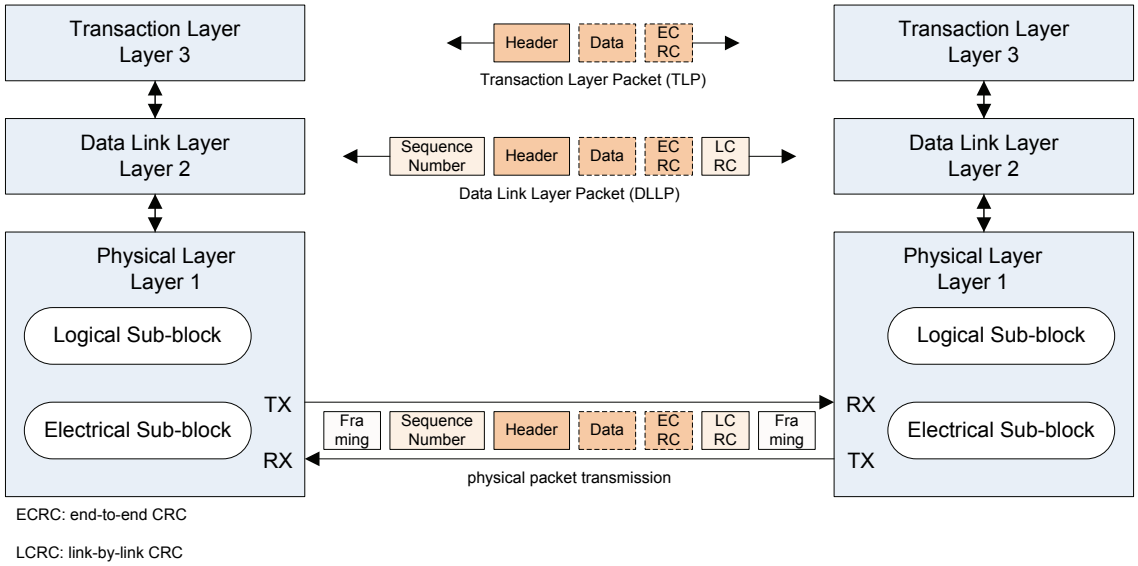


Figure 2.9: Three layered architecture of PCI Express

a PCI Express system. It consists of several point-to-point links of the root complex to multiple endpoints, which are I/O devices. The root complex connects the CPU/memory subsystem to the I/O components. The endpoints can be legacy endpoints, which use the PCI Express protocol, but are required to work with a legacy PCI device driver which uses functions supported only for PCI compatibility. Standard PCI Express endpoints use only the semantics specified for pure PCI Express endpoints, and PCI Express-PCI Bridges connect PCI hierarchies to the PCI Express fabric.

PCI Express is a three layered architecture, specifying a physical layer for the interconnection of the components, a data link layer for link management and data integrity, and a transaction layer to assemble read and write Transaction Layer Packets (TLPs) and for credit-based flow control. Figure 2.9 gives an idea of the data-flow and the respective packet formats through the layers.

2.1.3.1 Transaction Layer

For this diploma thesis only the transaction layer is relevant. The transaction layer of an endpoint communicates with the transaction layer on the other side of the link using transaction layer packets. These packets are of different types and can be addressed to one of four address spaces: memory mapped, I/O mapped, configuration and message. Memory mapped read and write transactions are the standard transaction type for PCI Express endpoints. I/O mapped transactions are only supported for legacy device drivers which require their use. Configuration transactions are used to access the configuration registers of PCI Express devices and are primarily used during the initialisation phase. Message transactions can be viewed as virtual wires which take over all protocol messaging

tasks which were previously signalled by out-of-band signal lines, e. g. interrupts, flow control management and power management.

Generally, except for messages, there are two types of transactions, read and write. TLPs which initiate a transaction sequence are called requests. If they operate in the memory mapped address space, they are called memory read and memory write requests, respectively. Memory write requests are posted requests, which means that they don't require an answer from the receiver. Memory read requests are non-posted requests. Read transactions are split transactions, requiring the receiver to send a completion TLP back to the requester which initiated the transfer. To do so, all requests are tagged to be uniquely identifiable, so that the interconnect fabric can route them appropriately to the initiator of the transaction.

2.1.3.2 Transaction Layer Packets

Depending on the address range of an endpoint, Transaction Layer Packets (TLPs) basically consist of a four Double Word (DW) long header for 64-bit addressing, and of a three DW long header for 32-bit addressing plus the data payload and an optional end-to-end CRC field. Figure 2.10 and 2.11 depict the header formats for requests and completions. All fields marked with an `R` are reserved for future use and must not be used.

The first DW of the header is common to all TLPs. The format field (`Fmt`) specifies the length of the header (three DW or four DW) and whether data is included in the packet or not. The `Type` field specifies the type of the TLP. The basic types are read and write, and the type field also defines the address space. Therefore, memory mapped requests have a different type than I/O requests and messages. The fourth type of TLPs are completion as response to previous read requests. Further sub-types are available for locked transfers and special message routing information.

Bits four to six of byte one specify the traffic class (`TC`) which is used for Quality of Service (QoS). All three bits being zero corresponds to best effort traffic. If the TLP Digest (`TD`) bit is set, the TLP includes an 32-bit end-to-end CRC (ECRC) field after the data payload. If the error forwarding by data poisoning (`EP`) field is set, it is indicated that some component along the path of the TLP has found the data to be poisoned. The `Attr` field contains two bits for optimised traffic handling, the no snoop and relaxed ordering attribute. Finally, the `Length` field specifies the length of the payload in DWs.

Memory Request TLP

Memory request TLPs can be initiated by all participants in the PCI Express complex. They are used to perform read and write transactions on memory mapped devices. The header of memory requests has five additional fields to the common header fields. The `Requester ID` holds the unique number of the requesting PCI Express component. This Identification (ID) is formed by the bus number, device number, and function number.

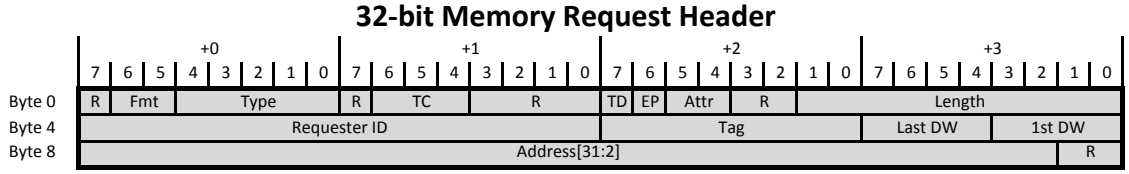


Figure 2.10: Header format of PCI Express memory requests

The bus and device number of an endpoint might change during operation and have to be tracked. Together with the `tag` field, which identifies a read request within an endpoint, a unique identifier in the PCI Express system is associated with each request.

Usually, it is sufficient to use only the five least significant bits of the tag field, which corresponds to 32 possible uncompleted requests. The requester is responsible for keeping track of the uncompleted requests and must not have more than 32 outstanding requests at a time. However, the nodes can be configured to use all eight bits of the `tag` field if required, allowing for 256 concurrently outstanding requests. As write requests are posted requests and do not need a completion, the `tag` field for write requests is unspecified and may have an arbitrary value.

The `Last DW` and `1st DW` fields specify the valid data bits for payload that is not quadword (64-bit) aligned. The four bits of the `1st DW` field correspond to the first four bytes of the data payload, indicating whether a byte contains valid data or not. The four bits of the `Last DW` specify the usage of the last four bytes of the data payload. The `Address` field finally stores the full byte address of the transaction and can be 32 bits or 64 bits long, depending on the address range of the PCI Express node. All transactions performed on addresses below four Gigabyte must use a 32-bit address.

Completion TLP

A completion TLP is always the response to a previous non-posted request. It is the second and final part of a split transaction and either returns some data, or in the case of I/O transactions informs the initiator about the completion of the transaction. Figure 2.11 depicts the header format of a PCI Express completion TLP. A completion header has several specific fields in addition to the common header fields. Most of them are used for the correct correlation of a completion to the corresponding read request.

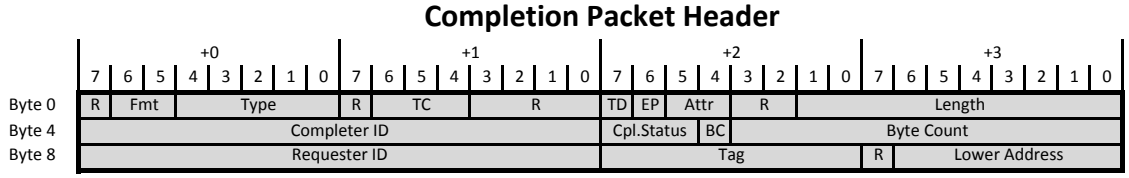


Figure 2.11: Header format of PCI Express completions

The traffic class and attribute field of the common header part have to match the values of the corresponding request. The `Completer ID` specifies the ID of the completing component. The `Cpl Status` field holds information about the completion status of the completer. Currently supported completion states are successful completion, unsupported request (e. g. wrong or unsupported message for a message transaction), configuration request retry status and completer abort, which is used if an endpoint does not support a specific request, although it would be allowed by PCI Express (e. g. read of an unreadable address).

The `BC` field is a field for legacy PCI-X components and indicates that the byte count has been modified. It is possible to use multiple completions of smaller size to complete a single read request. For this case, the `Byte Count` field exists and indicates how many bytes still have to be transferred to complete a read request. The `Requester ID` and `Tag` field must have the same values as stored in the corresponding read request. These fields are used for routing by the PCI Express complex, to return the completion to the correct requester. Finally, the `Lower Address` holds the seven least significant bits of the address of the first enabled byte of the read request.

2.1.4 Altera PCI Express Interface

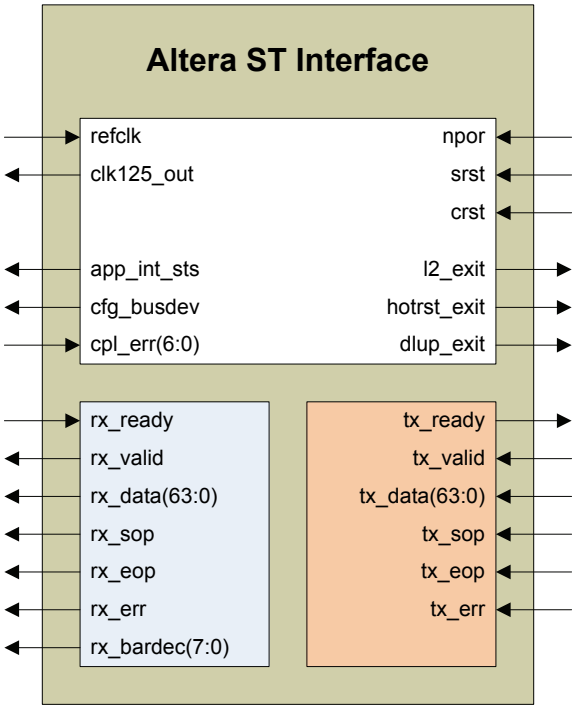


Figure 2.12: Signals of the Altera MegaCore PCI Express Avalon Streaming interface

The Altera® MegaCore PCI Express Interface is the interface used between Altera's PCI Express hardware core and the application hardware as described in [20]. A fully featured

PCI Express endpoint embodying the functions of all three layers is quite a complex piece of hardware. Altera offers a ready made solution as part of their MegaCore library, offering an interface to the transaction layer of the PCI Express endpoint. The interface requires the user application to handle data in the form of PCI Express TLPs.

The interface basically consists of an adapted Avalon® bus interface for the transmission of TLP data plus additional control and information signals. Figure 2.12 shows the signals of the Avalon bus and important control signals. The Avalon bus is adapted for a direct point to point connection transmitting PCI Express TLPs only. Therefore, no address signals are included and the read and write indicating signals are omitted as well, because the corresponding information is contained in the TLP.

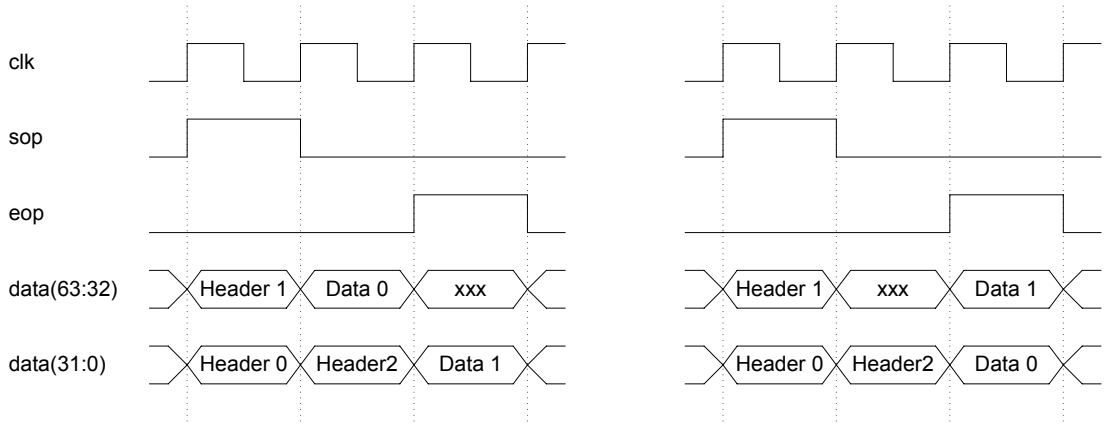
The `ref_clock` signal must be sourced by the 100 MHz reference clock of the PCI Express connector. The reference clock line is not allowed to source any other unit than the Altera interface. The transceiver module of the Field Programmable Gate Array (FPGA) device transforms the 100 MHz input clock to a 125 MHz output clock, which can be used as clock source for the application. The assertion of the `app_int_sts` signal triggers the Altera core to generate a PCI Express message TLP and send it upstream, which corresponds to a legacy PCI interrupt. All signals on the right side of the white block in the figure are reset signals and are used for different levels of reset.

The transmission of a TLP always starts with the assertion of the Start Of Packet (SOP) signal line (`rx_sop` and `tx_sop`) during the first data cycle. Simultaneously with the assertion of the SOP indicator the data valid signals (`rx_valid` and `tx_valid`) are asserted together, indicating that the data lines contain valid information. The `ready` signal is used by the receiving side to insert wait states, but in contrast to the AHB the transmitting side only needs to react on a change of the ready signal within three clock cycles. The valid signals must stay asserted throughout the whole transmission, except for a three clock cycles delayed reaction on a change of the ready signal. They must not be used to insert wait states by the transmission side. Actually, there are no means at all for the transmission side to stall a transmission by inserting wait states. The transmission is finished when the End Of Packet (EOP) signal is asserted during the clock cycle when the last data is transmitted. After this last data cycle all signals are deasserted again and the transmission is finished.

The interface is responsible for the transmission of TLPs between the PCI Express core and the application hardware. The TLP information simply is transferred using the 64-bit wide data lines of the Avalon interface. But the mapping of TLP payload to the Avalon bus depends on the address alignment of the TLP. Figure 2.13 shows the difference between non-quadword and quadword aligned addresses. For non-quadword aligned addresses the first DW of payload data directly follows the header data, while the data of quadword aligned addresses only starts with the next full quadword and leaves the upper DW after the header data unused.

The rest of the signals concern configuration specific information, e. g. the bus device number (`cfg_busdev`) or the addressed memory bar (`rx_bardec`), depending on the real

	Bit 63	Bit 0
Header 0	PCIe Header Byte 0, Byte 1, Byte 2, Byte 3	
Header 1	PCIe Header Byte 4, Byte 5, Byte 6, Byte 7	
Header 2	PCIe Header Byte 8, Byte 9, Byte 10, Byte 11	
Data 0	PCIe Data Byte 3, Byte 2, Byte 1, Byte 0	
Data 1	PCIe Data Byte 7, Byte 6, Byte 5, Byte 4	



(a) Non-quadword aligned TLP **(b)** Quadword aligned TLP

Figure 2.13: Mapping of 32-bit addressed PCI Express TLP to Avalon ST bus

address assigned to the PCI Express node during initialisation. Further signals are used for different levels of reset, for interrupt generation, and error reporting.

2.1.5 Precision Time Protocol

The Precision Time Protocol (PTP) is a standardised protocol for clock synchronisation in packet-oriented, distributed network architectures such as Ethernet. It is defined in the IEEE 1588 standard [3], a standard for precise synchronization of clocks in measurement and control systems.

A basic system for clock synchronization consists of a sole grandmaster clock and various slave clocks which synchronize to the grandmaster. For synchronization, so-called Sync and Delay-Request messages are used. Sync messages are sent by the master using multicast to inform the slaves about the current time of the master. The knowledge about the absolute time of the master at the transmission time is not enough for precise clock synchronization. The delay of the packet from the master to the slave has also be taken into account. Therefore, the slave can send a Delay-Request message to the master. Figure 2.14 shows these messages in the synchronization principle. Together with the Sync message the round-trip delay as well as the clock offset can be calculated. A detailed description of this calculation can be found in the standard [3], clause 11.

Sync and Delay_Request messages are part of a group called event messages, which have to be timestamped. One timestamp is taken when an event message is sent, and an-

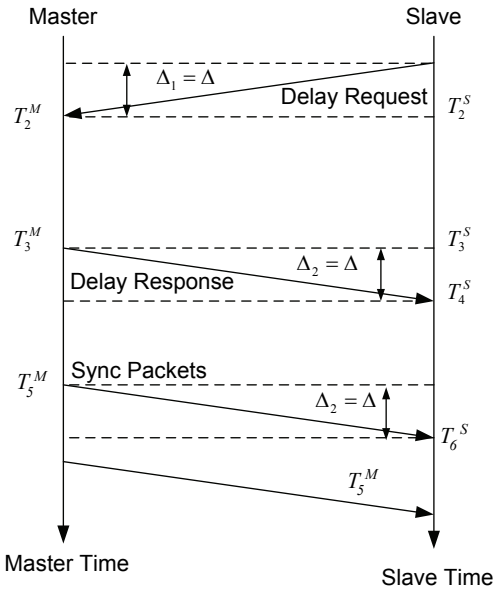


Figure 2.14: Synchronization messages and round-trip delay measurement

other timestamp is taken when the event message is received. With a Sync message the receive-timestamp is drawn by the slave and therefore is known, while the send-timestamp either is directly included in the event message (one-step clock), or it is sent in a dedicated Follow_Up message (two-step clock), informing the receiver of the send-timestamp. Therefore, the receiver of a Sync message, the slave, always has knowledge about the send-timestamp and the receive-timestamp. Therefore, the sum of the delay of a packet in both directions as well as the clock offset of the slave clock compared to the master clock can be calculated. Under the assumption of constant delay the sum of the delay of the two packets corresponds to the round trip delay. The slave clock then uses this information in a control loop to correct its clock.

Uncertainties remain due to jitter and asymmetric delays of packets. Because of liberties in serialisation and deserialisation of the bit stream in the PHY, but also because of asymmetric cable properties the delay might vary depending on the direction of transmission. The IEEE 1588 protocol provides no means to measure the transmission asymmetry. However, the protocol allows for asymmetric delay cancellation if it is known in advance. Further uncertainties are added because of any jitter introduced during transmission. To keep the jitter as low as possible it is important to draw the timestamps as close as possible to the actual transmission of a packet on the physical medium to reduce jitter influence of higher layer protocol processing.

2.2 Typical Application

The design of a component that connects a PCI Express core to an AHB system can be used in a variety of applications. Typically, such applications will be peripheral components which add some functionality to a standard PC system. As AHB is a very common, general purpose high-speed high-performance intra-chip bus system it can be used for all kinds of applications which require data transmission. The PCI Express protocol also offers a highly scalable, high performance standardised connection. PCI Express already is the main standard for connecting peripheral components to a PC system. Therefore, an AHB to PCI Express bus-switching unit can be used for low cost standard components as well as for high performance devices. Examples for such devices are extension cards for control and measurement, specific hardware extensions for calculations and parallel computing and extension cards for bus interfacing, like the Syn1588 NIC.

The Syn1588 NIC

The bus-switching unit that was developed in the course of this diploma thesis is specifically designed, but not limited to the use with the Syn1588 NIC hardware [21] design. Therefore, the NIC may also be considered as an application example for the bus-switching unit.

The Syn1588 NIC embodies the knowledge gained through the long term project SynUTC. The project started years before the PTP standard was approved by IEEE in the year of 2002 [22] with the number 1588. Still, the current hardware version is compliant to the current IEEE 1588 standard, version 2 [3]. Figure 2.1 already showed the basic components of the NIC hardware. In parallel to the standard Ethernet path over the Ethernet MAC unit the hardware support for clock synchronization is depicted.

The dedicated hardware for clock synchronization uses an interval based algorithm together with adder based clocks for smooth rate adjustment [23]. This approach enables seamless clock synchronization by continuous amortisation with no points of discontinuity of the controlled clock. The adjustment of the step size of the adder in an adder based clock varies the rate of the clock, so that it is possible to increase or decrease the clock rate according to the measured clock offset from the master clock.

Clock Synchronization Operation

In order to get accurate timestamps of received and transmitted Ethernet frames, the MII Scanner (MIIS) scans all data on the MII and analyses it. When the start of an Ethernet frame is detected, the MII Scanner asserts a signal and triggers the CSC to take a timestamp. The scanner continues to observe the complete Ethernet frame, checking whether its content is a PTP packet that really needs to be timestamped. If not, the previously taken timestamp is dropped and no further action is taken.

If the packet needs to be timestamped another signal is asserted, informing the CSC to store the current timestamp together with an identification number of the PTP packet. The interrupt line is asserted and the interrupt status register is set to identify the CSC as the source for the interrupt. The device driver then can read out the timestamp and packet ID by accessing the corresponding registers in the CSC.

The dedicated hardware is needed to reduce protocol jitter by higher layers, as the packets are scanned when they enter or leave the PHY. The only remaining jitter is induced by the PHY and by environmental influences on the physical medium itself. In comparison to the ISO OSI reference model, the timestamper has a special role. It scans the interface between the layer one and layer two implementation, but in order to detect PTP event messages it must have knowledge concerning the protocol structure of all higher layers up to the application layer, on which the PTP stack is realised.

Chapter 3

Challenges and Possible Solutions

After the individual protocols, their signals, their basic operation and their specialities were presented in the previous chapter, in this chapter first the principle of operation of the Ethernet NIC will be described. Based on the operation of the whole NIC and the tasks of the bus-switching unit in particular, the requirements for the bus-switching unit are derived and the challenges are formulated. Then, possible theoretical approaches to the problems are discussed and compared. Finally, the results from the theoretical considerations are put together, in order to present a reasonable realisation for the bus-switching unit.

A short note on the glossary for this chapter is given here. In this chapter different parts of a whole have to be named. To avoid misunderstandings it is pointed out that the following four terms are used intentionally for a specific part and can not be used interchangeably.

Ethernet packet is a packet as defined and depicted in figure 2.2, including the preamble and the Start-of-Frame Delimiter (SFD). Packets are transmitted over the MII connection between the MAC and the PHY, and over the physical Ethernet link between the transmitting and receiving PHYs.

Ethernet frame is only the layer 2 presentation of the Ethernet packet. For received Ethernet packets frames are available after the MAC has processed them, and for transmitted packets frames are available before the MAC has processed them.

Memory slice names a DMA memory region at the host PC which is reserved to store a single Ethernet frame, and therefore is also referred to as frame buffer. The number of available slices is exactly the number of Ethernet frames that can be buffered on the device driver side.

Cell is used to indicate a part of an Ethernet frame. During DMA transmission between the device driver located frame buffer and the frame buffer in the MAC, an Ethernet frame is disassembled into several equisized cells and reassembled in the destination frame buffer.

Furthermore, for the ease of reading, in the following the full term for PCI Express memory read/write request TLP will often be shortened to read or write request, respectively, except for passages where the short form could lead to confusion. PCI Express completion TLP will be shortened to completion, and Altera PCI Express MegaCore Streaming Interface will be shortened to PCI Express interface.

3.1 Principle of Operation

To understand the requirements for the bus-switching unit, the principle of operation of the complete hardware system has to be understood. A device driver at the host PC is responsible for the communication between software applications on the host PC and the hardware NIC. During the initialisation phase, the device driver reserves a virtual memory area for memory mapped access to the registers of the NIC, as well as a memory area for the DMA transfers. These two different types of data transfer are described in the following sections.

In the following descriptions only the major issues are pointed out to get an overview of the functionality, with special emphasis on the requirements for the bus-switching unit. To improve the readability the references to the specific detailed descriptions in the text are saved, but it is reminded that the used protocols were already described in section 2.1, while a detailed description of the actual design of the bus-switching unit and its functional particularities is provided in section 5.3.

3.1.1 Register Operations

This type of data transfer is used for reading and writing the status and configuration registers, both of the MAC as well as of the CSC. All register read and write operations are initiated by the device driver at the host PC. Figure 3.1 gives an illustration of the data paths and the respective data formats.

To point out the conceptional differences between PCI Express and AHB the packet and transmission formats of the two protocols are sketched in the figure, including the most important fields. In the PCI Express TLP, starting with the least significant bit on the right side, all control information is transmitted in-band and therefore is included in the TLP header. In contrast, the AHB transfer uses out-of-band signals for control information, which are dedicated signal lines in parallel to the data lines, as illustrated in the figure.

For register operations a memory read or write request TLP is sent over the PCI Express link, initiated by the device driver. As there are different TLP types which have to be treated and routed differently, the bus-switching unit first has to analyse and route the TLP to the respective module. Figure 3.2 shows the message flow only inside of the

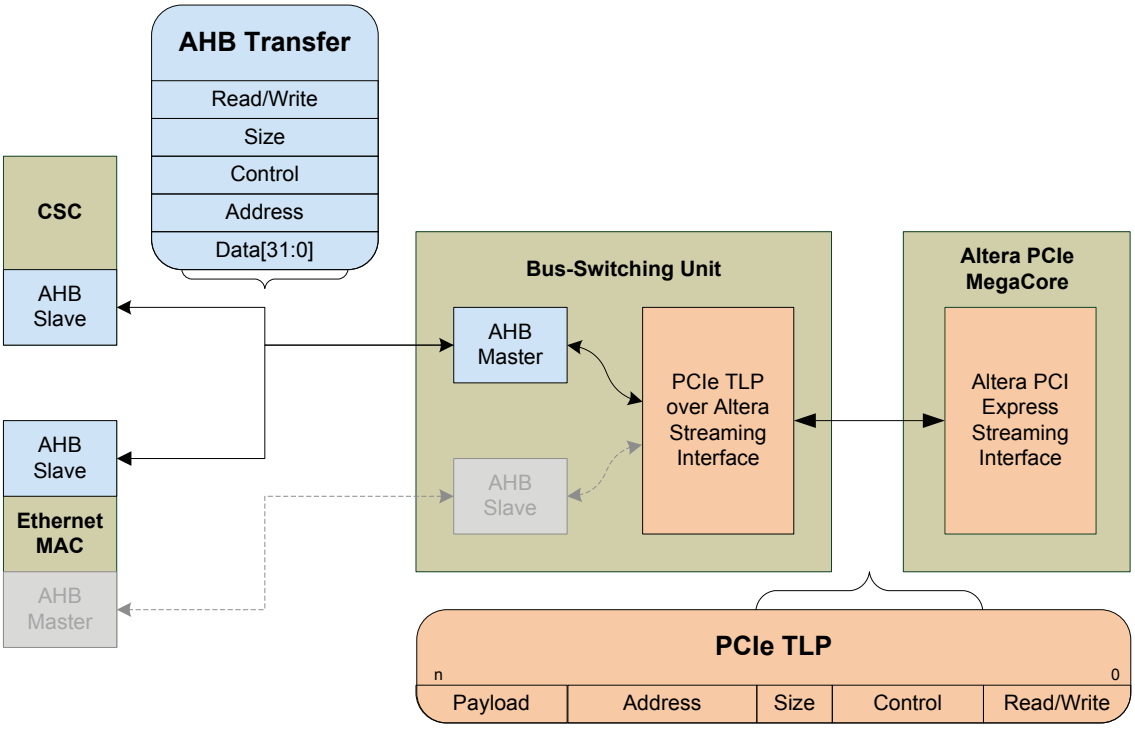


Figure 3.1: Register read and write operation

bus-switching unit. The involved TLPs are depicted, and the small number indicate their respective sequence for a register read operation.

As depicted in figure 3.2, an incoming read or write request TLP initiates a sequence in the bus-switching unit. The small numbers in the figure mark the sequence of the information flow in chronological order. The number 1 at the PCI Express connector arrow indicates that the first step is the reception of a memory request TLP which has to be analysed by the bus-switching unit. If it is identified as a read or write request, it is forwarded to the AHB master. There the address is decoded and an AHB transfer is started.

In the case of a write request, the AHB slave side takes care of writing the value to the corresponding register, if it is available. For the bus-switching unit, the task is completed with the end of the AHB transfer. In case of an error, there is no notification of the device driver, as the PCI Express write request is a posted request with no reply. In case of a read request, the AHB master initiates a read transfer. The master then has to wait for the slave to present the data on the data line. A PCI Express memory read request is a non-posted request which requires a reply. Therefore, the bus-switching unit then assembles a PCI Express completion packet, using the data received from the AHB slave, and sends the read data back to the device driver. As the read operation is a non-posted operation, the device driver can be informed about a probably occurred error, e. g. a non-existing register address.

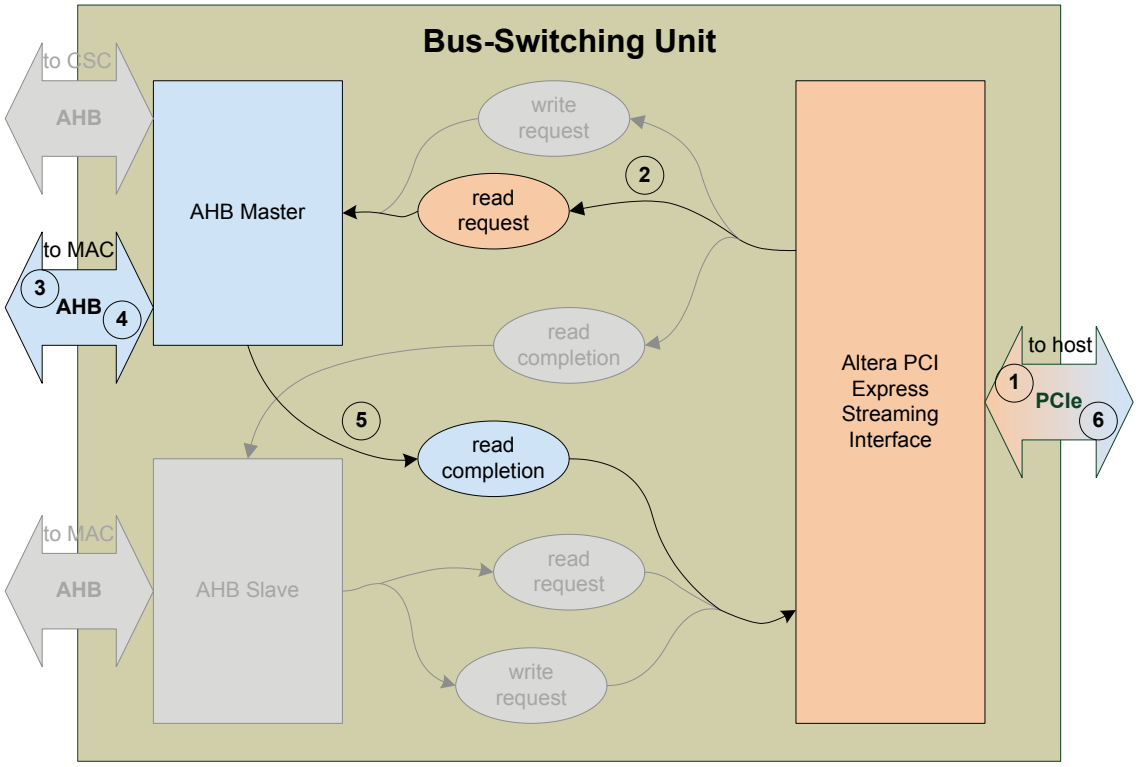


Figure 3.2: Information flow in the bus-switching unit for register operation

To transmit a PCI Express TLP, the PCI Express interface requires 64-bits of data at each clock cycle to be present at its data lines. First the 3 DW header information, then the data payload. As all register read transactions are limited to only one DW (32-bits) of data the data payload can be presented in a single clock cycle. The transmission of the assembled PCI Express completion TLP to the PCI Express interface only starts when the data already is present at the AHB interface. Therefore, the TLP can be presented to the PCI Express interface at once as it is required, with no need for stalling by inserted wait cycles.

3.1.2 Burst Direct Memory Access Operation

The second type of data transfer concerns the actual Ethernet packet traffic between the MAC and the host PC. Figure 3.3 gives an illustration of the data paths and the data formats for the burst type operation. All data transfers of this kind are initiated by the Ethernet MAC. As DMA is used for data exchange with the memory area at the host PC, interrupts and interrupt status registers have to be used to signal events to the device driver.

The DMA memory is split into slices of the size of the longest possible Ethernet frame. Each slice is meant to store exactly one Ethernet frame. The number of available slices

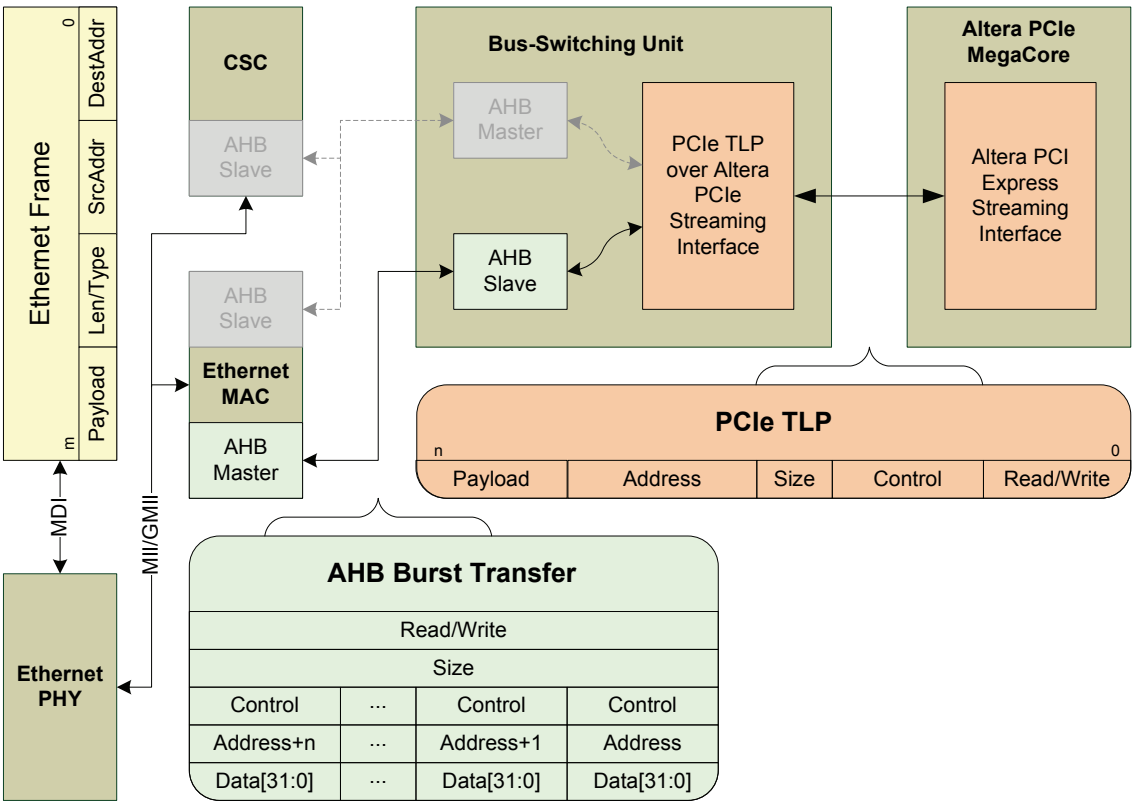


Figure 3.3: Ethernet over AHB-burst-transmission

for transmit and receive frames can be varied, but is limited to the maximum number of available buffer descriptors in the MAC. The buffer descriptors are special registers, each storing the address and status information of a single DMA memory slice. These buffer descriptors as well as a register storing the number of available DMA memory slices are written by the device driver using register write operations during the initialisation phase.

Each buffer descriptor consists of two 32-bit wide registers. One stores the DMA address of the corresponding memory slice, which is the frame buffer, and the other stores control and status information for the corresponding buffer. The control and status register is used for some configuration and basic error reporting, but mainly to store the size of the content of the buffer and to report whether it currently contains valid data or not.

3.1.2.1 Receiving an Ethernet Frame

When the MAC receives an Ethernet packet on the MII interface, it processes the frame according to the specification of the Ethernet standard 802.3 [5]. When it is found to be a valid packet, which is intended for the host PC, the MAC interface splits the Ethernet frame into smaller cells. Each cell is transferred using a single AHB burst transfer, and then is packed into a single TLP and transmitted over the PCI Express connection. The size of a cell can be configured in advance by writing the corresponding control register.

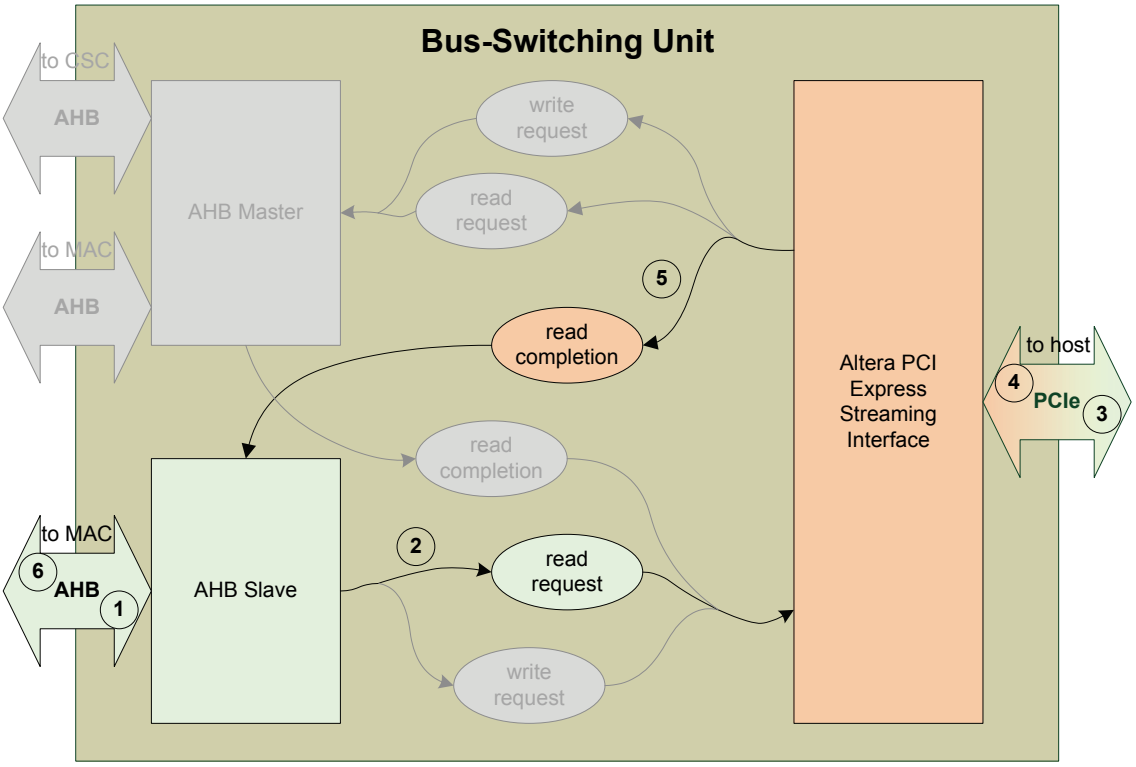


Figure 3.4: Information flow in the bus-switching unit for DMA operation

Figure 3.4 shows the sequence of the information flow inside of the bus-switching unit for DMA burst operations. When the MAC receives a valid Ethernet packet (number 1 in the figure) it initiates an AHB burst transfer. The base address for the transfer is retrieved from the buffer descriptor and is increased for each consecutive transferred DW of data. At each clock cycle 32 bits of data are transferred, until the total amount of the predefined cell size is reached. This is illustrated in figure 3.3 by showing several 32 bit wide data blocks side by side in the light green "AHB Burst Transfer" data format box.

The AHB slave in the bus-switching unit then assembles one PCI Express TLP per AHB burst transfer, i. e. the data of a cell size (number 2 in figure 3.4). The AHB side-band signals are used to form the TLP header, while the content of the AHB data lines is transmitted in the payload. The AHB interface only presents 32 bits of data during one clock cycle, while the PCI Express interface requests 64 bits of data per clock cycle. The PCI Express interface does not permit stalling by inserting wait states. Therefore, all data for a complete TLP has to be available before a transmission can be started. Hence, the data has to be collected in a buffer before a TLP can be scheduled for transmission. The PCI Express interface then transmits the TLP over the PCI Express connection (3), and the DMA controller on the receiving side writes the data to the addressed DMA memory slice.

This procedure (step 1–3) is repeated for each cell, until all data of the complete Ethernet frame is transmitted. After the last cell has been transmitted, the interrupt status register is set accordingly to indicate that the received Ethernet frame is available in the addressed DMA memory slice. The status part of the corresponding buffer descriptor is set as well to indicate the size of the Ethernet frame and to mark the memory area as used. Then the interrupt line is asserted to trigger the device drivers interrupt service routine.

3.1.2.2 Transmitting an Ethernet Frame

If there is an Ethernet frame to be sent, the device driver writes the data into the memory area assigned to the NIC for DMA transfers. It then writes the corresponding buffer descriptor in the MAC, informing the hardware about the presence of data, its size and address. Consequently, the AHB master in the MAC starts an AHB read burst transfer. The sequence of the information flow again is depicted in figure 3.4. When a read burst transfer is started (number 1 in the figure), the bus-switching unit has to assemble a read request TLP (2) and send it over the PCI Express interface (3). In a read request itself no data is included, only the according read request header has to be sent and some identifying data has to be stored in order to be able to match the incoming completion to the outgoing read request. The size of the requested data is a cell size, which is controlled by the same control register that also controls the size of the cells in which received Ethernet frames are split up to.

The DMA engine of the host PC handles the read request and reads the data from the assigned DMA memory slice. The data is packed into a completion TLP and returned to the PCI Express interface of the NIC (4). This incoming completion has to be analysed in the same way as the incoming read and write requests of the register transfer are analysed. If it is identified as a completion, it is checked whether it is the expected completion and fits to the previously sent read request. If so, it is routed to the AHB slave (5), which only then can present the read data to the AHB data line (6). The slave presents the data in units of 32 bits to the AHB master, until the data of a cell is transferred and thus the AHB burst transfer is completed. The AHB master on the MAC side receives the data of the burst and stores it in a frame buffer.

This procedure (step 1–6) is repeated as well until all data cells stored in the DMA memory slice are transferred. When all data is transferred, the complete Ethernet frame is available in the frame buffer of the MAC. It then sends an Ethernet packet over the Media Independent Interface (MII) to the PHY, which puts the data on the physical Ethernet link. As soon as the MAC sent the complete Ethernet packet to the PHY, the interrupt status register is set and an interrupt is issued to signal the device driver that the packet has been sent.

3.2 Challenges

Based on the principle of operation of the Ethernet NIC, the basic requirement for the bus-switching unit is clear – it has to manage and translate the data-flow between the different bus interfaces. Still, a number of challenges are faced. Those challenges are identified and described in this section. Theoretical backgrounds are considered to support fundamental design decisions which have to be made before the unit can be designed and implemented.

Based on the description of the principle of operation, it becomes obvious that for all incoming PCI Express TLPs similar challenges are faced. The PCI Express and the AHB interface have to be rate-matched, as the PCI Express interface presents 64 bits of data per clock cycle, while the implemented AHB interface accepts only 32 bits of data per clock cycle. Regardless of the type of the TLP, consecutive TLPs of the same type (memory request or completion) are switched to the same module. Therefore, care has to be taken that neither an ongoing operation is disturbed by a new incoming packet, nor that a packet is lost while a previous operation is still being processed. During an ongoing operation, no new TLP can be handled by the module. Therefore, the PCI Express interface has to be prevented to present data when it can't be processed. As described in section 2.1.4, the PCI Express interface is only required to react on a change of the controlling ready signal within three clock cycles. Therefore, some buffering mechanism has to be applied to avoid data loss during this reaction time.

Outgoing PCI Express TLPs can result from both types of transfer, register operations and DMA burst transfers. To be able to transmit TLPs, both modules have to use the same shared resource: the transmission part of the PCI Express interface. As described, the register operations and the DMA burst transfers are initiated independently from each other. Furthermore, the AHB master and slave modules in the bus-switching unit have no knowledge of each other. Hence, there is no correlation between their sending of TLPs. Therefore, some kind of locking algorithm has to be applied to prevent the two modules from writing their data simultaneously to the shared data line.

Considering the above, the main requirements and challenges for the bus-switching unit can be identified as

- Protocol translation
- Packet-switching
- Bus arbitration
- Rate-matching
- Prevention of data loss

Each of these challenges are discussed now in detail by analysing requirements, considering theoretical backgrounds and analysing possible solutions.

3.2.1 Protocol Translation

The term "protocol translation" means the interconnection of two interfaces employing two different communication protocols in a way that they can exchange data. Therefore, some logic has to be developed with detailed knowledge about both protocols. For each direction, the data of one protocol interface has to be read, transformed (or translated) to a format that is understood by the other protocol interface and written to the other protocol interface. Therefore, a protocol translator consists of a fully featured interface for both protocols and some logic to translate the data formats and control information.

The protocol translation can either be done directly inside of the AHB interface control module or in an external module, which is connected to and controlled by the AHB interface control module. The advantage of the latter type is a more modular design, which simplifies especially the reuse of the AHB modules.

On the other hand, a too fine grained modularisation has disadvantages as well. When closely interconnected functional elements are split into many small sub-modules, it gets very difficult to understand the overall operation. Therefore, maintainability is decreased.

A quality measure for decent functional partitioning is the information-flow-measure [24], which measures the amount of exchanged information between different sub-modules. If different sub-modules operate on the same, or on subsets of the same information sources, a large amount of information has to be exchanged. In such a case, it might be better to combine the sub-modules to get rid of the information exchanging signals [24]. The combined module will have a slightly increased complexity in terms of Mc Cabe's cyclomatic number [25], nesting level and code length compared to the complexity of the single sub-modules, but the overall understandability can be increased by processing closely related functions in the same module. Splitting up the AHB interface control and the protocol translation would lead to a high number of the information-flow-measure, as they would both need to operate on the AHB data and address lines, which together already make up 64 bits of exchanged information. Still, maintainability and understandability are subjective measures, depending on the preferences of the developing engineer. Therefore, optimising for the mentioned measures can not guarantee a perfect design, but the measures can be used as guidelines for design decisions.

A different consideration further weakens the argumentation for a separate protocol translation module. The main advantage of the separate approach would be the simpler reuse of the AHB interface module. This argumentation might hold true for an AHB interface without the need for wait states. But generally speaking, there is no generic AHB interface implementation which can be taken from one unit and be reused without change in a different unit. Even the AHB slave interface for register operations in the MAC and the CSC are different. The reason for that is the interface nature of the AHB module. The definition of the bus itself for sure is generic. It even offers the possibility of wait state insertion on both sides, master and slave, to be able to adjust to a large variety of applications. But the AHB interface module is the interface which connects the application

specific part to the generic AHB signals. Therefore, the interface module has to be aware of the application specific needs. For example, the ready signal has to be controlled by the slave interface in a way that fits the specific behaviour of the application. Therefore, the slave interface module always has to be adapted to meet the application's requirements and can't be reused without alteration.

A separate protocol translation module can't be reused as well. It is a translation module specifically designed for the translation of AHB to PCI Express and vice versa. Therefore, it can only be used together with an AHB interface. If such a protocol translation unit is needed at a different place, the AHB module is needed as well, and therefore a separate protocol translation module does not augment reusability at all. From that point of view, the advantage in terms of reusability is too small to justify the splitting of the AHB slave and the protocol translation module.

3.2.2 Packet-Switching

The term "packet-switching" is chosen on purpose to point out the similarity of the nature of the given problem to well known interconnection problems of packet-based networks. Even though the general problem here appears in a very simplified environment, the basic requirements stay the same. The incoming packets arrive on one port, they carry some kind of address information and they have to be forwarded to an outgoing port according to the address information. Therefore, similar techniques as in the well known field of Ethernet switching can be used and are discussed in this section. Although packets on the processing level (layer two of the OSI reference model) usually are referred to as frames, they still are packets in the sense of individually addressed data and information capsules.

Because of the very simple nature of the possible linkage of the modules, the packet-switching can, similarly to the protocol translation, also be realised integrated in the respective AHB interface control modules or as a separate module. The argumentation about the reusability of the AHB interface module is similar to the considerations about the reuse of the AHB interface module with the protocol translation.

Figure 3.5 shows the principle operation of a separate switching module. The packet-switcher observes the type and address of the incoming PCI Express TLP (symbolised by the magnifying glass in the figure) and routes the TLP to its intended destination. Therefore the path to the intended destination is opened, symbolised by the open semaphore in the figure, and the path to the not-intended destination is closed.

A separate PCI Express TLP switching module, in contrast to the protocol translation module, can easily be reused. Additionally, a separate switching module brings advantages also in terms of scalability. Compared to the protocol translation, the argumentation for understandability with two separate modules is also different. While the protocol translation unit is tightly coupled to the timely sequence of the AHB slave module throughout the complete transfer, the packet-switching module only has to take action at the very beginning of the reception of a PCI Express TLP and is not at all involved in the further

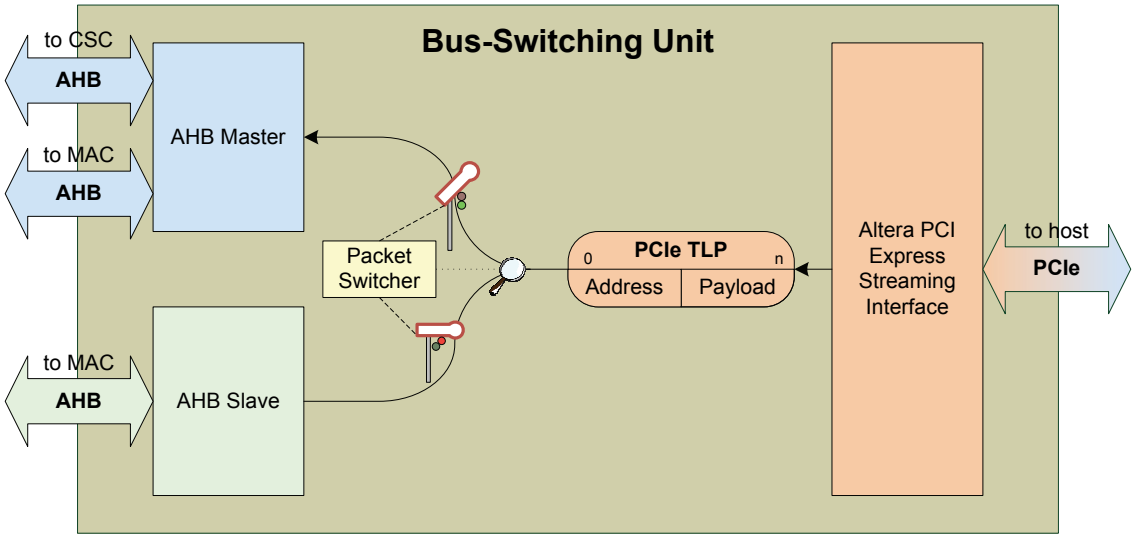


Figure 3.5: Operation principle of a separate switching module

processing of the TLP by the AHB interface. Therefore, the functional coupling is not of a parallel nature, but of a sequential. Hence, the separation in two distinct modules does not lead to an obfuscated functional partitioning with reduced maintainability, but to a clear, sequential functional modularisation.

For the separate switching unit, the decision about the switching technique still has to be taken. The methodology for switching can be space division or time division switching. Space Division Switching (SDS) allows multiple packets from different inputs to different outputs to be switched at the same time. Time-Division Switching (TDS) uses a shared medium for all inputs and outputs. Therefore, the switching has to be defined by granting access to the shared medium at different instants of time. Further, the packet-switching of the PCI Express TLPs can either be done in a store-and-forward or cut-through like manner. Store-and-forward means, that a complete packet is stored upon reception. Only after the complete packet is received it is forwarded. Cut-through switching means, that the packet is forwarded as soon as the destination is known. As the PCI Express type field, which in this case is the decisive factor for the destination module, is at the very beginning of the TLP header, the packet can be forwarded immediately.

To be able to judge whether switching techniques known from Ethernet are appropriate for the PCI Express TLP switching, the differences of the environments have to be considered. In Ethernet switches, the main argument for a store-and-forward technique, in contrast to cut-through, is the avoidance of unnecessary data traffic in the case of corrupted packets [13]. Considering the CSMA/CD access method of Ethernet, the probability for broken packets in a not fully switched, full duplex environment is considerably high. This is not true for the PCI Express TLPs arriving in the bus-switching unit. Referring to the ISO OSI reference model [15], PCI Express TLPs are layer three packets. Broken packets are already filtered out at the lower layers. Therefore, it is not expected to receive any broken

packet at all. Hence, there is no need to prefer a store-and-forward like technique, instead a cut-through like technique seems to be more appropriate.

For the decision about space and time division switching, the requirements for the bus-switching unit have to be considered as well. Though there is only one port for incoming PCI Express TLPs, still packets can arrive faster than the data sink can process them. Therefore, space division switching would still be possible to allow the processing of two packets destined to different modules at the same time. However, achieving the highest possible throughput is not the main intention of the design. Even if it was, under normal operation conditions mostly packets of DMA bursts are involved. With this kind of traffic, SDS would not have any advantage over TDS, because all packets have to be routed to the same receiver. TDS can be realised by input buffering or output buffering. TDS needs less logic than SDS, it is easier to understand and thus to maintain. As already discussed, an input buffer is needed anyway. Therefore, even though the maximum theoretical bandwidth can not be achieved by using input buffering [14], it is a reasonable choice for the bus-switching unit.

3.2.3 Bus Arbitration

Bus arbitration concerns the access control of a shared bus. In the case of the bus-switching unit, two modules have to access the same shared bus, namely the transmission line of the PCI Express interface. The two modules are the AHB master when sending a request and the AHB slave when sending a completion. Since both modules are AHB units, it would seem natural to use the arbiter module described in the AMBA® AHB [6] protocol specification. On a closer sight, it is obvious that this is not possible, as the shared bus is not part of an AHB system and thus not at all underlie the timing specifications of AHB. Therefore, a dedicated arbitration scheme has to be developed.

The same question, together with a similar argumentation arises, whether to have a separate arbitration unit or an integrated functionality with dedicated signalling. The possibilities for arbitration are numerous. Simple, purely sequential operation of all modules might be a solution, as well as more complex, quasi pipelined operation, blocking only when really necessary. A trade-off between design simplicity and performance has to be made. Considerations about these topics are presented in section 3.3.2.

3.2.4 Rate-Matching

Rate-matching deals with the connection of two distinct interfaces, which operate at different data rates. In the case of the bus-switching unit, the general problem is simplified, because both interfaces use the same clock, which usually is not the case. Therefore, only the data width is accountable for the different data rates. There are two basic methods to perform rate-matching in such a case: buffering and stalling of the interface with the higher data rate. Stalling slows down an interface by inserting idle wait cycles. This

might deteriorate the system performance, especially if the faster interface is a shared medium. Furthermore, the interface specification has to support this kind of technique. On the other hand, if the faster interface can not be slowed down by stalling, buffers might need to be very large to prevent data loss. Therefore, a trade-off between buffer size and system performance has to be made.

3.2.5 Prevention of Data Loss

Finally, all of the previous mentioned challenges have to accommodate for the one ultimate requirement, the prevention of data loss. The blocking of specific functional units might lead to data loss. Therefore, buffers of an appropriate size have to be implemented. Both, the AHB as well as the PCI Express interface support back pressure, which allows slowing down the respective data sources. Using this technique greatly reduces the required buffer size, since only data which arrives during the propagation delay of the back pressure mechanism has to be buffered.

3.3 Possible Bus-Switching Architectures

The main challenges for the bus-switching unit have been identified and discussed. Now, some considerations concerning the architecture of the bus-switching unit as a whole have to be taken. The design decisions concerning the previously mentioned challenges can not be made completely independent of each other. The whole unit has to meet the requirements and an efficient design can only be made if all dependencies and trade-offs are considered together. Therefore, not all possible combinations will be discussed, rather more general considerations about the complete unit will be presented.

3.3.1 System on a Programmable Chip Builder

The necessity to connect different units of a design is a very common problem. For common problems standard solutions are desirable. Altera provides such a solution, the so-called System on a Programmable Chip (SOPC) Builder [26]. It is a tool to automatically interconnect design modules. Each module is required to have an Avalon bus interface. The tool then generates the interconnect fabric, including address decoding and bus arbitration. It can even be configured to use dynamic bus-sizing logic to connect narrower slaves to wider masters and vice versa.

There is a possibility to generate the Altera PCI Express MegaCore Unit with an Avalon Memory Mapped (MM) interface, which is designed to be used together with the SOPC builder [27]. This interface tackles all the above mentioned challenges. It has a protocol and address translation unit, it uses FIFO memory structures in both directions, as well as large RAM blocks of 1 KB size for completion buffering and reordering.

The SOPC builder appears to be a powerful tool to quickly get an evaluation system up and running. However, there are a couple of drawbacks remaining. For example, the individual units need to be equipped with an Avalon bus interface, the design needs a relatively large amount of resources, and possibilities for individual optimisation are missing. Concerning the Avalon bus, the existing AHB interfaced modules would need to be packed into additional bus mapping units, to convert AHB to Avalon and vice versa. Although AHB and Avalon are quite similar, there are differences mainly in the control signals and in the requirements for the control signal timing, which leads to the need of a more complex bus protocol translator that goes beyond a simple signal mapping. Altera did offer an AHB to Avalon bus interface mapper, but this device has been discontinued.

Therefore, still a protocol translator would have been needed in order to facilitate the features of the SOPC builder, though the complexity of the AHB–Avalon translator is definitely lower than what is needed for the PCI Express translation. Furthermore, the PCI Express interface generated for the use with the SOPC builder, as well as the generated interconnect fabric, offer much more functionality than needed for the specific application of the network interface card. The requirement for additional logic and memory is not a problem with the very powerful FPGA device of the evaluation board, but might be a problem with smaller and cheaper FPGA devices for a probably upcoming product development. For those reasons, it was not an option to use the SOPC builder design variant in the course of this diploma thesis.

3.3.2 Bus Arbitration Considerations

As the automatic generation of the bus interconnection is not reasonably feasible, the bus-switching unit has to be designed from scratch. From the above mentioned challenges, the design of the bus arbitration is a key factor, since most of the other design choices are affected by the bus arbitration behaviour.

The realisation with Finite State Machines (FSMs) is the most reasonable variant for the design of the AHB bus interface control modules, as well as for the protocol translation. Especially Moore machines are relatively easy to understand and therefore to maintain. All following considerations thus will be based on an FSM realisation of the mentioned modules. For a better understanding of these considerations, it is important to bear in mind that all Ethernet traffic uses AHB burst transfers, which are processed by the AHB slave module in the bus-switching unit. All register operations use the AHB master interface. For more details refer to section 4.2.

As already mentioned, there are plenty of possibilities to prevent concurrent access of different modules to a shared resource. Just a few examples are round-robin scheduling of the affected state machines, mutual exclusion, usage of semaphore signals with an additional mechanism to avoid the concurrent claim of the semaphore, usage of an protocol aware observing unit which controls the access to the shared resource, or the usage of a bus arbiter with dedicated control signals to request and grant bus access.

For the given requirements of the bus-switching unit, and bearing in mind the typical use case of mostly Ethernet traffic with only sporadic register access for DMA and interrupt control, a strict round-robin scheduling scheme seems not appropriate. Since an Ethernet frame is split to multiple cells, there are a lot of burst transfers, until comparably few register operations take place. Strict round-robin scheduling would unnecessarily deteriorate the throughput of the Ethernet traffic, reserving time for not required register operations. Thus, a demand-oriented approach seems to be more suitable.

Still, the blocking policy and control has to be defined. A very simple approach is mutual exclusion of the operation of the state machines of the affected modules. As long as one module's state machine is operating, the start of the other state machine is prohibited, until the first one has completed its operation. A simple prioritisation has to be made to prevent concurrent operation when both state machines encounter their respective starting condition at the same time. The prioritised state machine is always allowed to start, as long as the other state machine is in idle state. The not prioritised state machine is only allowed to start when the prioritised one is in idle state and the starting condition is not met. Therefore, the starting condition of the prioritised state machine has to be observed as well.

The mutual exclusion and prioritisation control can be realised by an external control unit or integrated in the state machines. In the case of an integrated control approach, the state machines can observe the necessary signals and combine those signals with the start request to decide on the start of its operation. The observation can also be done by an external control unit, the bus arbiter. The bus arbiter again can be realised as a Moore machine, which adds an additional clock cycle until the start signal can be asserted, or as a Mealy machine. In the actual case of only two units, which access the shared resource, the arbitration is so simple that it is not necessary to use a dedicated bus arbiter. On the other hand, an arbiter as a Moore machine increases scalability and simplifies the understandability of the design, and thus increases maintainability.

A different performance increasing blocking policy could take advantage of the fact that the shared resource is only utilised in a fraction of the operation time of the affected state machines. Thus, mutual exclusion can still be performed in a similar manner, but protecting the shared resource by blocking the other state machine only when the shared resource is effectively used. For this kind of blocking all state machines must be allowed to be stalled at any given time. This holds true for the bus-switching unit, because both AHB modules, the master as well as the slave are allowed to stall their respective counterparts. The slave can stall the master by deasserting the ready signal and the master can stall the slave by issuing a busy transfer. Therefore, each transfer can be stalled at any given time and it is possible to block a state machine in any given state. For this kind of bus arbitration, it is strongly recommended to use a dedicated bus arbiter unit. If no dedicated bus arbiter is used, the locking signals would have to be hidden in one or more specific states of the state machines, which greatly obfuscates the operation sequence, and thus reduces maintainability as well as scalability.

Chapter 4

Development System

This chapter gives an overview of the surrounding system components, which define the boundary conditions for the designed bus-switching module. First, the selection of the development platform to support the required Gigabit Ethernet connections is described. Then an overview of the different interconnections of the units of the legacy NIC hardware is given. To have a good overview of the overall system operation before going into the specific design details, the interaction of the device driver with the hardware is presented as well.

4.1 Choosing the Development Platform

First of all, a suitable hardware platform had to be chosen. The platform needs to support evaluation and measurements of the protocols mentioned in section 1.2. All commercially available evaluation boards, which match the requirements, use PCI Express to communicate with the host PC. Thus, the existing Ethernet NIC hardware design had to be adapted to be able to communicate with the host computer using the PCI Express protocol specification. Due to the different interface standards of the existing Ethernet NIC design on the one side and the Altera PCI Express core on the other side, a bus-switching unit had to be developed.

The evaluation system is required to host the existing NIC design and the bus-switching unit as well as the PCI Express protocol core. It has to be able to communicate with the host PC and to operate the physical layer protocols listed in section 1.2. For the hosting of the hardware only FPGA devices are suitable. All Mask Programmable Gate Arrays (MPGAs), no matter if sea of gates, standard cell, or full custom Application Specific Integrated Circuit (ASIC) based technologies are by far too expensive and inflexible for an evaluation design. Higher clock speed could be the only driving argument for an MPGA design, but is not the issue of this diploma thesis, since there are commercially available FPGAs which are fast enough.

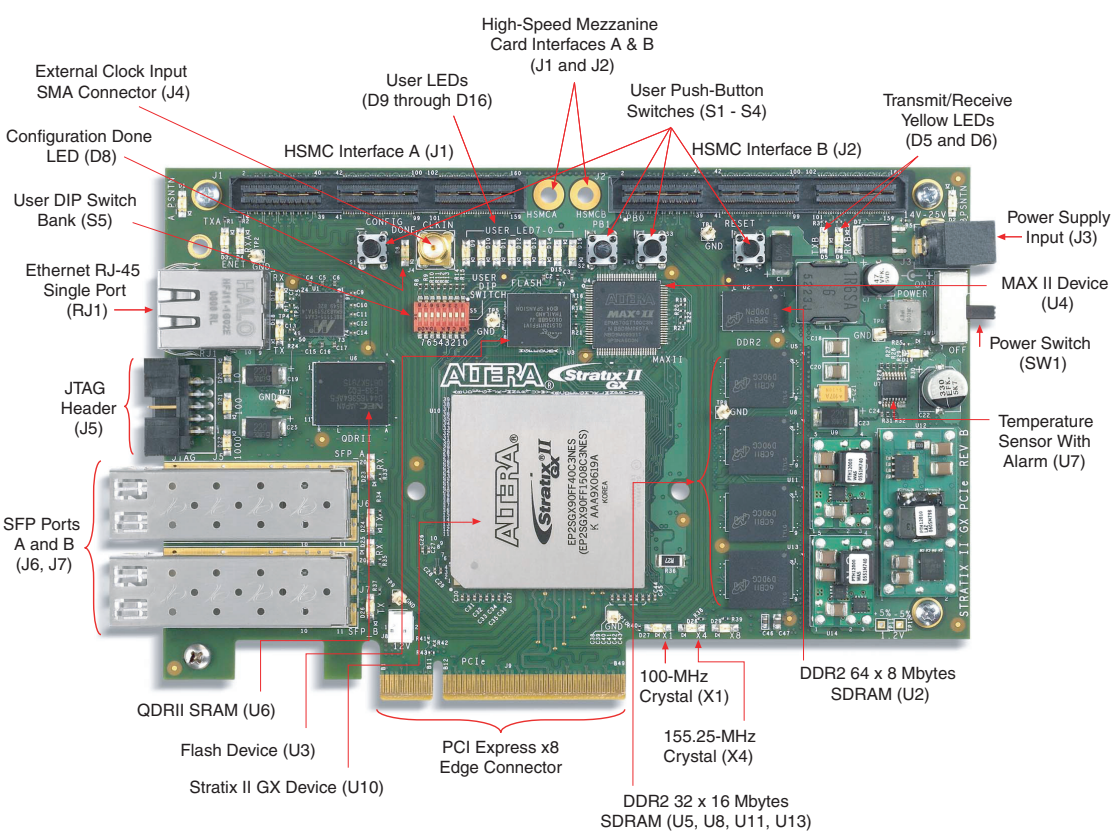


Figure 4.1: Altera PCI Express development kit, Stratix II GX Edition [28]

The use of a pure software design, which is operated on a microprocessor, is not applicable as well, as the high-precision clock synchronization depends on specific hardware in addition to the standard Ethernet hardware realisation. Hence, an FPGA based development platform was chosen. It fits the requirements perfectly, offers the flexibility to insert new modules to the existing hardware core, and it allows for immediate testing of newly developed design units.

Therefore, available products of known FPGA vendors were examined. The big players in the market definitely are Altera® and Xilinx®. Lattice®, an FPGA producing company, which mainly concentrates on market niches, like small, low cost FPGAs, was also considered. All three of them offer evaluation boards with Ethernet PHYs for 10/100/1000 Megabit/s and Small Form-Factor Pluggable (SFP) slots for optical transceivers. Another company on the FPGA market is Actel®, but in contrast to the other three companies, they currently don't offer evaluation boards with the required interfaces. As there were only minor differences between the three evaluation boards regarding price and capabilities, prior experiences with the mentioned companies led to favouring the most powerful solution, the Altera® PCI Express Evaluation Board with a Stratix® II GX device. Figure 4.1 shows the board with all its components.

FPGA

The Stratix II GX is a high-end FPGA device produced in 90 nm technology. It offers sixteen high-speed transceiver modules with integrated clock data recovery ability, 8B/10B en- and decoding, and Serializer/Deserializer (SERDES) capabilities, which is needed for the PCI Express connection, as well as for the connection to the optical link modules. It offers eight Phase Locked Loops (PLLs), more than 90.000 equivalent logic elements, and on-chip RAM blocks of more than four megabyte in total. With those key features, more than enough resources are available for the implementation of the complete hardware design of this diploma thesis, leaving plenty of possibilities for enhancing the design, or adding new features.

Board Features

Most of the components on the evaluation board are directly wired to specific pins of the FPGA. Among others, the mounted oscillators are of importance for this design. Although oscillators with different frequencies are present on the board, only the 100 MHz signal can be used for the FPGA design. However, because of the several PLLs inside the FPGA, and one SMA connector for external clock input, enough flexibility for different clock speeds is provided. The eight user configurable on-board Light Emitting Diodes (LEDs) also proved to be very useful for debugging reasons, although even more LEDs could have been used. The push-buttons were used for selective reset and for event triggering.

Peripherals

The relevant peripheral components on the evaluation board are the Marvell® 88E1111 Ethernet PHY and the PCI Express connector. The Ethernet PHY on this board supports the 10/100/1000BASE-T protocols as specified in the Ethernet standard IEEE 802.3 [5], which operate at different bit rates over twisted pair cable. The PHY itself would also support 1000BASE-X, which is suitable for the connection of an optical medium, but on the evaluation board the respective pins are not connected. However, dedicated SFP slots are provided for optical link networking.

4.2 Network Interface Card Hardware Architecture

Figure 4.2 shows the main components of the NIC, their respective location as well as the different protocols that are used at the different module interfaces. The lightly coloured blocks depict the physical and logical location of the units and components. The blue area depicts the evaluation board domain. All components in this domain are separate physical ICs or plugs and wires which are mounted on the evaluation board Printed Circuit Board (PCB) and are not modifiable, except for some programmable configuration

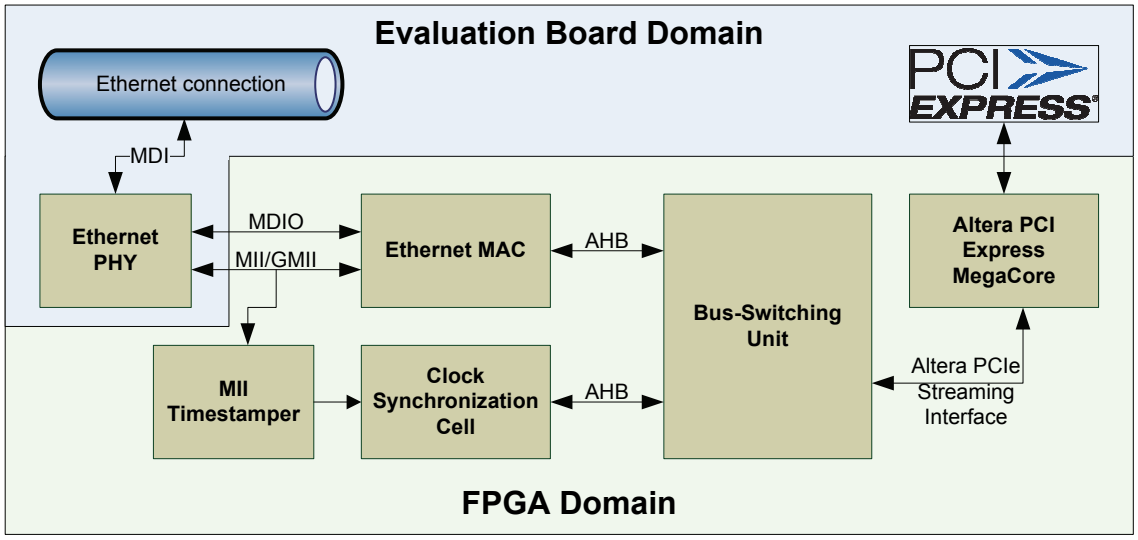


Figure 4.2: Module interfaces and hardware domains of the NIC

registers of the Ethernet PHY. The FPGA device is also part of the evaluation board and mounted on the PCB, but all functional units inside the FPGA are modifiable if the design source is available, or at least the hardware configuration can be selected for the Altera PCI Express MegaCore unit. As depicted in the figure, the MAC unit resides in the FPGA. Therefore, the MAC design of the existing NIC hardware design can be used, even though the PHY and the surrounding PCB has changed.

In contrast to the Media Dependent Interface (MDI) connection between the physical Ethernet medium and the Ethernet PHY, the MAC is connected to the PHY over the Media Independent Interface (MII) connection. As the name says, the MII connection is independent of the used physical medium, which connects multiple Ethernet nodes with each other. Therefore, the same MAC module can be used, regardless of the attached medium. However, the protocol is not completely independent of the used bit-rate. For data transmission rates of 10 Mbit/s and 100 Mbit/s, MII is used, only the clock is changed between 2.5 and 25 MHz, respectively. In case of Gigabit Ethernet, the GMII protocol is used, which is similar to MII, except for a little difference in the clocking scheme as described in section 2.1.1 and the use of eight data lines instead of four. When the PHY operates in Gigabit Ethernet mode, the GMII protocol is used, while in slower operation modes the MII interface is used. As the broader GMII is supported, the narrower MII can also be used on the same interconnection lines. All Ethernet data frames are transmitted either over the GMII or MII connection, while management data is transmitted using the MDIO interface.

On the right side in figure 4.2, the Ethernet MAC is connected to the bus-switching unit. For this connection the existing MAC design uses both, an AHB specification compliant master as well as a slave interface. Therefore, the bus-switching unit is required to provide also an AHB slave as well as a master interface as the corresponding counterparts. The

AHB interface is an intra-chip interface, as both sides, the master as well as the slave module, reside in the same FPGA device. The development of the bus-switching unit is the main part of this diploma thesis and is described in detail in section 5.3. Its main task is the arbitration and translation of AHB data transfers to the Altera PCI Express Streaming Interface and vice versa.

The Altera PCI Express MegaCore module is connected to the bus-switching unit using the Altera PCI Express Streaming interface, which is described in section 2.1.4. It is responsible for the transmission and reception of the PCI Express TLPs, as well as for all management and control functions of the PCI Express protocol. The module itself is a soft-core implementation, which is connected to the hard-core high-speed transceiver module. A soft-core representation of the module means that all required resource and interconnection information is available, but the fitter tool has to process this information together with the user defined application code and program the complete system into the FPGA. Thus, a soft-core realisation has no dedicated space in the hardware, no guaranteed timing and prolongs the fitting of the design. In contrast, a hard-core realisation provides a dedicated function in hardware with guaranteed and constant timing behaviour. It is a permanent part of the FPGA which is implemented already at the production time of the FPGA and therefore is neither modifiable nor removable. The advantage of a soft-core solution over a hard-core realisation is flexibility, as only the parts that are really needed have to be implemented, or the whole function can be left out completely if not needed, thus leaving more space for the application logic.

The soft-core part of the PCI Express protocol stack is connected to the hard-core part, which is a high-speed transceiver module provided by the Stratix II GX FPGA. The respective pins of the FPGA belonging to the transceiver are hard-wired to the PCI Express connector on the edge of the evaluation board. The PCI Express connector is plugged into the host PC, enabling communication between the NIC and the host PC.

4.3 Device Driver Operation

To complete the overview of the operation of the system, a brief explanation of the functionality of the device driver is given as well. The described functionality of the device driver is emulated by the fast emulation of the PCI Express connection, which is described in detail in section 6.2.

During the initialisation phase the device driver first reads the version number of the MAC and the CSC to make sure the driver version is compatible with the hardware version. Then some registers are set to their initial values, including the Ethernet PHY registers, which are mapped to the MAC register area, and the MAC configuration for the Ethernet link. Important for DMA transfers is the initialisation of the interrupt mask and enable registers, the burst size defining the cell size for AHB transfers, the number of available DMA Ethernet frame buffers and corresponding buffer descriptors, as well as the buffer

descriptors themselves. On start-up, the device driver reserves memory for the DMA transfers in the host PC's memory, according to the specified number of frame buffers. It then writes the start address of each frame buffer to the address field of the corresponding buffer descriptor.

Ethernet Frame Reception

When an Ethernet frame is received by the MAC, it is processed as described in the previous section 3.1.2 and written to the DMA memory. A counter is implemented to use the buffers in ascending order of the buffer descriptors. When the buffer descriptor with the highest number is reached, the one with the lowest number is selected as the subsequent descriptor. As soon as the complete frame has been written to the memory the empty bit of the currently used buffer descriptor is cleared to indicate that valid data is present in the buffer memory, and the length field is set according to the size of the received Ethernet frame. The MAC then sets the "receive frame" bit in the interrupt source register and generates an interrupt, informing the device driver about an important event. As only a single interrupt is used, the device driver has to read out the interrupt source registers of the MAC as well as of the CSC, which is also capable of generating interrupts.

The device driver first reads the interrupt source register and immediately writes back to clear the interrupt. When reading the device driver sees the "receive frame" bit of the interrupt source register of the MAC asserted and therefore reads out the current buffer descriptor. The device driver has a counter on its own and keeps track of the currently used buffer descriptor. The device driver now sees the cleared empty bit of the buffer descriptor and therefore knows that valid data is present at the memory address which is assigned to the buffer descriptor. The device driver then reads the buffer memory and forwards the content to the network subsystem. It then sends a write request to set the empty flag of the buffer descriptor again, indicating that the data has been read and the buffer memory can be used for new incoming Ethernet frames.

It is possible that during the currently described process of handling the reception of an Ethernet frame another frame arrived and was written to the DMA memory corresponding to the subsequent buffer descriptor. This might even have happened before the previous interrupt could be cleared, and therefore the reception of the new frame could not generate an interrupt signal. Therefore, the device driver keeps on reading the status of the subsequent buffer descriptors and processing possibly present data, until a buffer descriptor with a set empty bit is reached.

Ethernet Frame Transmission

When the device driver has to transmit an Ethernet frame, it first has to write the data from the network subsystem to the DMA memory for transmit frames. Similar to the receive frame buffer, the device driver and the MAC hardware independently keep track

of the currently used buffer descriptor. The device driver always chooses the memory assigned to the currently used buffer descriptor, with the buffer descriptors being selected in ascending order.

The device driver then sends a write request to the MAC, clearing the empty flag of the corresponding transmit buffer descriptor. The MAC realizes the clearance of the empty flag and starts an AHB burst read transfer to read the data from the DMA memory at the address specified by the buffer descriptor. When the complete frame has arrived at the MAC, the MAC starts the actual Ethernet transfer, sets the interrupt source register to indicate the frame was sent, sets the empty flag of the buffer descriptor and generates an interrupt to inform the device driver of the new event. The device driver again clears the interrupt and reads the transmit buffer descriptors. As the driver knows how many transmit buffer descriptors were written, it checks whether all of the used transmit buffers are empty already, or it stops the read out at the first buffer descriptor indicating a non-empty memory.

Because of the independent tracking of the currently used buffer descriptor in the hardware MAC unit, as well as in the device driver, a lost write request to a buffer descriptor would lead to an inconsistent view of the used buffer descriptors in the driver and the hardware. This can lead from degraded performance to permanent failure. Therefore, it is of utmost importance to ensure the reliable data transmission from the device driver down to the hardware register, including PCI Exrpress transmission, protocol translation and AHB transfer.

Chapter 5

Design and Implementation

The previous chapters described the practical and theoretical framework of the whole system. In chapter 3, possible approaches and solutions were presented. In this chapter, the actual design of the bus-switching unit is described, regarding the theoretical possibilities as presented in chapter 3. It is also described how a typical work flow for FPGA design was applied to the current problem, and which adaptations had to be made to fit the given tasks. Moreover, traps and pitfalls that occurred on the path to the successful implementation are discussed, and guidelines on how to avoid some of those problems in future projects are presented.

5.1 Design Work Flow

A typical design workflow for FPGA based design consists of several stages, while some of them have to be repeated in an iterative manner. Figure 5.1 shows the flow diagram of a typical design flow. As depicted, simulation is a central element which is performed at

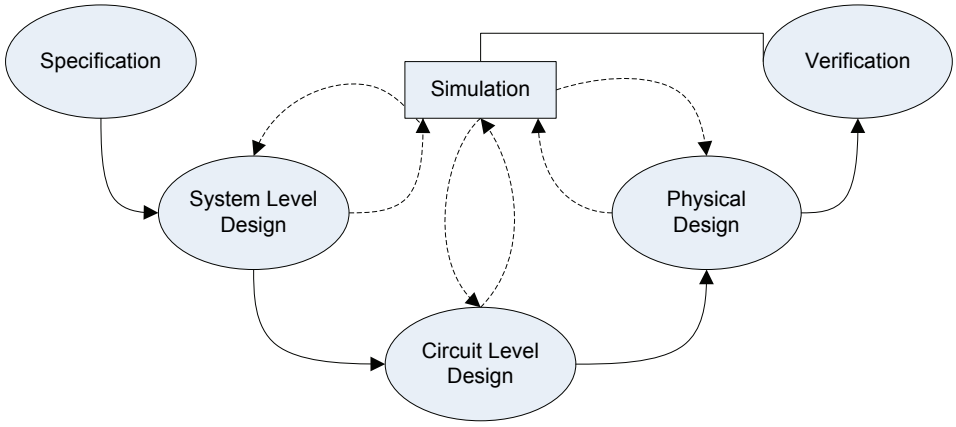


Figure 5.1: Typical design flow

different levels for the different stages of the design. The bridging line between simulation and verification symbolises that simulation is an integral part of verification.

In contrast to a design from scratch, for this diploma thesis an already existing, working hardware design was used as a base to start from. The original hardware design was developed for a Lattice FPGA device, whereas the new design has to work on an Altera Stratix II GX device. Although the Very High Speed Integrated Circuits Hardware Description Language (VHDL) is designed to be mostly independent of the used technology and hardware platform, there are still design components for which the used platform has to be considered. Such elements, for example, are memory blocks, which could be described in a generic manner, but can be used more efficiently if the code is optimised for the specific FPGA technology. Most FPGA vendors offer tools to create specifically optimised code for their respective devices. Another example are on chip PLLs or other device specific function blocks, like high-speed transceivers.

The timing of such device specific elements is very likely to vary from device to device. The timing of modules usually is the most crucial part of a hardware design. Therefore, a changed timing behaviour of a single element bears the potential to harmfully effect the functionality of the complete system, which might end up in a total system failure. To minimise the influence of a single element, it is desirable to design the system in a way that timing dependencies are limited to only directly connected components. Due to multi-stage timing requirements, or due to critical-path-optimisation, this is not always possible, and there might be more complex timing dependencies. In order to get a stable, proven base to start from, it is important to first perform and verify the transition of the system to the new platform, before any other modules are added or modified. In this diploma thesis, it was not possible to test the original system design with the new, adapted module parts on the actual hardware platform, because the evaluation board supports PCI Express only, whereas the original system design is based on PCI. Nevertheless, it was possible to use the original simulation environment to test the new modules, as it offers the simulation of a PCI bus system as well.

This first step of the design phase guaranteed, that the changed device specific modules did not affect the timing in a destructive way. This knowledge is very important, as it offers a trusted system base to start from. Any new module or additional modification adds new error sources. The more changes are performed at a time, the more complex error conditions can become. For a successful system design, it is of utmost importance to keep the modification steps as small and isolated as possible.

5.2 Hardware Transition

Before the real development of the new design could start it was necessary to generate a stable, proven platform to start from. The existing hardware design had to be transferred from a Lattice LFXP20 to an Altera Stratix II GX FPGA device. To do so, the hardware

platform dependent elements had to be identified. Basically, they can be divided into two groups. The first group contains the elements dependent on the FPGA device, namely

- RAM
- FIFO structures
- PLLs

and the second group contains the elements dependent on the hardware components present on the evaluation board, namely

- Quartz oscillators
- Ethernet PHY
- PCI Express connector.

5.2.1 FPGA Device Specific Alterations

The group of FPGA device specific elements depend on the internal structures of a specific device. For example, RAM cells basically consist of a memory array with a small configurable surrounding logic. The arrangement of the memory cells, as well as the organisation and type of the surrounding logic parts, vary from device type to device type. The surrounding logic parts are used to form the so called glue logic, which e. g. takes care of the correct addressing of the underlying memory, the handling of simultaneous read and write accesses etc. Because of the varying physical implementation of those memory units, standard memory blocks used in applications are dependent on the used FPGA device. FIFOs are likewise dependent as they consist of RAM blocks with some additional control logic. Beside these similarities, RAM and FIFO elements still have to be treated separately if no generic FIFO control logic implementation is used. In both cases, the original timing behaviour has to be extracted and the corresponding elements in the design for the new platform have to meet the original timing requirements. Except for the naming scheme, the interfaces usually are quite similar between different vendors, but attention has to be paid to the different timing models. The elements might differ in the handling of the input and output signals and thus differ in the timing behaviour. They might be registered on each side, which adds a delay of one clock cycle by each register stage. The quality of a PLL is also highly dependent on the underlying technology and physical implementation. For all three mentioned types of technology dependent elements, the main requirement of meeting the old timely behaviour stays the same.

Unfortunately, it is not always easy to access the information about used components, their configuration, and thus their timing behaviour. Generally speaking, five methods can be identified to get the necessary details about components of a third party project.

Data sheets of the configuration of the used components might be present. The information about the timing details can be directly accessed and used for the own design. This is the preferred method to get the respective information, but it is not always applicable. Information about the exact timing of internal memory structures usually is not found in the data sheets of any device. The design internal timing is not relevant for end users. Thus, chances are very low to have documented information about the internal timing, if a project is not explicitly designed for exchangeable modules.

Source code of the project is more likely to be available than the specifically documented information about the internal timing. If the source code is available, it can be used as well to get the relevant information about the timing, though it is not always easy. The device specific elements are often created using some kind of generation tool, which creates more or less readable code, usually with a lot of parameters describing the actual behaviour. If the parameters have speaking names, the needed information might be extracted very quickly, but care has to be taken as some important information or cross dependencies might be missed. Using this method can lead to a very quick solution, but it is highly recommended to counter check the result with a different method, for example by simulation. The combination of these two methods, source code inspection and usage of simulation, has been used for this diploma thesis.

Simulation can provide reliable information about the timing behaviour of components, but the simulation test bench and other modules for simulation purposes as well as appropriate test cases have to be available. Even when they are available, it might take some time to find the corresponding test cases and signals to reveal the detailed behaviour of the investigated modules.

Developer contact might finally give access to some missing information. A contact should be available in most of the cases, but there is no guarantee that support is offered, and if it is offered, responses could take some time.

Hardware observation is the least suggested possibility and should only be used if all other methods fail. The hardware design has to be programmed into the FPGA and some means for signal observation inside the FPGA has to be applied. As it might be necessary to make a lot of changes at once to be able to get meaningful results, various complex error sources might be added, which makes it very difficult to observe the correct functionality of the replaced modules.

5.2.2 Board Specific Alterations

The necessity for the second group of alterations is due to the dependence on the available hardware components and their respective interfaces. Except for the third item, the transition to a PCI Express interface, which is the main part of this diploma thesis, the

other two mentioned items were not expected to raise major problems. However, in the course of the practical realisation at least the connection of the Ethernet PHY was not straight forward.

5.2.2.1 Clocking Scheme

The original clocking scheme had to be altered because of the specific requirements for the PCI Express MegaCore clocking. In the original implementation, two external clocks were fed into the design, a so called user clock, which sourced the Clock Synchronization Cell (CSC), and the PCI clock, which sourced the PCI core and the AHB components. The user clock was operated at a frequency of 25 MHz, while the PCI clock was driven at a frequency of 33 MHz. In the new design, the PCI Express interface offers no dedicated clock line. The receive clock can be derived from the received data, the transmit clock has to be sourced at the transmitter. On the used Altera hardware platform the Stratix II GX FPGA device has a high-speed transceiver module included which is used as the electrical sub layer for the PCI Express PHY. This receiver requests a 100 MHz input clock, which is then converted to a 125 MHz output clock, the frequency at which the Altera PCI Express MegaCore Streaming Interface operates. Therefore, it is called the application clock.

In the original design, the PCI clock was also used for debouncing of the reset buttons as well as for delaying the Ethernet PHY reset. In the new design, the 100 MHz input clock must not drive any other logic except for the high-speed transceiver, and the derived application clock is only available after the reset is deasserted. Therefore, the clocking scheme for debouncing and Ethernet PHY reset had to be redesigned to be sourced by the user clock.

5.2.2.2 Ethernet Physical Layer Device Connection

The communication between an Ethernet PHY and Ethernet MAC device is standardised in the IEEE 802.3 standard [5] clauses 6, 22 and 35. For data transmission the Media Independent Interface (MII) and Gigabit Media Independent Interface (GMII) protocols are used, while for station management functions the MDIO interface is used. Using the MDIO, registers in the PHY can be configured or read out, such as PHY ID, link speed, auto-negotiation, etc. For proper operation of the tristate bus a pull-up resistor has to be inserted. The data sheet [29] of the used Marvell® PHY requires a pull-up resistor in the range from 1.5 kΩ to 10 kΩ to be inserted to pull the bus high in idle state, when both drivers are in high impedance state. On the Altera evaluation board, a 4.5 kΩ pull-up resistor is mounted, which is compliant to the requirements of the PHY.

To verify the functionality of the MDIO interface, the new design was implemented in the FPGA device in an early development phase. After the register read and write operation design was completed as described in section 5.3.5, the PHY was connected to the FPGA.

Unfortunately, the PHY did not respond to any register read requests. Observation of the signal levels with Altera's Signal Tap tool revealed, that the MDIO interface bus turnaround failed. After the station management of the MAC started a read operation and switched to high-impedance state, the MDIO signal line stayed high, although the PHY should have taken over control and responded. To solve the problem, first all timing requirements and electrical specifications defined in IEEE 802.3 clause 22 and the datasheet of the PHY [29] were checked on the schematic level of the board layout [30], but no violations were found.

As no obvious design error could be identified, a different approach had to be taken. Together with the evaluation board, Altera delivers a couple of reference designs. One of those reference design projects contains the initialisation of the PHY. In this reference design, there were three different versions of the MDIO read operation control. The first version switched the station management to high-impedance state when the PHY should respond, as it is required by the standard. The version finally used in the reference implementation drives the bus constantly low, even when it should be in high-impedance state during PHY response. Copying this behaviour to the design of this diploma thesis solved the problem, and registers of the PHY could be read.

The reason for this behaviour might be a wrong pull-up resistor. The influence of a too large or too small value of the resistor is investigated to derive a possible failure scenario. Figure 5.2 shows the MDIO bus line with the two drivers and the pull-up resistor.

When all drivers are in high-impedance state, the pull-up resistor drives the bus line high. A receiver thus can not distinguish between an actively driven high state and a high-impedance state. A too high value, or even a missing pull-up resistor would result in a too low voltage level of the bus line in the high-impedance state, making it impossible for the PHY to detect the high impedance state of the station management in the bus turnaround phase. In that case, the bus would have to be driven active high by the station management to fake the beginning of the turnover phase. As the bus line has to be driven low, a too big pull-up resistor can not be the cause for the problem.

A too small resistor could result in a permanent high state of the signal line. If the PHY drives zero to the bus line, it has to be able to draw enough current to make the voltage drop over the pull-up resistor large enough for the bus line to switch to low voltage state.

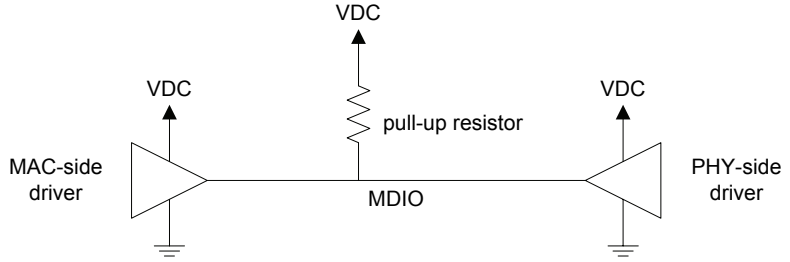


Figure 5.2: Pull-up resistor and drivers of the MDIO bus line

If the current-limitation of the driver is too low, not enough current can be drawn and thus the bus line stays high. When the opposite side also drives the bus active low, the current can be split to both drivers, which is enough to pull the line to the digital low level. If now the PHY drives the line high, the driver must be strong enough to pull the line up again (supported by the small pull-up resistor) and override the low driving opposite side. This basically results in a short circuit for the time the signal line is driven high, but the current-limitation saves the parts from getting destroyed.

Concluding from the observed behaviour, it seems likely that some kind of leakage current bypasses the mounted 4.7 kΩ pull-up resistor and thus hinders the MDIO signal line to be driven low by the PHY. The existence of the different file versions of the reference project might suggest that the developers of the evaluation board also had some problems with the MDIO interface.

5.2.3 Configuration of the PCI Express Endpoint

The available PCI Express Endpoint design by Altera has several configuration options. The most important options are described here and the chosen configurations are reasoned.

The address space is configured for the use of 32-bit addresses only. The application requires only 13 bits (8 kBytes) of address space, therefore no 64-bit addresses are needed. The memory mapped region is configured as non-prefetchable memory, because the CSC uses a FIFO structure for the stored timestamps and packet IDs where reading is destructive. A side effect of the use of non-prefetchable memory is the possibility to use 32-bit addresses, as only non-prefetchable memory is allowed to not support 64-bit addresses.

It further is configured as a single lane connection (x1 PCIe), which supports up to 250 MByte per second of raw data. This is fairly enough for the theoretical maximum of Gigabit Ethernet's 125 MByte per second. Support of ECRC is disabled, as the encapsulated Ethernet packets have their own FCS for data integrity checks.

The maximum payload length is set to 256 byte per TLP. This size allows high protocol efficiency, while keeping the maximum blocking times and required buffer sizes reasonably low. The supported number of tags is set to the standard of 32, which allows for 32 concurrently outstanding completions. Under normal operation conditions, this number should be fairly high enough. Further, as no virtual channels and traffic classes were supported in the original PCI system, there is no need for virtual channels in the PCI Express either, therefore the number of virtual channels is set to the minimum of one.

5.2.4 Reset Circuit for the PCI Express Endpoint

The several reset signals and the associated reset logic are depicted in figure 5.3. The reset signal names ending with a number sign are low active signals. The five reset sources can be reduced to three different types, the power-on reset signal on the PCI Express connector

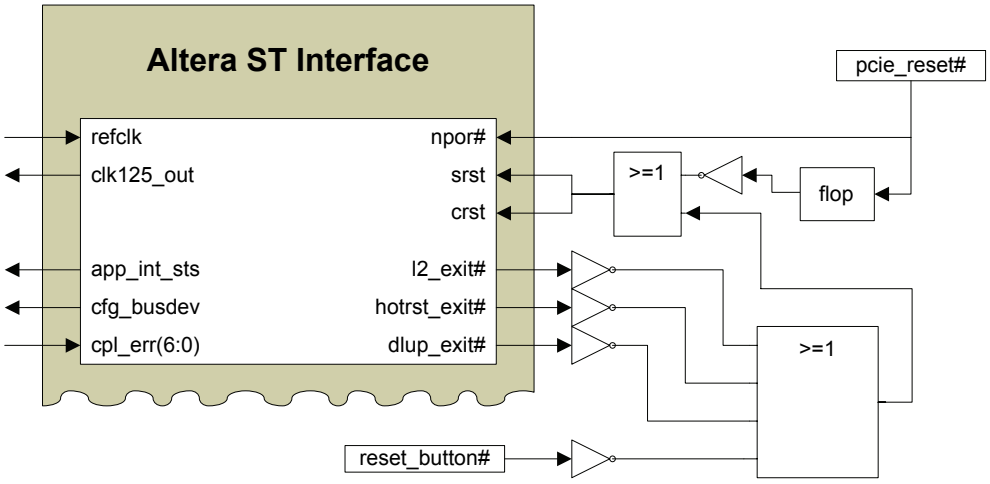


Figure 5.3: Reset circuitry for the PCI Express Endpoint

(`pcie_reset#`), the user push-button reset (`reset_button`) and the three hot reset signals triggered by PCI Express messages (`l2_exit#, hotrst_exit#` and `dlup_exit#`).

The `pcie_reset#` signal is the power-on reset of the host PC. When this signal is active, the main reset signal of the Altera interface, the negative logic power-on reset (`npor#`) is also set to zero (activated). This reset is also called fundamental reset and resets all register settings, including even the sticky registers which are not affected by a hot reset. Additionally, the datapath (`srst`) and configuration (`crst`) resets are asserted. To ensure that the root complex is stable and ready for link training, those resets are held active (`flop`) for some microseconds.

On any activity of the three hot reset signals or the activation of the user push-button the datapath and configuration resets are asserted. Such a hot reset is applied when the software requires a hot reset (`hotrst_exit#`), when the PCI Express nodes leaves the low power state, (`l2_exit#`) or when the data link layer looses connection (`dlup_exit#`).

5.3 Architectural Structure of the Bus-Switching Unit

The development of the bus-switching unit was the main part of this diploma thesis. It is the core unit for the translation and arbitration of the AHB interfaces and the Altera PCI Express Streaming Interface, as already discussed in section 4.2 and in chapter 3. The block diagram of the architecture is depicted in figure 5.4. First, the functions of the individual modules are described. Reasons are given, how and why functional requirements were mapped to the individual modules. Furthermore, for important decisions the influence of possible variations are discussed.

Beside the AHB interfaces and the PCI Express interface, a header decode unit, an address decoder and both, input and output FIFOs are present. Referring to the discussion about

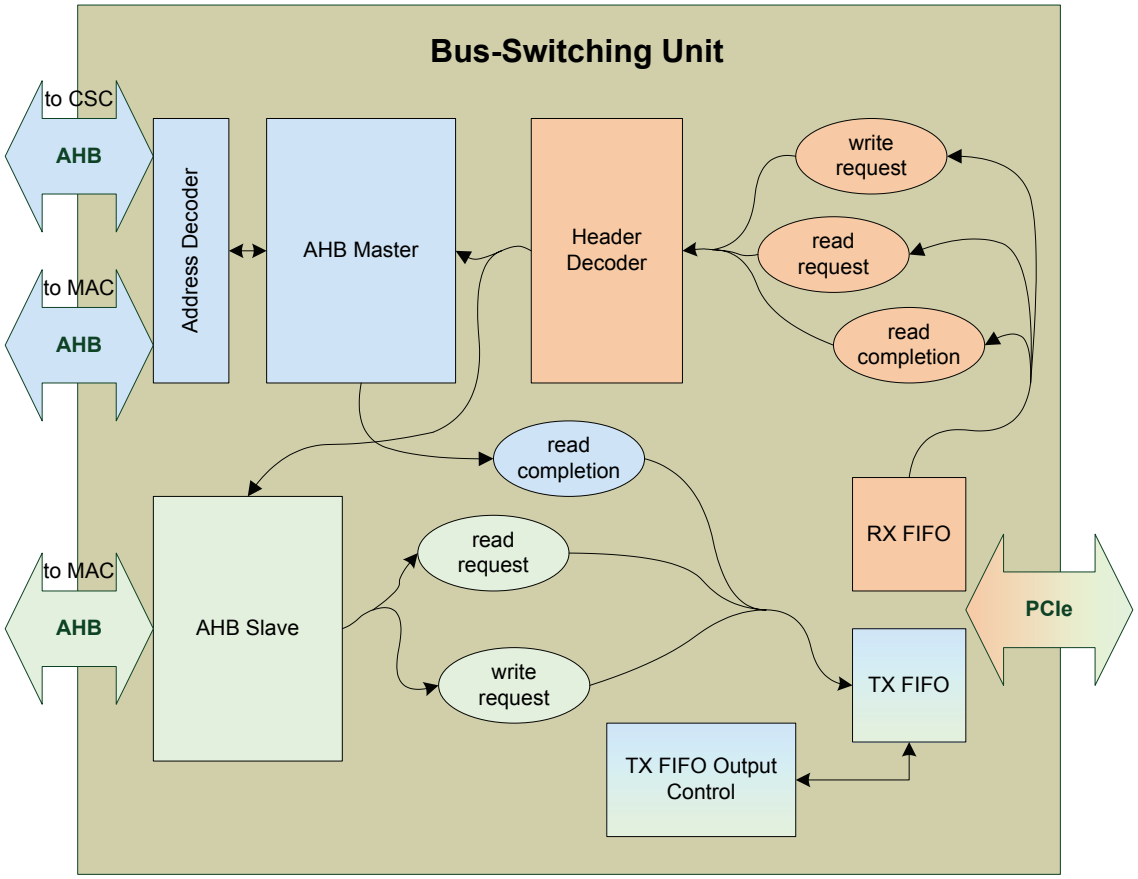


Figure 5.4: Architecture of the bus-switching unit

the possible solutions in chapter 3, the dedicated bus arbiter unit and the packet translator are missing. Those functions are integrated in the header decoder and the AHB interfaces. For the protocol translator, the argumentation of section 3.2.1 applies. For the bus arbiter, it has to be explained that a mutual exclusion scheme for the complete AHB interface state machines is used. Therefore, the simple case of mutual exclusion, as described in section 3.3.2, eventuates. This simple mutual exclusion scheme of the state machines was not the first choice during the development of the bus-switching unit. For an explanation about the history of this decision refer to section 5.4.

Numerous considerations lead to the conclusion, that a buffer for incoming PCI Express TLPs is needed. The necessity for rate-matching the faster PCI Express interface to the slower AHB interface requires to either stall the PCI Express interface, or to buffer the incoming TLPs, as described in section 3.2.4. Since the PCI Express interface, as described in section 2.1.4, is only required to react within three clock cycles on changes of the ready signal, the incoming data of at least three clock cycles has to be buffered. The use of the mutual exclusion scheme for the state machines also require the buffering of incoming TLPs, when the state machine currently is blocked. Since a buffer has to be implemented anyway, it is as well used as a shared memory for packet-switching, as described in section

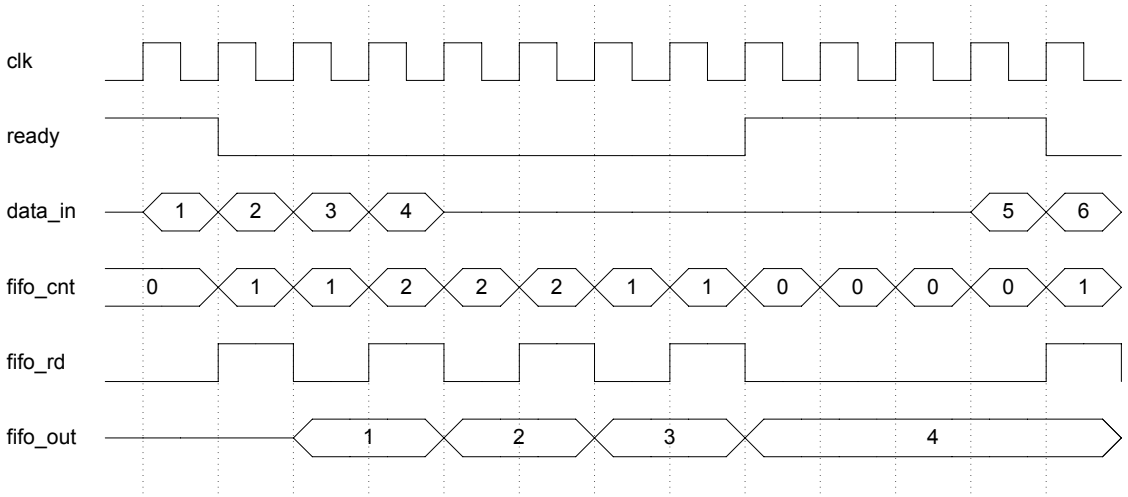


Figure 5.5: RX-FIFO timing with simple ready signal control

3.2.2. Therefore, the controlling read signal for the Receive FIFO (RX-FIFO) becomes a shared resource as well, as all modules with RX-FIFO data access have to be able to take over control of the read signal. This additional shared resource must also be considered in the bus arbitration.

The incoming data to be buffered does not have to be reordered, or read in advance. It has to be stored, 64 bits (the size of a quadword) per clock cycle, and read in the same order as it was written to the buffer. Therefore, the most reasonable architecture for the buffer is a FIFO structure. The size of the FIFO can be varied. The minimum size is three times 64 bits, but this minimum size already needs some small additional logic to prevent buffer overflow, and thus data loss.

5.3.1 Receive FIFO Memory

The simplest version can be realised with a FIFO with the size of four quadwords. Figure 5.5 shows an example timing diagram for this simple version. The simple logic for the ready signal of the PCI Express interface has to deassert the ready signal on the reception of each data block, and reassert it when the buffer is empty. The deassertion of the ready signal has to be done immediately on reception of data, because it is not known in advance if the data of the FIFO will be read in the next clock cycle. If it is not read, the data stays in the buffer. The ready signal is required to stay deasserted until the data of the FIFO is read again, since the three remaining stages of the buffer are needed to buffer possibly incoming data until the PCI Express interface reacts on the deassertion of the ready signal. The ready signal can only be reasserted, when the FIFO is empty. From then on it takes three clock cycles until the PCI Express presents the next data. Thus, in order not to loose any data, many performance deteriorating wait-states have to be inserted. With this simple logic, the use of a six- or seven-staged FIFO would greatly

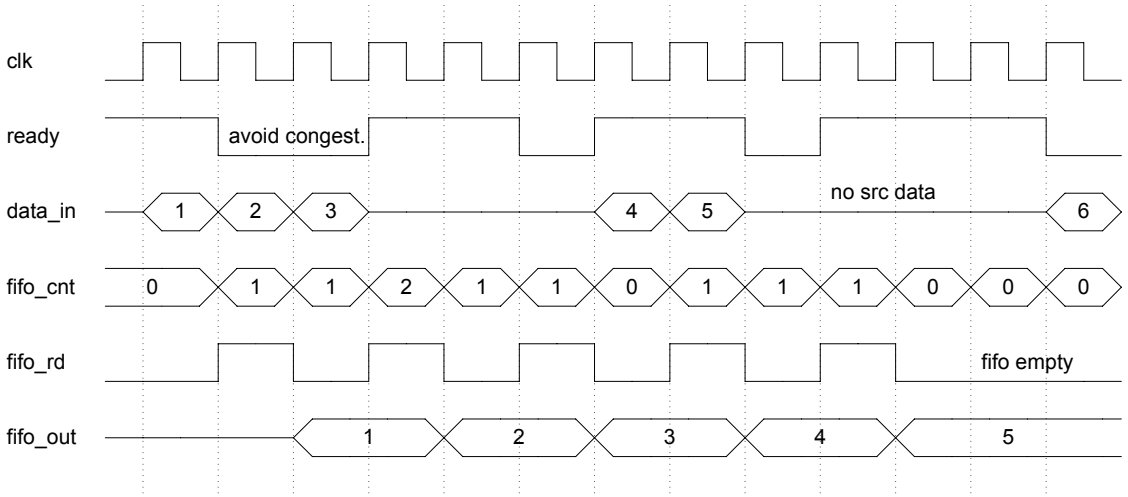


Figure 5.6: RX-FIFO timing with intelligent ready signal control

increase the performance, because the ready signal would not have to be deasserted after the reception of only a single TLP.

With a little more intelligent control of the PCI Express interface ready signal, the minimum of a three staged FIFO buffer suffices to prevent data loss without any deterioration of the system's performance. Figure 5.6 shows the corresponding timing diagram. The idea behind the ready signal control is an active surveillance of the maximum possible FIFO fill for each clock cycle. Therefore, the state of the ready signal at the current, and the two preceding clock cycles is counted and added to the current value of the FIFO fill counter. An asserted ready signal is counted as one, a deasserted ready signal is counted as zero. If this number is smaller than three (the FIFO depth), the ready signal for the next clock cycle is asserted, otherwise it is deasserted. This mechanism ensures, that at any time the FIFO has enough remaining space to store all possible incoming data from outstanding asserted ready signals. It also prevents the ready signal to be high for more than three consecutive clock cycles, which is required to maintain the minimum size of three FIFO stages.

The RX-FIFO can certainly be made larger than the minimum size, but no increase in performance can be achieved. A larger FIFO can store more incoming TLPs when the processing state machine is blocked, but it is not necessary to do so. The PCI Express interface unit has a receive buffer for itself, which is needed to maintain full bandwidth because of the PCI Express credits update loop delay. A larger RX-FIFO in the bus-switching unit would only come into account, when first incoming TLPs were not accepted, until the PCI Express internal FIFO was full, and then all buffered TLPs were processed faster than new data could be delivered over the PCI Express link. When the system operation is considered it becomes clear, that under normal operation conditions this situation does not occur.

The device driver can only send two different types of TLPs: completions when they are requested, and memory read or write requests for register operations. Since the completion packets, which are involved in Ethernet traffic, can only be sent when they are requested, they can't come faster than the requests can be sent. A new request can not be sent, until the completion of the last request arrives and is completely processed. Therefore, with only Ethernet traffic, the system stays in a self-regulated equilibrium and is only required to buffer the data of one PCI Express TLPs.

The other type of TLPs, register read and write requests, might lead to full buffers when the device driver floods the network interface card with requests. As all register read requests require a completion to be sent back, they definitely can't be processed faster than they can arrive. Even if the processing speed was high enough, the internal transmit buffer of the PCI Express interface, which is of the same size or smaller as the receive buffer, would be filled up. Because PCI Express has symmetric transmission speeds, outgoing TLPs can't be sent faster than incoming TLPs can arrive. Thus, read requests can not be processed faster than they can arrive, and therefore the FIFO size of the bus-switching unit is not relevant. Register write requests are different. They can be processed faster than they can arrive, since no completion has to be sent back. Therefore, it is possible to fill the PCI Express interface internal buffers with write requests when the bus-switching unit is currently not accepting incoming TLPs, and then process them faster than new requests can be sent. Considering the fact that under normal operation conditions only a few consecutive register write requests are sent, the theoretical possibility of the influence of the RX-FIFO in the bus-switching unit is not performance deteriorating, because the performance influencing condition does not happen under normal operation conditions. The only case when several subsequent write requests are likely to occur is the initialisation phase of the NIC. Since the initialisation is done only once, and is most likely outlasted by the auto-negotiation procedure of the Ethernet PHY, also this regular use case can not be counted as performance deteriorating.

5.3.2 Transmit FIFO Memory

The reason for the Transmit FIFO (TX-FIFO) in the bus-switching unit is a completely different one than for the RX-FIFO. The only common reason is the necessity for rate-matching. For the TX-FIFO, not the slow reaction time of the faster transmitting unit requires the data to be buffered, but the missing possibility to stall the faster receiving unit. The PCI Express interface requires a PCI Express TLP to be transmitted at once, without inserting any wait states once the transmission has started. The data sink, which is the the PCI Express interface, works with twice the data rate of the data source, the AHB interface. Therefore, the TLP has to be buffered until the complete TLP is available, before it can be presented to the PCI Express interface.

The size of the TX-FIFO can easily be derived from this requirement. It has to be able to hold a complete TLP, which includes the size of the TLP header, and the size of the

data payload, which is exactly one cell size, as configured in the corresponding register of the Ethernet MAC. Since the cell size can be reconfigured dynamically, but the size of the TX-FIFO can not, there must be an absolute limit for the cell size, and the TX-FIFO must be able to accommodate a TLP of the corresponding maximum size.

Beside lowering the maximum cell size, there are two different methods to reduce the depth of the TX-FIFO. If the AHB master guarantees never to insert a wait state, thus guaranteeing only to start a burst transfer when the complete burst of the size of one cell can be transferred, then the TX-FIFO depth can be reduced to the size of a TLP header plus half the size of one cell. In this case, the transmission of the TLP must begin as soon as half of the cell size of data is written to the FIFO. From then on, the PCI Express interface side draws 64 bits per clock cycle out of the FIFO, while the AHB side puts 32 bits per clock cycle of new data into the FIFO. Because the transmission begins exactly when half of the cell has arrived, the FIFO gets empty only when drawing the last quadword out of the cell.

The requirement for no wait states of the AHB master goes beyond the specification of AHB. Half of the FIFO size of memory can be saved, but at the prize of reduced compatibility and reusability of the bus-switching unit. A better solution to completely get rid of the TX-FIFO would be a change of the data width of the AHB interface. With a 64 bit wide AHB data bus, there is no need for rate-matching, and thus no need for a TX-FIFO. Unfortunately, the existing interface specification is a legacy requirement, and therefore unchangeable.

On the other hand, increasing the buffer size of the TX-FIFO does not directly effect the system performance. There is a corner case, though, where a larger buffer does affect the performance. When the PCI Express bus is congested, the internal buffer of the PCI Express interface is full, the TX-FIFO of the bus-switching unit is full and all frame buffers of the Ethernet MAC are full as well, then a larger TX-FIFO would prevent incoming Ethernet frames from being discarded. This situation only happens when the device driver is not able to handle incoming data streams fast enough, which did occur in test runs. However, the investigation of the reasons for this situation is out of scope of this diploma thesis, as it concerns the device driver and not the hardware modules. Besides, it is not the job of the bus-switching unit to buffer incoming Ethernet frames. For that case, the frame buffer of the MAC would have to be increased, not the TX-FIFO of the bus-switching unit.

5.3.3 Header Decoder

The unit in figure 5.3 labeled “Header Decoder” can be identified as the packet-switching module, as described in section 3.2.2. Additionally, it has a part of the bus arbitration function integrated. It is named header decoder because it has to read and analyse the TLP header in order to route the packet to its intended destination.

The functional rationale for the packet-switching is the cut-through technique with shared memory switching. The header decoder is implemented as a Moore machine, i. e. an FSM with the output signals being only dependent on the current state of the state machine, not on the value of the input variables. The implementation as a Moore machine increases the response time, because no immediate response on the change of input signals is allowed. This reduces the performance of the whole system, but the understandability of the control sequence is greatly increased, which again reduces the risk of bugs during development and hence increases the stability of the system.

Packet-Switching

In the following paragraph, the sequence of operation is described. The PCI Express interface presents the TLP in sets of eight bytes per clock cycle. When a PCI Express TLP arrives, the first set of eight bytes of data is written into the RX-FIFO and can be read by the header decoder module in the next clock cycle. The header of the TLP is twelve bytes long, as described in detail in section 2.1.3.2. The first arriving eight bytes on the PCI Express interface thus are the first part of the TLP header. As depicted in the figures 2.10 and 2.11, the format and type fields are included in this first part of the header. Those two fields are used to determine the destination of the data transmitted in the TLP.

Memory read and write requests belong to transactions initiated by the device driver. The device driver is only allowed to initiate register read and write operations of the length of exactly 32 bits of data. Therefore, when a memory request TLP is received, the length field is checked to have exactly the value of one, which corresponds to one DW of data. In the case of a read request, the requester ID and the tag information, as well as the attribute and the traffic class field are stored in a register, as this information has to be included in the completion TLP. The content of the attribute and the traffic class fields are ignored, as different traffic classes are not supported by the NIC. If the data is indicated as poisoned, the TLP is dropped. If an error is detected in a read request, an error message is sent back, informing the requester that the sent request is unsupported. If an error is detected in a write request, the TLP is dropped and no further action is taken. If all checks are passed, the request is routed to the AHB master module. This is done by asserting a control signal, allowing the module's state machine to start its operation. For an overview of the information flow of the TLPs inside of the bus-switching unit, refer to figures 3.2 and 3.4.

Completion packets belong to DMA operations and are the response to previously issued memory read requests. The memory read requests are issued by the AHB slave, and therefore the completion packets are routed back to the AHB slave module by asserting the corresponding control signal. The completion packets contain the data of Ethernet frames, which are bound to be transmitted over the Ethernet link. Upon reception, the header decoder module checks a completion TLP for a successful completion status, for correct traffic class and attribute fields and for the correct size, which is controlled by a

configurable register in the Ethernet MAC unit. In the case of an error, an error message is sent back, informing the requester about the unexpected completion. Any other TLPs except for the described ones are discarded upon reception.

Bus Arbitration

Concerning the bus arbitration, one part is integrated in the idle state of the state machine of the header decoder, whereas the second part is integrated in the idle state of the AHB slave. As already mentioned, the simple mutual exclusion version is used, which allows the integration of the very short code for arbitration in the respective modules. A description and some thoughts about the mutual exclusion scheme for bus arbitration were presented in section 3.3.2.

The reason for integrating the bus arbitration function in the header decoder module is its leading position in the processing sequence. Only the header decoder module can start the processing of a received TLP, which is stored in the RX-FIFO. To ensure that only one state machine is allowed to run at a time, the header decoder module monitors the operation of all other state machines and only starts the processing of a new TLP when all state machines are in idle state. When a TLP is present in the RX-FIFO, the header decoder module starts the processing by decoding the header of the TLP and asserting the respective control signal, granting the addressed module to access the RX-FIFO and further process the TLP. The header decoder then waits until it receives a response signal asserted by the addressed module. This control signal tells the header decode state machine, that the respective module has finished its operation, is back in idle state and does not access any shared resource any more. Only after this response signal is received, the header decoder itself changes back to idle state, and thus is ready to start a new operation. By waiting on the end of the operation, it is assured that two modules can never work simultaneously. This is important, as all bus modules share the TX-FIFO as well as the RX-FIFO resource.

The header decoding module thus works as a bus arbiter, passing control to the respective bus module using a dedicated start control signal. Actually, there are three different start signals. One is for the AHB slave module upon reception of a completion, and two are for the AHB master module, one for read and one for write. As the header decoder has to analyse the format and type field of the TLP to be able to switch it to the correct module, it already has knowledge about whether it is a read or write request. Therefore, the respective signal is asserted, both telling the AHB master to start the respective operation.

For the prioritisation of the modules, the following considerations were taken into account. Remember that the prioritisation mechanism is only relevant when both independent sides want to start a transfer at the same time, the device driver by sending a read or write request to the PCI Express interface of the bus-switching unit, and the Ethernet MAC by starting an AHB transfer which involves the AHB slave in the bus-switching module. For

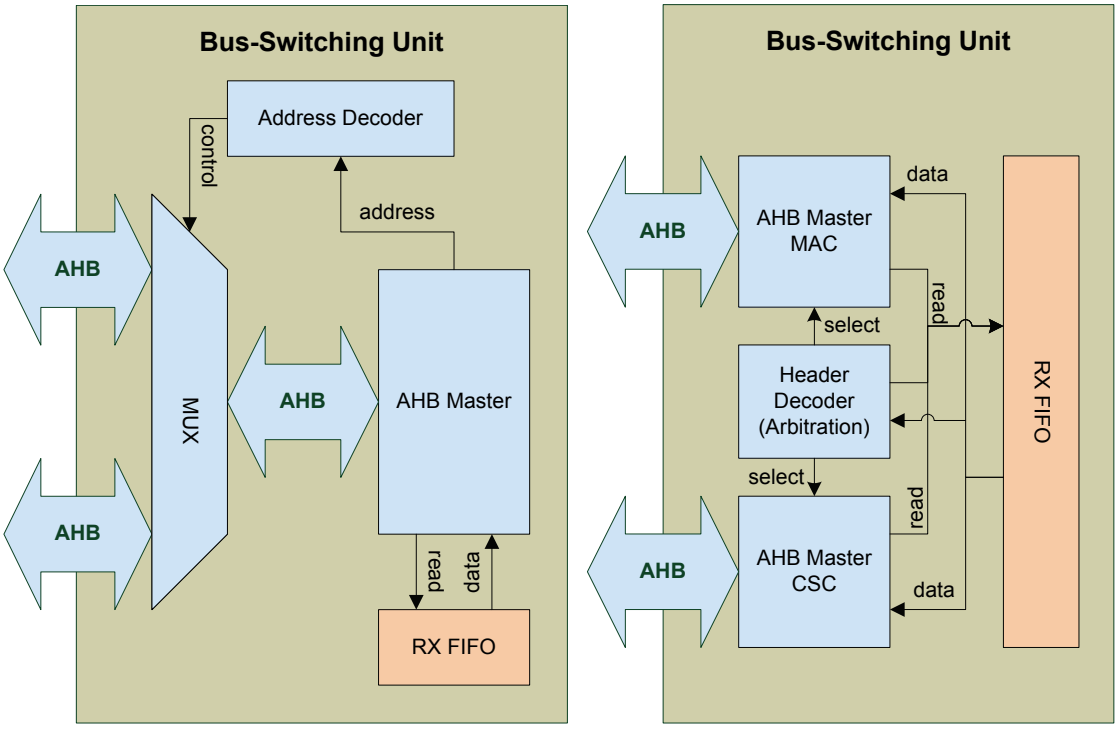
the decision on which transfer to prefer upon concurrent starting conditions, the different nature of the transfers has to be considered. Register operations are restricted to the length of only one DW of data per transfer. Therefore, their TLPs are short and the operations are processed quickly. For that reason, register operations are prioritised over the longer DMA transfers.

The prioritisation is easy to implement. As already mentioned, the header decoder module can only start the processing of a new incoming TLP request, when itself is in idle state. Because it waits for all controlled state machines to return to idle before itself switches into idle mode, no conflict can occur with subsequent incoming requests. The only point of conflict can be the AHB slave, which is controlled by the Ethernet MAC unit. Therefore, before the header decoder module starts the processing of a TLP, it has to check whether a DMA operation currently is in progress. This is done by observing the state machine of the AHB slave interface in the bus-switching unit, which is responsible for the DMA transfers. If the header decoder module finds the AHB slave to be in idle state, the processing of an incoming TLP can start. If the AHB slave is not in idle state, the header decoding module has to wait until the slave module has finished its operation. Only after the header decoder observes, that the slave module has changed back to idle state, which indicates that the previous operation has finished, the processing of the received TLP is started. Because the prioritisation of the incoming TLPs, it suffices to ensure that the AHB slave state machine is not currently working. If the AHB slave receives a transfer at the same time, requiring the state machine to start its operation, the AHB slave is of lower priority and has to wait for the incoming TLP to be processed first. Therefore, the AHB slave has to observe the starting condition of the header decoder as well, but the header decoder does not have to care about a concurrent starting condition of the AHB slave.

Concluding, the packet-switching is done by granting the addressed bus module access to the shared RX-FIFO memory, which holds the incoming data. After the header decoder has interpreted the first quadword of data, the respective bus module takes over control, and continues to process the incoming data. Therefore, the bus arbitration behaviour corresponds to cut-through, shared memory switching. The realisation as a Moore machine results in a lower overall performance. The control output signals can only be asserted after the first part of the header is decoded. Although the TLP data is already present, the further processing module first has to sense the asserted control signal, until it can process the data. Therefore, the processing is delayed, but the sequence is much easier to understand, to develop, and to maintain.

5.3.4 Address Decoder

The address decoder as such is a very simple module. Since there are only two slaves present, a very simple address decoding can be applied. All addresses below a certain value belong to the Ethernet MAC unit, all larger addresses belong to the CSC. For the



(a) Standard architecture **(b)** Architecture after logic duplication

Figure 5.7: Address decoding with one and two AHB masters

sake of simplicity, the address boundary is 0x1000 (hexadecimal notation), which allows address decoding based on the obervation of a single bit.

However, the real structure of the AHB interfaces and the address decoder is different. The different architecture was necessary because of timing problems during the implementation phase. Figure 5.7 shows the difference between the standard architecture and the actually implemented architecture. Although, from a functional point of view the structure of a single master, an address decoder and two slaves is maintained, the logical structure is changed. Instead of a single master with a multiplexer, two identical AHB masters are used, each of them having a direct point to point connection to only one of the slaves. By performing this logic duplication, a register retiming is achieved. The large multiplexer for the AHB signals from figure 5.7a became obsolete, because there are no shared AHB signal lines any more. Only the read signal for the RX-FIFO is shared. But, in contrast to the AHB signals, this is a single data line. Because of the mutual exclusion scheme of the state machines, no real multiplexer is needed, it suffices to or-connect the two sources. By duplicating the AHB master, the or-gating was moved from the AHB side of the master to the RX-FIFO side of the master. Additionally, only a single line instead of the whole AHB signals has to be gated. Therefore, the registers have been retimed and the routing became much simpler.

Due to the duplication of the AHB masters, the need for address decoding is moved to the functional partition of the bus arbiter, as a different addressed slave also means a different addressed master. Thus, the address decoding has to be done in the header decoder module. As the address is not contained in the first quadword of a TLP header, the second quadword also has to be read. Only after the address is obtained, the corresponding control signal can be asserted. From then on, the operation is equal to the standard version with one master and a separate address decoder, as described in section 5.3.3.

However, in all previous considerations the two AHB masters were treated as if they were realised in the standard way, as one master with a following address decoder. For the sake of understandability and consistency, this style will be kept in all following descriptions except for the rare cases, where the special realisation as two distinct masters have to be taken into account.

5.3.5 AHB Master – Register Operations

Although there are actually two AHB masters, as was described in the previous section, the operation of only one master is described, because the two master modules are exact duplicates, and therefore work in the same way.

The AHB master module is responsible for processing the register read and write operations. The principle of operation was already outlined in section 3.1.1. The AHB master module does not support all optional features specified in the AHB specification [6]. It rather is implemented as an AHB lite [18] master, with support for non-sequential transfers only. An AHB lite master is a simplified version of a full AHB master, which operates only in a single master environment, and therefore, no AHB bus arbiter is required, nor do split and retry slave responses have to be supported. The support of non-sequential transfers only is a further restriction to a full AHB master, but it is sufficient, considering that the AHB master interface solely is involved in the register read and write operations, which are defined to be exactly 32 bits per operation. Therefore, in any case all data can be transmitted in a single, non-sequential transfer and no support of burst transfers is required.

The AHB master interface is implemented as an FSM. It has to read the PCI Express TLP data from the RX-FIFO, which is a shared resource. Therefore, it is not allowed to start its operation independently. As already discussed, the header decoder module acts as an arbiter as well as a packet-switcher. When the header decoder identifies a TLP for register operation, the controlling start signal is asserted. When in idle state, the AHB master does nothing more than waiting for the assertion of either one of the start signals, the start signal for a read or write operation. As soon as the start signal is received, the AHB master can start its operation. For the reasons presented in section 3.2.1, the protocol translation is integrated in the AHB master state machine. Therefore, the master has to read the data of the TLP and translate it to an AHB transfer.

Read Request

If a read request TLP is received by the bus-switching unit, the only information the AHB master needs to extract from the TLP is the address of the read request. Because of the special realisation as two distinct masters, the address decoding is already done by the header decoder module. Therefore, at the time the AHB master state machine starts its operation, the part of the TLP header containing the address has already been read from the RX-FIFO and is no longer available. However, the address has to be presented to the AHB slave. The header decoder thus stored the address in a register, which now can be accessed by the AHB master module. No further information of the PCI Express TLP is needed, hence no data from the RX-FIFO needs to be read for a register read request.

To initiate an AHB read transfer, the master signals the start of a non-sequential transfer of the fixed size of 32 bits to the slave and waits for the ready signal of the slave to be asserted, which indicates that the AHB read data lines hold valid data. Because of the mutual exclusion scheme of the state machines, it is guaranteed that currently no other module wants to access the shared TX-FIFO resource. Therefore, the AHB master immediately can start to write the first eight bytes of the TLP completion header into the TX-FIFO. The following eight bytes of data expected by the PCI Express interface depend on the address of the register read operation. As described in 2.1.4, it has to be distinguished between quadword aligned and non-quadword aligned addresses. In any case, the lower four bytes, bytes 9 to 12, contain the four remaining bytes of the header. This remaining part of the header contains the completion specific data which had to be extracted from the requesting PCI Express TLP. As the complete TLP was only read by the header decoder, this information, as well as the address, had to be stored in separate registers, which can now be accessed by the master module to build the proper completion.

The upper four bytes, bytes 13 to 16, depend on the address of the read request. In the case of a quadword aligned address, they can be of arbitrary value, in the case of a non quadword aligned address they are expected to hold the completion data, which is the data provided by the AHB slave. For non quadword aligned addresses, the TLP is complete and can be sent, whereas quadword aligned requests require another eight bytes of data to be presented to the PCI Express interface. Bytes 17 to 20 are expected to hold the data, while the remaining four bytes, bytes 21 to 24, can be of arbitrary value. Therefore, the assembly of the completion packet for a quadword aligned addressed read request takes three clock cycles, whereas the assembly of the completion packet for a non quadword aligned addressed read request takes only two clock cycles.

When all data is written to the TX-FIFO, a control signal is asserted which causes the TX-FIFO control module to start the transmission of the TLP to the PCI Express interface. Only after the TX-FIFO control module confirmed the start of the transmission, the AHB master module asserts the response signal to the header decoder module, indicating that the transfer is completed, and changes back to idle state.

It might be argued that the assembly of the completion packet could already start before the actual data of the slave is present, as the first eight bytes of data only contain header

information which is independent of the read data. This is true, and the benefit would be a speed gain of one clock cycle per register read request, thus shortening the time where no Ethernet data can be transferred, because the mutual exclusion scheme only allows the operation of one state machine at a time. However, it would lead to more complex states in the master state machine. The goal of this diploma thesis is the implementation of a stable test setup for clock synchronization, where high performance Ethernet throughput is not required. Thus, the simpler but less performant design was chosen, as it simplifies the development of a stable hardware.

If, at any time during the AHB transfer an error or an unexpected behaviour of the AHB slave occurs, the dedicated error messaging signals of the PCI Express interface are asserted according to the occurred error. The PCI Express interface then uses the error messaging means of the PCI Express protocol to inform the device driver about the occurred error.

Write Request

In contrast to the read request, the write request is a posted PCI Express transaction which requires no completion packet. Therefore, there is also no possibility for error reporting.

When a PCI Express write request is received by the header decoder module, the corresponding start signal for a write request is asserted. The AHB state machine then already has to distinguish between a quadword aligned and a non quadword aligned address. In the case of a non quadword aligned address, the data was already contained in the upper four bytes of the TLP portion that had to be read by the header decoder to extract the address. As no further data had to be read from the RX-FIFO since then, the data to write is still accessible at the registered output of the FIFO. Therefore, the AHB master can immediately start a write transfer. The address is also still accessible at the output of the RX-FIFO and no additional registers are needed to store TLP data. The master simply waits until the slave responses with an asserted ready signal, indicating that it is ready to receive the data, and presents the data contained in the write request TLP. As soon as the slave indicates the successful completion of the write operation by having the ready signal asserted again, the AHB master asserts the response signal to the header decoder module, indicating the completion of the transfer, and changes back to idle state.

In the case of a quadword aligned address, no data is included in the address containing quadword. Only the subsequent quadword holds the data. Therefore, the data has to be requested from the RX-FIFO by asserting the read signal. As the data is available only one clock cycle after the asserted read signal was detected by the RX-FIFO, the previous output data, containing the address information, can still be accessed at the output of the RX-FIFO. Therefore, the AHB transfer is started and handled in the same way as with a non quadword aligned address, except for the fact the data at the output of the RX-FIFO changes between the address phase and the data phase of the AHB transfer.

5.3.6 AHB Slave – Direct Memory Access Burst Transfer

The AHB slave module is responsible for the actual Ethernet traffic. It is the AHB counterpart to the master module of the MAC unit, which controls the AHB transfers for Ethernet traffic. The master in the MAC unit is an AHB lite master, which allows the slave to omit the support of split and retry responses. Additionally, the slave is not able to handle pipelined write transfers. According to the AHB specification [6], the slave is not required to do so, as long as it keeps the ready signal deasserted during the last burst transfer, until it is ready to accept the new transfer.

The AHB master in the MAC unit belongs to the legacy hardware environment of the Ethernet NIC and is not to be changed in the course of this diploma thesis. Unfortunately, the AHB master does not completely conform to the AHB specification. Instead of using the AHB specified HSIZE and HBURST signals to indicate the length of a burst, it uses an unspecified, extra out-of-band signal to pass the information about the burst length to the AHB slave, while the HSIZE and HBURST signals are not implemented at all. Therefore, the slave in the bus-switching unit has to violate the specified rules of how to handle burst length information, in order to be able to communicate with the implemented AHB master of the MAC unit.

The AHB slave module has a special role in the bus arbitration mechanism. As already mentioned in section 5.3.3, the bus arbitration mechanism is integrated in the respective state machines. The two participating state machines are the header decoder module as well as the AHB slave module, as both receive data from external units and therefore have to initiate access to the shared resources of the bus-switching unit. Thus, on the one hand the AHB slave module acts as a controlling instance of the arbitration mechanism, on the other hand it is controlled by the header decoder, the second controlling instance of the arbitration mechanism, when an incoming PCI Express completion TLP has to be processed. Therefore, the two cases have to be handled and discussed separately.

When the AHB master in the MAC initiates a transfer, the slave module has to act as the controlling instance of the bus arbitration mechanism. The AHB slave can not prevent the master from starting a transfer, but it can prevent it from sending data. Therefore, when the master starts a transfer, the slave has to make sure that no other module currently has access to the shared resources, to the RX-, and TX-FIFO. This is done by observing the header decoder state machine and its starting condition. As already discussed, the header decoder state machine is only in idle state, when all processes controlled by the header decoder are in idle state as well. As the AHB slave module is the only module that can act independently of the header decoder, it suffices to observe the state of the header decoder state machine alone. Because of the lower priority of burst transfers, the starting condition of the header decoder state machine has to be observed as well. The starting condition of the header decoder is a non empty RX-FIFO. Therefore, the AHB slave is only allowed to start its operation when the header decoder state machine is in idle state and the RX-FIFO is empty. Otherwise, it has to wait in idle state for those two conditions to become true.

When the AHB master in the MAC unit initiates an AHB transfer, and the slave is not allowed to start its operation, it keeps the ready signal deasserted, thus preventing the master to continue the transfer. When the AHB slave is allowed to access the shared resources and therefore to start its operation, it asserts the ready signal and thus allows the AHB master to continue the transfer.

5.3.6.1 Dataflow of Received Ethernet Frames

When an Ethernet frame is received by the Ethernet MAC, it is split into cells of the specified cell size. The cell size is controlled by a read- and writeable register in the MAC. The MAC then starts an AHB burst write transfer, to write the data into the DMA memory area assigned by the device driver.

As already discussed in section 3.2.1, the protocol translation from AHB to PCI Express TLPs is integrated in the AHB interface modules. Therefore, the AHB slave has to translate each AHB burst transfer to a corresponding PCI Express TLP and store it in the TX-FIFO. When the AHB master initiates a write transfer, it starts with the address phase in the first clock cycle, where the destination address for the first data is given. Because the ready signal of the slave can only be deasserted when the slave encountered a new transfer, the master still sees the ready signal asserted. Therefore, in the next clock cycle it presents the first Double Word (DW) of data and the destination address for the second DW. The address for the first data is only present during the very first clock cycle of the AHB transfer. It has to be stored in an additional register, because the slave is not able to write the address information to the TLP in the very first clock cycle. First of all, it has to wait until it is allowed to start its operation and access the TX-FIFO, and even if it does not have to wait, the address information is not contained in the first eight bytes of the TLP which can be written immediately, but only in the second part of the header.

As soon as the arbitration controlling part allows the state machine to start the header of the TLP is built. The first eight bytes of the header don't require any information of the AHB transfer. Only the second eight bytes are dependent on the address of the AHB transfer. First, because the lower four bytes, bytes nine to twelve, hold the address itself, and second, because bytes thirteen to sixteen are required to hold data in the case of a non quadword aligned address, and random data otherwise. In the case of a quadword aligned address the first DW of data is needed one clock cycle later than with a quadword aligned address. For this reason, the ready signal is deasserted for one additional clock cycle, compared to the sequence of a non quadword aligned addressed transfer.

After the header is built, the data has to be written to the TX-FIFO. The 64 bit input port width of the TX-FIFO requires the AHB data to be buffered in a 64 bit wide register. The incoming 32 bit per clock cycle are written alternating to the lower and the higher DW of the register. After the higher DW is written, the data of the register is transferred to the TX-FIFO. This is done until all data of the burst is written to the TX-FIFO. Then the

control signal for the TX-FIFO control module to transmit the data to the PCI Express interface is asserted, and the AHB slave state machine waits until the TX-FIFO control module confirms the start of the transmission. Only after the confirmed transmission it changes back to idle state and thus releases its claim for the shared resources. If an error occurred during the AHB transfer, the data in the TX-FIFO is not transmitted, but deleted instead.

5.3.6.2 Data-Flow of Sent Ethernet Frames

When Ethernet frame has to be sent, the data first has to be read from the shared memory part of the host PC. Still, only the AHB master module of the MAC unit is able to initiate DMA burst transfers. Therefore, the device driver first has to write a register – register operations are initiated by the device driver – and by that inform the MAC, that the DMA memory area holds Ethernet data which is ready for transmission. The AHB master in the MAC module then initiates a read transfer.

The AHB slave in the bus-switching has to translate the read transfer to a PCI Express read request. Therefore, it has to build and transmit a TLP, following the same arbitration procedure as described for received Ethernet frames. When the transmission of the read request TLP is confirmed, the AHB slave state machine also changes back to idle, allowing other operations to take place until the PCI Express completion arrives. When the completion arrives, first the header decoder modules gains control, and processes the TLP as described in section 5.3.3. Because for a completion the single AHB slave module is the intended target for the data, and not one of the two AHB master modules, there is no need of the header decoder to examine the second part of the TLP, which holds the address. Therefore, after the first available quadword of the TLP has been read from the RX-FIFO, control and access to the shared resources is passed on to the AHB slave module.

The AHB slave then examines the remaining part of the header, which contains the information to which read request it belongs. When this information identifies the TLP to be the correct completion to the previously issued read request, the TLP is further processed. It has also to be distinguished between a quadword aligned address and non quadword aligned address, as again the bytes 13 to 16 of the TLP hold random data or the first data of the transfer, respectively. For the remaining data, at each clock cycle the lower and the higher DW of the data of the RX-FIFO are presented alternating to the AHB read lines. Every second clock cycle the RX-FIFO read signal is asserted to present the next quadword of data at the output of the RX-FIFO. This again is done, until all data of a burst is transferred. If an error occurred during the AHB transfer, the AHB transfer is cancelled and the remaining data of the TLP is removed from the RX-FIFO.

5.4 Pitfalls During Development

In the course of the design and implementation of the described functions, a couple of unexpected, but major challenges arose and had to be solved. Some of them arose from incomplete or wrong documentation, some of them because of third party errors, and some of them because of own faults. In the following a brief overview will be given, describing the errors encountered and the methodology used to solve them, as well as recommendations to avoid similar mistakes in the future.

5.4.1 Evaluation Board Errors

The first problems encountered had to do with the hardware of the evaluation board. At the very beginning of the practical work with this diploma thesis, some testing was done to try out and verify the design flow with the evaluation board. Therefore, the PCI Express reference setup provided with the evaluation board kit was altered and should be programmed to the FPGA on the evaluation board. First of all, some errors in the provided reference design had to be fixed. Apparently the source code of the reference design did not undergo a final check before distribution, as the project did not even compile. With later updates of the design, these problems were fixed by the developers.

5.4.1.1 Power Supply

After fixing those minor issues, the design was programmed to the FPGA. Unfortunately, the PCI Express core needs to be initialised, which is only done at startup of the host PC. Therefore, after programming the FPGA, the host PC has to be restarted. A hardware reset suffices to initialise the PCI Express connection. However, due to a misinterpretation of simultaneously appearing other errors in the first place, it was thought that a hardware reset is not enough, but the system had to be shut down and turned on again. When shutting down the system, the FPGA looses its setup, and is automatically loaded with the unaltered reference design, which is stored on an extra flash memory on the evaluation board. Usually it is possible to store own hardware designs in this flash memory to be loaded at startup, but not with designs using the Altera PCI Express MegaCore unit using only Altera's free OpenCore Plus hardware evaluation feature. This feature allows simulation and evaluation of the hardware design, but only for one hour after programming the FPGA. It is not allowed, and not possible to create a design file fitting the requirements for the flash memory to be loaded at startup. Therefore, the altered PCI Express reference design could not be tested, because either the PCI Express connection was uninitialised, or the altered design was lost due to power down of the host PC.

The evaluation board offers the possibility of external power supply in addition to the power supply provided by the PCI Express connection. In the originally included manual it was stated, that one could plug in the external power supply if the power supply of the

PCI Express was not sufficient [31]. Using the additional external power supply promised to solve the problem of the loss of the programmed design due to powering down the host PC. Hence, first the external power supply was turned on, and then the host PC was switched on. The result was unsatisfactory, as the switching MOSFET for the two-phase, synchronous step down switching regulator dissolved to black smoke.

Fortunately, operation with the PCI Express power supply only was not affected by this defect. The problem could have been avoided by studying the provided schematic of the board layout, but without thorough investigation of the functionality of the switching regulator IC it is not obvious that the concurrent attaching of both power supplies leads to a damaged board. However, the problem of the initialisation of the PCI Express core was solved by using the hardware reset instead of a power down of the host PC, and the evaluation board was repaired by replacing the damaged part.

Another error due to an issue of the evaluation board was the MDIO connection of the PHY as described in section 5.2.2.2. Later tests with different evaluation boards revealed that this issue was not a problem with all of the cards. The reason for this behaviour thus is still unknown and might result from a rather arbitrary appearing material fault. Still, the fact that more than one version of the design example exists, and that the latest version pulls the bus low as described, it seems that the problem affects a bigger number of boards.

5.4.2 Hardware Design Errors

In spite of the issues with the evaluation board itself, the hardware design itself provided some sources for errors as well. The causes for errors were manifold, as some of them resulted from a wrong interface implementation in the legacy hardware core, some resulted from incomplete or misunderstood documentation, or simply from faulty programming.

5.4.2.1 Documentation

Additionally to the misleading documentation of the power supply, another missing and misleading description in the manual for the evaluation board caused a lot of problems. As described earlier, it is not allowed to stall the PCI Express interface when transmitting a TLP to it. This limitation was not mentioned in the original manual, on the contrary, a timing example suggested that stalling the interface was possible [32].

When trying to transmit data to the PCI Express interface without a TX-FIFO, all register operations worked perfectly well, but no DMA transfer was received at the host PC. Moreover, after a certain amount of issued DMA transfers the PCI Express interface did not accept any further data, and the host PC running an ubuntu linux with kernel version 2-6-18 had a complete hang, and did not react to any input anymore. The reason for this behaviour was hard to find, but after several attempts the only real difference

between a DMA and a register transfer was the length of the data. Because of the rate-matching of the slower AHB interface to the faster PCI Express interface, with DMA transfers the data valid signal of the PCI Express interface was deasserted every second clock cycle. When this behaviour was changed to a constantly asserted data valid signal during the transmission of one TLP, the host PC was able to receive the TLPs and did not hang. After this discovery, the TX-FIFO was implemented to be able to transmit correct data without stalling of the PCI Express interface. Quite at the same time this error was discovered, Altera also updated the manual, explicitly stating that it is not allowed to deassert the data valid signal during the transmission of a TLP [20].

Two further errors resulted from a wrong or inaccurate interpretation of the documentation. In the beginning some register operations succeeded, and some failed. It could be identified, that the failed write operations always were three clock cycles long, while the succeeding operations were only two clock cycles long. The reason for that was the already described difference between quadword- and non quadword aligned addresses of the requests. The description in the manual was misinterpreted. The quadword alignedness was considered to be some system preference, not a packet-based, address dependant property. Therefore, it was not correctly implemented. A review with the supervisor solved the problem.

Another error that occurred is a quite well known, nevertheless it also happened during the design of the bus-switching unit. Resulting from inaccurate examination of the PCI Express interface user guide, the byte order mapping from the TLP definition to the PCI Express interface signals was implemented wrongly. Therefore, it appeared that rather randomly correct and broken TLPs were received. For this issue again a review with the supervisor helped to identify the problem.

5.4.2.2 Design

Some further design errors occurred during the development of the bus-switching unit. One of the errors resulted from a wrong AHB interface design in the existing MAC unit of the NIC, while the other errors were programming errors in the bus-switching unit. As already mentioned, the AHB master interface in the MAC unit does not correctly use the `burst` and `size` signals to indicate the length of a burst. During the development of the bus-switching unit this deviation from the specification was easy to identify. Unfortunately, there were more errors concerning the timing of the control signals. The AHB slave of the bus-switching unit was first developed against the AHB specification, but it did not work with the erroneous AHB master. Therefore, the slave had to be adapted to deliberately violate the specification, in order to be able to communicate with the master.

A faulty signal calculation led to sporadic errors, as some completion packets were not accepted. The tag that is used to identify a PCI Express completion as the response to a certain read request is always increased by one, after a read request is sent. Therefore, the value of the tag of an incoming completion must be the previous value of the tag, which

is the actual value minus one. The values of the tag are limited to 32, which requires a rollover from 31 to zero. The check for the incoming tag value can be performed by calculating the expected by subtracting 1 from the actual tag value. This was correctly done in the design, but the error came because of a wrong operator precedence. Because the PCI Express core can be configured to support more than 32 different tag values, the three preceding bits of the incoming TLP header are checked as well to make sure that not only the lower five bits do match. Checking for these additional bits introduced the error, as the command was:

```
if ( pcie_rx_data_i(15 downto 8) =
     "000" & s_pcie_dma_tag - conv_unsigned(1, s_pcie_dma_tag'length) ).
```

The leading zeros `"000"` were added first, and only after that the value was decreased. Therefore, on the underflow rollover the resulting value was not `"00011111"` as intended, but `"11111111"` instead. The cause for this problem was found during a design review with the supervisor.

Finally, an unwise design decision was made. As already mentioned, the bus arbitration scheme using mutual exclusion for the different modules was not the first choice. Instead, in the beginning a more demand dependant scheme was used, locking shared resources, especially the TX-FIFO only when really needed. In order to not loose any unnecessary clock cycle by blocking a resource, a Mealy machine was chosen for the bus arbitration. To cover all possible cases of simultaneous requests, a quite complex logic was needed, which was not intuitively understandable. The complex logic was error prone, and therefore the system was unstable. Though generally working, after the transfer of a few thousand to hundred thousand Ethernet frames, the system stopped working due to dropped TLPs, because the the device driver depended on their reliable delivery. Therefore, to save maintenance effort, the bus arbitration was changed to the described strictly sequential scheme. Doing this, performance was traded in for maintenance effort and stability, which finally led to a stable running PCI Express Ethernet NIC.

Chapter 6

Simulation and Verification

Simulation of hardware designs is a very powerful and important part of hardware development. Simulation can be used for testing, debugging, verifying, and optimising the code throughout the whole design process. The possibility to observe each and every signal at each and every instant of time provides great opportunities for testing and debugging of individual modules as well as of the complete system. The possibility for the mere manual examination of the signals is already a very important feature. The real power of simulation unfolds in the possibility for automatic observation not only of signal states at a given point in time, but of sequences of bundles of signals as well. Therefore, designs can be verified for correct top level functionality, as well as for inner timing requirements, adherence to bus specifications and 100% logic testing.

A big drawback of simulation is the big development effort needed to provide a reasonable simulation environment, which is able to perform the above mentioned tasks. Nevertheless, even for the development of comparably simple modules the development of a decent simulation environment must not be neglected. Although there are tools to analyse the actual signal flow in hardware for FPGA based designs, e. g. the Altera SignalTap II Logic Analyzer, such tools cannot replace simulation, they only can support simulation to tackle problems which are not accessible in simulation. An example for such a case is the described problem with the MDIO signal line due to unknown hardware failures which can not be simulated.

In addition to the lack for the mentioned automatic verification possible in simulation, with hardware inspection tools like the SignalTap tool, the design cycles are prolonged. For each small change of the logic, the complete FPGA based part of the system has to be resynthesised, remapped, refitted, and reprogrammed to the FPGA device. Such a cycle can take several minutes to hours, or even days for very complex designs. Thus, although in the beginning time for the development of a simulation environment has to be spent, in the end a lot of time can be saved by having a profound simulation at hand. Even more for mask programmable gate arrays, simulation is the only means for testing

and verifying, as a production cycle takes weeks to month, and expensive masks have to be produced for each production run.

In the course of this diploma thesis two simulation environments were developed, both having a different approach and different aims.

6.1 Accurate Altera PCI Express Simulation

One of these two simulation environments here is called the accurate Altera PCI Express simulation environment. Altera, because the complete environment for the simulation is provided by Altera, and accurate, because the Altera PCI Express interface as well as the data-flow on the PCI Express link, including configuration, management and flow control messages, is accurately simulated. Thus, this simulation environment is very important for the verification of the complete system, as all PCI Express transactions are properly simulated according to the timing and behaviour of the PCI Express interface.

Drawbacks of this simulation environment are the lacking flexibility for module testing of the bus-switching unit. For verification of the bus-switching unit not only a typical use case has to be investigated, but also corner cases have to be applied to assure that the unit reacts in a stable way under all possible operation conditions, and even in the case of erroneous input to the interface. Further, exact timing sequences have to be tested to make sure that all possible concurrencies are handled properly. Those possibilities are not offered by the Altera simulation environment, as it accurately simulates the PCI Express interface, but does not offer direct influence on the interface. For example, it is not possible to test the case of congestion with filled buffers of the interface, as no other traffic which could block the root port is simulated. The most important drawback though is the simulation time. Because of the accurate simulation of the PCI Express link, including the setup and initialisation, the simulation is very complex, and thus a simulation takes several minutes on a powerful computer until the transmission of the first TLP is simulated and can be observed.

Nevertheless, due to the accurateness, this simulation environment was very important to tackle the no-stall issue, which was described in section 5.4.2.1. At this point of time, the fast PCI Express emulation unit, as described in section 6.2, was already present and used. Because of the missing knowledge about the no-stall requirement of the Altera PCI Express interface it was not implemented in the emulation and could not help in finding the issue. The Altera simulation environment did show the same behaviour as the actual hardware system, and thus could be used for further tries and tests.

Figure 6.1 shows the architecture of the simulation environment. On the left side is the Device Under Test (DUT), the hardware design of the NIC. All other components are part of the simulation environment, simulating the physical channel, the root port as well as the device driver. The device driver simulating module is the module responsible for

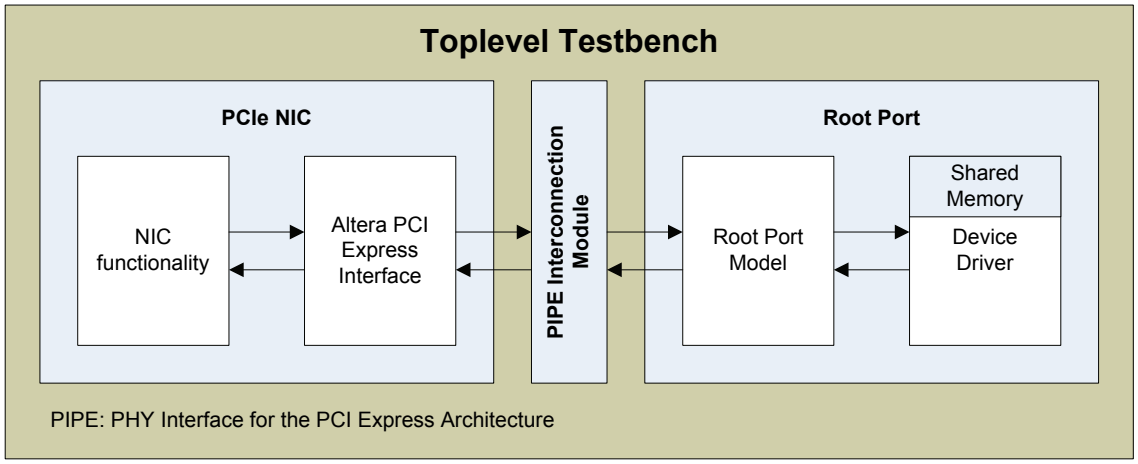


Figure 6.1: Architecture of the Altera simulation environment

interaction with the hardware design. It is used for the initialisation of the PCI Express connection as well as for all PCI Express transactions.

The device driver has a shared memory which is used for all PCI Express transactions. It is accessed by the device driver simulating module as well as by the root port simulating model, therefore it has to be realised as a shared memory. Its size has to be exactly double the size of the address space of the NIC, once the size for read data, and once the size to store data to write to the NIC. The framework of the simulation environment offers some functions to read and write the shared memory, as well as to initiate PCI Express transactions using data from the shared memory.

Register Operations

If the driver needs to write data to a register of the NIC, first the shared memory region for write transactions at the given address has to be written. A function of the simulation framework can be called to do so. Then another function of the framework has to be called to send a write request to the NIC. The function takes arguments to specify the address of the data in the shared memory as well as the destination address in the NIC and the size of the data. The framework then builds a TLP and sends it over the simulated physical link to the target, the NIC. Read requests are sent similarly, with the only difference that no data has to be written to the shared memory in advance. Still, the register address to read from has to be passed, as well as the address in the shared memory to store the response data to. Because a completion packet does not have a dedicated destination address, but only holds a copy of the lowest six bits of the address of the read request, the simulation framework has to map the data of the incoming completion to the specified address of the shared memory.

Ethernet Traffic

For Ethernet traffic, the device driver passively receives read and write requests. Similarly to the real use with hardware and the device driver in the host PC, first the buffer descriptor registers in the NIC have to be set. The transmit buffer descriptors are set to store the address of the shared memory which holds the Ethernet data for frames which have to be transmitted and the receive buffer descriptors are set to store the address of the shared memory which is intended to store the incoming Ethernet data. The Ethernet traffic then is completely handled by the framework without being noticed by the device driver module. This is an accurate simulation of the reality, where the shared memory is written and read by the DMA controller without notice of the device driver. Therefore, to simulate usual traffic, the PCI Express interrupts have to be observed.

However, no interrupt handling has been implemented in the Altera simulation environment, as it was not used for the simulation of a series of Ethernet frames, but only to verify the transmission of a single AHB burst transfer from and to the device driver. Due to the long simulation time of the Altera simulation environment, interrupt handling to allow the simulation of actual Ethernet traffic was only implemented in the fast PCI Express emulation.

6.2 Fast PCI Express Emulation

The second simulation environment is an emulation of the PCI Express interface without the use of any third-party simulation Intellectual Property (IP) cores. It is called fast because of its comparatively short simulation time. Compared to the accurate Altera PCI Express simulation, depending on the desired length of a simulation run, the speed-up ranges from one to two orders of magnitude. Especially with often required short simulation runs, e. g. until the first PCI Express transaction takes place, the speed up is in the range of two orders of magnitude, which saves minutes with each simulation run.

The emulation in the name comes from the nature of the environment. The fast environment is not intended to accurately simulate the behaviour of the Altera PCI Express interface unit. It is not designed as a *simulation* of the complete data path, but only as an *emulation* of the behaviour of the complete PCI Express side. It emulates the behaviour of the Altera PCI Express interface with the controlling device driver in the background, therefore combining the functionality of the PCI Express interface and the device driver. The emulation can be used to verify the input to the PCI Express interface and produce interface output by emulating a real PCI Express connection. In the following, TLPs which usually are initiated by the device driver, sent by the PCI Express root port, received by the Altera PCI Express MegaCore unit, and presented to the PCI Express interface are described as emulated received TLPs, because they are not actually transmitted, but only their reception is emulated. The fast PCI Express emulation simulation environment is simply called emulation unit.

As already mentioned, the main reason for the development of the fast emulation unit was the speed-up compared to the Altera simulation, as well as the more flexible control. In contrast to the Altera simulation the output signals can be directly controlled. Therefore, all kinds of timely sequences, error conditions, and concurrencies can be realised, enabling a thorough testing of the bus-switching unit. In order to be able to develop different test cases, the fast emulation was developed as a programmable testbench providing several commands to control the action sequence.

The fast emulation supports a couple of operations, which can be controlled by an external input file. In contrast to the accurate Altera simulation environment, no internal functions are defined which are called from within the simulation module, rather specific functions are mapped to a command and can be called from an external input file. The approach with an external input file offers more flexibility for the simulation setup. In contrast to an integrated approach, where functions are called from within the simulation module, with external input files, the simulation environment does not have to be recompiled when the input sequence or data is changed. This saves a lot of time when experimenting with different inputs. It also allows the simple creation of different test cases, with each testcase having a specific simulation sequence stored in a separate input file.

The internal realisation of the functionality also differs from the accurate Altera simulation. No shared memory is used for the data for PCI Express write requests. The data to write is directly passed to the module as argument to the command in the input file. For received completions, similarly to the Altera simulation a memory is used to store the data, but it is not a shared memory. For Ethernet frames, a dedicated memory is used just like in the real device driver. For each frame to be stored, a distinct memory area is reserved.

Write Verification for Register Operations

In addition to normal register read and write operations the emulation unit also supports write verification. When a register write with verification is requested, the emulation unit first emulates the reception of a PCI Express TLP by presenting a write request TLP on the interface to the bus-switching unit. Data and address are stored locally to allow for later comparison. Immediately after the write request, the reception of a read request for the same address is emulated. Then the completion is awaited and upon reception it is verified to be the answer to the previous read request. If so, the read data is compared with the previously stored data and a warning is written to the output if they don't match.

A read request immediately following a write request to the same address can, in general, lead to the problem that the read request still reads the old value of the not updated register. This happens only if the write request has not finished successfully until the read request is processed. In the present design, it is known that the AHB slaves keep the ready signal deasserted until the register write operation has completely finished. Therefore, it might be possible that the read request arrives while the write request

is still being processed, but the registers can not be accessed until the write request has completed its operation, therefore the subsequent read request always delivers the correct, new data.

Ethernet Traffic

For the simulation of Ethernet traffic, some preparations have to be made. As already mentioned, the emulation unit does not have a complete memory image of the registers of the hardware. To be able to handle the DMA transfers correctly, the emulation unit has to have knowledge about the addresses written to the buffer descriptor registers, as these addresses define the address space of the DMA memory region. Therefore, when the external input file requires to write data to the buffer descriptors, the emulation unit observes and stores the data written to the descriptors. The emulation unit then maps all incoming PCI Express requests, which are addressed to one of the DMA frame buffers to the corresponding memory of the emulation unit.

Interrupt Handling

The emulation unit is capable of emulating the behaviour of the device driver for Ethernet traffic. Therefore, the interrupt signal is observed and on assertion of the interrupt signal line the reception of a register write request is emulated to clear the interrupt. Further, the reception of register read operations are emulated to read the interrupt status registers. Depending on the read data of the status registers, further read operations to read the buffer descriptors are emulated. Similarly to the device driver, the emulation unit keeps track of the currently used buffer descriptors.

Verification of Received Ethernet Frames

It is possible to check incoming Ethernet frames for correctness. For simulation, an MII generator unit exists and is attached to the MAC. The unit can be programmed by an input file to generate MII signals as if the real Ethernet PHY had received a frame. The data for the Ethernet frame again can be loaded from an external file. This same file can be read by the PCI Express emulation unit as well. Doing so, the emulation unit can compare the data written to the DMA memory with the original source data from the input file. If the received data in the memory matches the data of the file, the complete data path through the Ethernet NIC is verified. If the data does not match, a warning is printed.

When Ethernet frames are sent, the emulation unit can also read data from a file. First, this data has to be stored in the memory area which is assigned to the current transmit buffer descriptor. The emulation unit emulates the reception of a write request TLP to the currently active buffer descriptor, telling the MAC that data is present. The read requests of the bus-switching unit are recognised and served with corresponding

completion packets, presenting the data of the DMA memory area to the PCI Express interface. The MII generator is also capable of verifying the received Ethernet frames against the data of a file. Therefore, also the path of a sent Ethernet frame can be verified.

6.3 Verification

Of the whole development and implementation process of a design, verification is the most important part. Verification of the resulting system is of course necessary to prove that the required functionality is fulfilled. It is a good idea though not only to verify a complete system, but to split the design in several distinct, independently verifiable modules. This was done during the development of this diploma thesis. Some testcases were developed and applied in simulation to test specific functions. However, most of the testcases evolved during development and were merged to a single testcase, verifying the complete functionality of the system.

After applying testcases for simulation, the complete system also was tested to work in hardware together with the device driver. In contrast to simulation testcases not every detail can be tested and analysed, but high network traffic and long test runs can be performed to also test for stability and reliability under high load.

All tests and verifications were done with 10 and 100 MBit/s Ethernet only. At the design time of this diploma thesis the MAC module of the legacy hardware did not support GMII. As the difference between MII and GMII only affects the connection between the MAC and the PHY, but not the connection between the MAC and the PCI Express interface where the bus-switching unit is involved, nothing except for the maximal achievable data throughput changes for the bus-switching unit. As the verification also aimed at verification for high load and congestion, the bus-switching unit is prepared to be connected to a Gigabit Ethernet capable MAC.

6.3.1 Verification by Simulation

For simulation the two described, different environments are available. As already mentioned, due to the long simulation time of the accurate Altera simulation environment, only simple testcases were applied to the accurate simulation environment. It was mainly used for testing register read and write operations and for short Ethernet packets during development. Most testcases were applied to the fast emulation unit. In the following, the applied testcases are listed and described, and the used simulation environment for each testcase is mentioned.

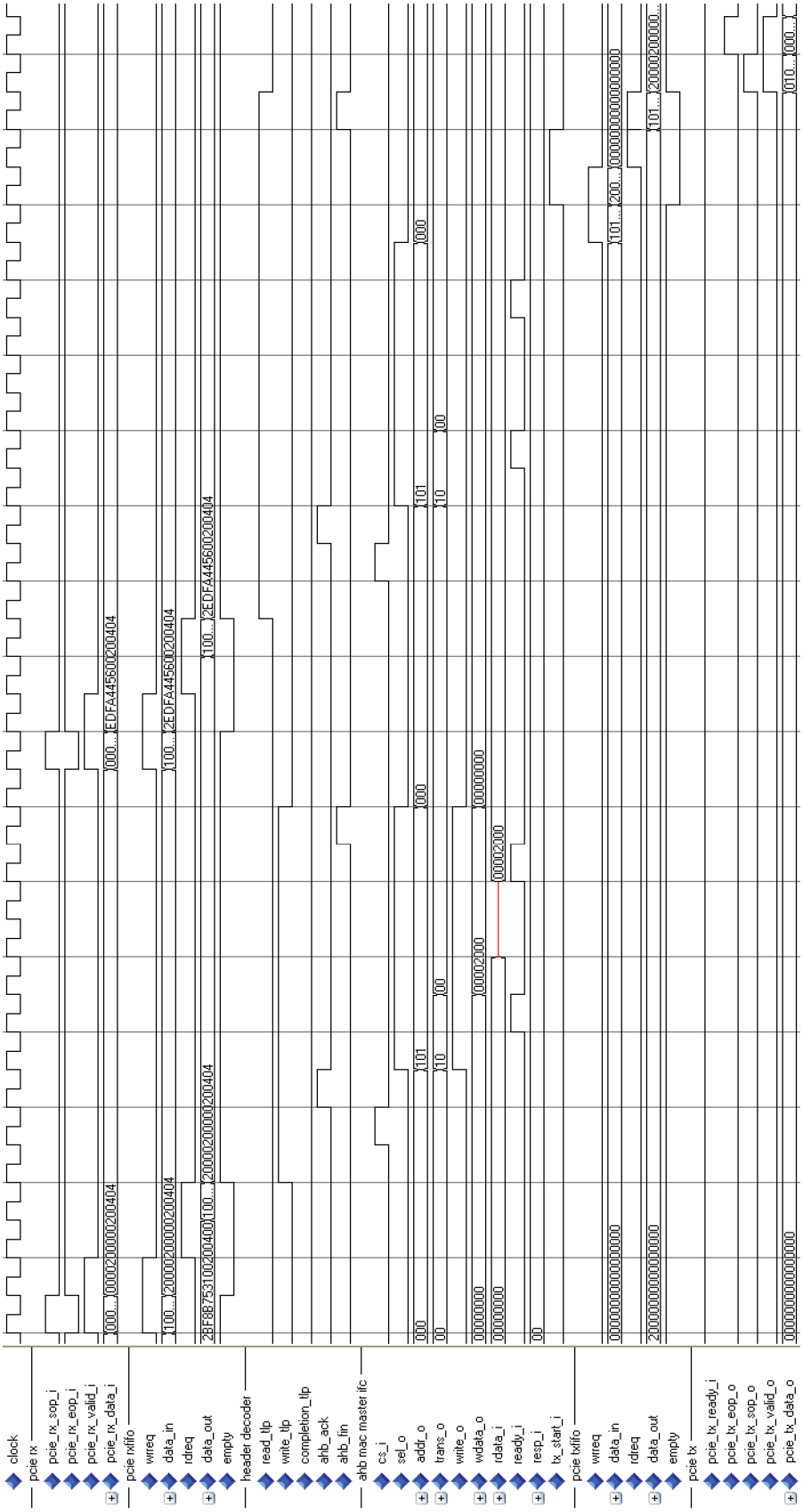


Figure 6.2: Register write verification in the accurate simulation environment

6.3.1.1 Register Write Verification

This testcase was applied to write a value to a register and verify that the value has been written. Write verification is not only possible in simulation, the device driver can be configured to perform write verification during normal operation. Therefore, a testcase using write verification not only verifies the correct register operation, but also simulates a regular use case of the real hardware.

Figure 6.2 shows a register write operation with write verification in the accurate simulation environment. A number of signals are depicted, showing the data flow during a register operation. The signals are sorted by the functional units they belong to. The first signal is the 125 MHz clock. The sections labelled `pcie rx` (at the top of the figure) and `pcie tx` (at the bottom of the figure) belong to the Altera PCI Express interface. The `pcie rxfifo` section shows the signal of the RX-FIFO which buffers the incoming PCI Express TLPs. The section labelled `header decoder` shows the signal of the separate packet-switching module, while the signals of the section `ahb mac master ifc` belong to the AHB master interface inside of the bus-switching unit, which initiates the AHB transfers for register operations. The `pcie txfifo` shows the signals of the TX-FIFO, which collects the complete TLP data until it is transmitted to the PCI Express interface.

In the beginning of a register read operation a PCI Express read request is sent. The asserted `pcie_rx_valid_i` signal indicates that valid data is available. The data is directly written into the RX-FIFO, with the `sop` and `eop` signals appended as the Most Significant Bits (MSBs), therefore the data in the FIFOs differs in the first digit from the data of the PCI Express interface.

The `empty` signal is deasserted and the header decoder can start reading the first part of the header of the TLP. The `write_tlp` signal is asserted by the header decoder, indicating that a register has to be written. Therefore, the chip select (`cs`) for the AHB master is asserted – with one clock cycle delay after it is known that the packet is a write request, because previously the address has to be decoded, as described in section 5.3.4. It is obvious that the duration of the packet transmission is only two clock cycles, which corresponds to two times a quadword of 64-bits. Therefore, this packet's address must be non-quadword aligned with the data being contained in the upper DW of the last transmitted quadword.

The AHB master then acknowledges its exclusive control over the shared resources, namely the RX- and TX-FIFO by asserting the `ahb_ack` signal and initiates a write transfer by asserting the select signal for the AHB slave of the MAC (`sel_o`). The `trans_o` signals are set to indicate a non-sequential transfer (10), the `write_o` signal is asserted and the `addr_o` signals are set to the register address. The Least Significant Bits (LSBs) of the second part of the TLP header hold the address of the register operation. As depicted, the `pcie_rx_data_i` addresses the register 0x404, whereas the `addr_o` shows 0x101. This is a pure presentational issue, as all addresses are known to be DW aligned addresses with the two LSBs being zero the `addr_o` signals simply omit the two LSBs, which results

in a presentation of 0x101. This is an uneven number, which again corresponds to a non-quadword aligned address and fits the prior observation of the short TLP.

The addressed AHB slave first delays the address phase by one clock cycle by keeping the `ready_i` signal low. This is not allowed by the AHB specification and is one of the errors of the AHB implementation of the legacy hardware mentioned in section 5.4.2.2. After the address phase is over the master presents the data to write, which was contained in the upper DW of the second part of the TLP. The slave again deasserts the `ready_i` signal and inserts a regular wait cycle. When the slave finally is ready to process the data, it asserts the `ready_i` signal and thereby ends the transfer. The master therefore informs the header decoder that it has finished its operation and releases its claim for the shared RX-FIFO by asserting the `ahb_fin` signal.

In the meantime a new TLP was already transmitted over the PCI Express system and arrives (by chance) right after the transfer has finished. This TLP is processed in exactly the same way as the previous one, with the only difference that this one is a read request. Therefore, the AHB master initiates a read request to the given address. The address phase again is illicitly delayed. When the AHB slave is ready it presents the data and again asserts the `ready_i` signal. The AHB master state machine again has exclusive access to the shared resources. Thus, it can access the TX-FIFO. It assembles a completion TLP with the read data in the payload and writes it to the TX-FIFO. When the whole completion packet is stored in the FIFO the content of the TX-FIFO is transmitted to the PCI Express interface and the AHB master again releases the claim for exclusive access to the shared resources by asserting the `ahb_fin` signal.

The device driver simulation module of the Altera simulation environment can then verify whether the received data coincides with the previously sent data or not.

6.3.1.2 Erroneous Requests

To verify the correct handling of erroneous PCI Express requests, a testcase was applied, which intermittently sends erroneous TLPs to the bus-switching unit. Correct handling of erroneous requests means that the bus-switching unit must return to a defined and operational state after the reception of erroneous requests. Nevertheless, it is desirable to inform the user or developer about the occurrence of an error. This is done by using the LEDs on the evaluation board. As too few LEDs are available, different errors are assigned different error codes. The binary error codes are displayed by the LEDs. As an erroneous state of the hardware usually lasts a few clock cycles at maximum, it is not enough to light a LED only as long as the erroneous state prevails, but to trigger the lighting of the LEDs by the detection of the error and keep it on until the user actively resets the error status LEDs. As it is possible and likely that more than one kind of error occurs at a time, the error codes are assigned in a way to reduce the risk of confusing the code of a single error with the displayed code of two different, consecutively occurred errors.

In the simulation, the state machines and signals can be observed individually, so it can be verified that the behaviour after an erroneous input is not deteriorated by the single erroneous event. Also, the observation of the LED controlling signals makes it easy to locate and identify detected erroneous events.

The testcase sends completely broken TLPs of arbitrary length to check the correct clean-up of the RX-FIFO. The bus-switching unit has to take care that no part of a broken TLP blocks the RX-FIFO and therefore leads to a hang of the hardware. On the other hand, no correct information must be lost even if it is in between broken data. The clean-up is done by (destructively) reading the FIFO until a correct part (SOP signal asserted and a correct first part of a TLP header) is found. But even if the first part is correct and consecutive DWs of the TLP are corrupt (e. g. the claimed size of the header does not match the actual size of the TLP), all actions that have been taken so far have to be made undone, if possible, or at least the information about an error has to be forwarded.

A similar error is the transmission of a correct TLP with a wrong register address. This error is only detected by the AHB slave. The slave then issues an error response, and the master has to cancel the transfer and release its claim for the shared resources.

All these kinds of errors were injected using the emulation unit. The verification of the return to the defined operational states was done manually. Additionally, after the error injection register operations with write verification and verified Ethernet transfers were performed. This does not verify the correct behaviour, but a failure of those operations would at least indicate a wrong error handling of the bus-switching unit.

6.3.1.3 Dropped Completion

Another testcase also checks for correct error handling, but aims at disturbing the split transaction mechanism of PCI Express read requests. Therefore, the emulation unit can be programmed to not return a completion for the next read request, or to send a completion that does not correspond to the last request.

The bus-switching unit implements a time-out counter which observes the time the root port needs to answer a read request with the corresponding completion. When this time-out counter expires, the still active AHB transfer has to be cancelled. If the corresponding TLP would arrive later, it must not be accepted as a correct completion anymore. According to the PCI Express specification this time-out must not be shorter than 50 µs, and is recommended to be not shorter than 10 ms. In the hardware version a 10 ms counter is implemented, but to avoid long simulation times to test the time-out the counter is reduced to 4 µs when used with the emulation unit.

The verification of the correct handling of this kind of error was done in a similar manner to the previously mentioned injected errors, by manually inspecting the state and performing automatically verifiable operations afterwards.

6.3.1.4 PCI Express Flooding

To verify the correct behaviour of the bus-switching unit with a flood of PCI Express TLPs, a different testcase was applied. This testcase floods the bus-switching unit with PCI Express requests and/or completions faster than they can be handled. It was applied to the emulation unit, because this unit is capable of emulating the reception of consecutive TLPs without any delay in between. Therefore it is possible to fill the RX-FIFO without artificially having to slow down the bus-switching unit.

Before the RX-FIFO gets full, the ready signal to the PCI Express interface has to be deasserted to perform back pressure and hinder the PCI Express interface to transmit more TLPs. The ready signal has to be deasserted early enough, considering that the PCI Express interface can continue transmitting data for three clock cycles after the deassertion of the ready signal. If this is not done correctly, data is lost, which is not allowed. The verification of the behaviour for this testcase was performed in the same way as with the injected errors.

6.3.1.5 Regular Ethernet Traffic

The operation of the bus-switching unit with regular Ethernet traffic was verified by using the MII generator unit. For only a few Ethernet packets (not more than the number of initialised buffer descriptors), the accurate simulation environment was used. No automatic verification of the sent and received Ethernet frames is implemented in the accurate simulation environment, but the data stream on the MII connection can be inspected and manually compared to the known, expected data.

In the emulation unit, the complete interrupt handling procedure is implemented, and therefore arbitrary Ethernet traffic can be simulated. Due to the automatic handling, the emulation unit has knowledge about when a received Ethernet frame is available in the simulated DMA memory, and therefore can verify the content of this memory automatically with the known sent data. In the other direction, the MII generator is also capable of comparing received Ethernet data (sent by the simulated NIC) with a given data set. The data of the Ethernet frames which's transmission is simulated is stored in a file. As both units, the PCI Express emulation unit and the MII generator can access this file, they both have exact knowledge of what data is expected to be transmitted by a specific Ethernet frame.

Figure 6.3 shows the transmission of a single cell of a received Ethernet frame to the DMA memory in the host PC using the emulation unit. In the `ahb slave ifc` section, it can be seen that the AHB master interface of the MAC unit asserts the control and select signals to activate the AHB slave interface in the bus-switching unit. The slave employs the same wrong `ready_o` behaviour as previously mentioned with the AHB slave of the MAC unit. This is the case because also the master of the MAC is implemented

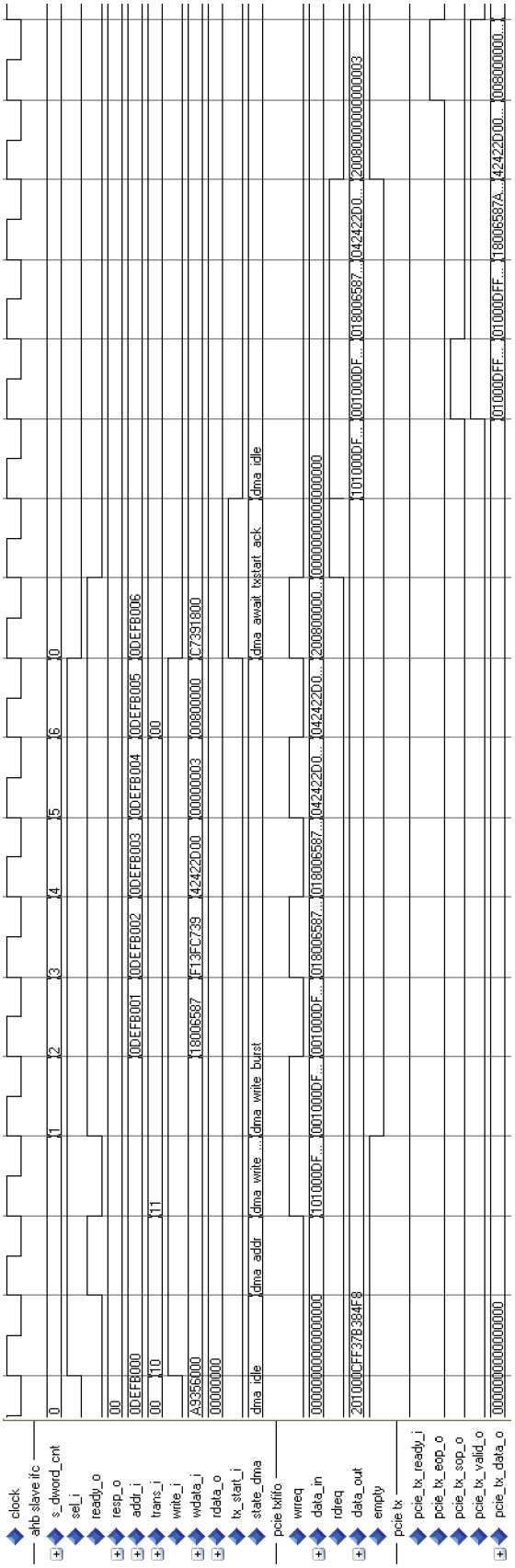


Figure 6.3: Transmission of a cell of a received Ethernet frame

incorrectly and does not work with a correctly asserted `ready` signal during the idle state of the slave.

In the figure it can be seen that first the master issues a non-sequential transfer (10 of the `trans_i` signal), and only after the slave has acknowledged the start of a transfer by asserting the `ready_o` signal the master switches to sequential transfer (11 of the `trans_i` signal), as defined in the AHB specification. The slave starts to assemble the PCI Express TLP in the TX-FIFO, which can be seen by the asserted `wrreq` signal and the activity on the `data_in` lines in the section `pcie txfifo` of the figure. The address `addr_i` is identified to be a quadword aligned address. Therefore, the first two quadwords of the TLP contain only the header and no data.

Then the data payload is assembled from the received AHB data. The first DW of data was already present at the `wdata_i` lines before the transfer was started. Therefore, this first DW is written to the TX-FIFO and simultaneously the AHB `ready_o` signal is asserted to indicate that further data can be processed. The `wrreq` signal of the FIFO is asserted only every second clock cycle, which is due to the rate-matching of the slower AHB to the faster PCI Express interface. It can be seen that the cell size is set to six DWs, which is also indicated by the `s_dword_cnt` signal.

When the complete cell is written to the FIFO, the transmission to the PCI Express interface with no inserted wait states can start, which is seen in the `pcie tx` labelled section. As soon as the transmission to the PCI Express interface has started, the AHB slave interface is informed about this fact by the asserted `tx_start_i` signal and returns to *dma_idle* state, thus releasing its claim for exclusive access to the TX-FIFO resource.

6.3.1.6 AHB Checker

For verification of a correct AHB implementation Altera offers AHB checker modules, which observe the AHB signals and check them for conformity with the specification. As already mentioned, the legacy hardware design has several errors in the AHB implementation, which produce a long list of errors when checked with the AHB checker modules.

In addition to the task for this diploma thesis where it was not allowed to alter the legacy hardware design, the AHB master side of the MAC was corrected due to private interests. The AHB checker modules confirmed the correctness of this altered implementation. The correction of the AHB master unit required some changes in the timing of the AHB module. Unfortunately, this altered timing had side effects on the complete MAC unit – with the correct AHB implementation it did not issue any interrupts. This shows the tight coupling of the AHB interface with its surrounding application module, as it was discussed in section 3.2.1.

6.3.2 Hardware Testing

To test the hardware design under high load, the evaluation board was plugged into a standard PC. It was connected to an 100 Mbit/s office network and incoming and outgoing flood ping was performed. Additionally, large files with the size of several Gigabytes were transferred in both directions using the SSH File Transfer Protocol (SFTP). Simultaneously, the PTP stack for clock synchronization was active to synchronize the clock of the evaluation board to a high-precision reference clock.

As SFTP is a secure and reliable protocol for data transmission, the received data is guaranteed to be unaltered. Thus, if the SFTP data transfers succeed, the data was successfully processed by the NIC implementation on the evaluation board. It is not guaranteed that no Ethernet packets have been dropped, for which the TCP protocol makes up, but it is guaranteed that over a longer time the hardware works correctly. The clock synchronization worked perfectly well aside to the high network load.

Chapter 7

Conclusion

This diploma thesis presented the evolvement of an existing NIC hardware design to be prepared to support Gigabit Ethernet over copper and optical fibre connections. Therefore, a bus-switching unit had to be developed to connect multiple AHB interfaces to a PCI Express interface. Before and during the design of this unit a number of design decisions had to be taken to develop a robust test platform that could support Gigabit Ethernet. The main influencing factors were the estimated development time and the primary intention for the use in a clock synchronization network without the urgent need for high data throughput. If possible along with these main requirements, flexibility for further enhancements was also considered.

For the PCI Express protocol core it was decided to use an existing hardware core, as PCI Express is a highly sophisticated protocol with the need for complex logic to implement the required functions for all three layers of the specification. Thus, the development of a PCI Express protocol core from scratch would have greatly extended the design time, would have been economically unreasonable, and would have delayed the availability of the hardware for scientific valuable measurements. To best fit the PCI Express protocol core to the used FPGA, a core provided by the FPGA developer Altera was chosen.

Several possibilities for configuring and connecting the PCI Express protocol core to the application hardware are available. The selected interface relieves the application hardware of a number of management tasks, e. g. the handling of corrupt and illegally addressed packets, and of buffering and managing the packet flow control, still leaving enough flexibility to allow for a customised handling of PCI Express TLPs.

The bus-switching unit for the connection of the AHB interfaces of the existing legacy hardware and the PCI Express interface was designed from scratch. There are existing AHB IP cores available, but as the design of such an AHB module is not very complex, and as the bus communication has to be tailored to the application's needs anyhow, it is not reasonable to use an existing core for the AHB interfaces. The requirement for interconnecting the bus interfaces was identified as a packet-switching problem similar to the well known field of packet switching in Ethernet. Therefore, knowledge and experience

from Ethernet switches was used to build a unit comparable to a shared memory based, cut-through packet switch employing time division switching. The focus was on a robust, easily maintainable design with less regard to scalability and data throughput, which are not stringent requirements for the hardware unit.

Extra effort was made to allow for fast, thorough and accurate verification of the hardware system. This resulted in the development of two distinct simulation environments, each designed to meet their specific intentions of accuracy and speed.

In summation of the presented work the lessons learned during the design and implementation progress have been compiled and written here. Finally, an outlook for further enhancements of the hardware design is given.

7.1 Lessons Learned

The greatest mistake you can make in life is to be continually fearing you will make one.

– Elbert Hubbard

During the design and implementation many challenges were raised that had to be tackled, and some of them have already been mentioned. During any creative work mistakes happen to occur. The important thing is to draw conclusions and learn how to avoid them in the future. Personal experiences and conclusions are listed on how to improve the applied practical procedures for future design and implementation projects.

Importance of Simulation and Verification

The most significant lesson learned in the course of this diploma thesis was the importance of simulation and small step verification. Although the complexity of the design, and thus the importance of simulation was underestimated in the beginning, fortunately a simulation environment was developed. In the end it proved to be extremely valuable to have a powerful simulation environment at hand to verify the design. A lot of time was saved by the development of a decent simulation environment right from the start, instead of relying on the way slower, less capable hardware debugging mechanisms.

The design process can greatly benefit from early thoughts regarding design for testability. Already at the system level design, when only a rough overview of functional blocks is known, considerations about functional partitioning for good testability should be taken. When functional parts can be split up inro smaller submodules, which can be tested and verified separately, unforeseen hidden mistakes of the bigger system can be avoided. Therefore, a lot of debugging time and money are saved.

It is advisable to prepare the design in advance in order to have some debugging mechanisms available. Preparations should be made to have some debugging information even

in the final hardware realisation. Examples are the LEDs for simple error reporting, live LEDs for certain critical modules to detect a hang, or the storing of state histories of state machines. For hardware signal inspection tools with limited memory capacity, e. g. the Altera SignalTap II tool, it might be useful to implement some kind of an event clock. The inspection tools store the state of the specified signals at each rising clock edge. If there are long idle cycles, a lot of memory is wasted by storing irrelevant data. An event clock only shows rising clock edges when some remarkable events which are worthy to be observed occur, thus much more relevant events can be recorded by ignoring irrelevant data.

Information Updates

Another critical issue during the development of this diploma thesis was wrong or inaccurate information of data-sheets, user guides, and manuals. Especially with relatively new products (younger than approximately two years) it must be expected that there are still previously undetected hardware issues or inaccurate documentation information. Therefore, updates for the product, erratas for documentations, and probably existing discussion boards or developer blogs have to be checked for updates on a regular basis, especially when major, inexplicable errors occur. It must also be kept in mind that professional product documentations are error-prone and thus cannot always be totally trusted.

If a development is a part embedded in a larger project, or surrounded by third-party components such as third-party IP cores or device drivers, it is important to keep track of updates of the components used. Particular attention has to be paid to change-logs. It might be a bad idea to frequently update to new, probably less reliable versions if they just support additional features not needed by the specific application, or if regular interface changes are performed. On the other hand, if there are regular bug-fixes it is highly recommend to update to the new versions of the third-party components.

Tracking Changes

A further important point for development and testing is the tracking and tracing of performed changes. When debugging a specific error some modifications might be made to several modules to narrow down the problem. It is very important to clearly mark the performed changes to be able to undo them later when the problem was found and solved.

For example, during tests of the register operations of the bus-switching unit, certain parts of the device driver were commented out to narrow down the possibilities for errors. If this applied limitation of the device driver was not removed after the tests, later broader tests would have shown only limited functionality. Since an error source can not always be easily localised it might happen that efforts are taken to find the reason for the limitations in the hardware, even though the (thought to be fully functional) device driver is the real source.

Time can be saved by marking modified portions of otherwise stable code and recording performed tasks and changes by writing some kind of journal to simplify later tracking of the activities.

It is also a good idea to mark questionable sections which are designed for a quick proof of concept and not for a robust final product. When a first test design evolves to a full product, all marked sections can be searched and worked on. Marking them during the first design when it is known that the functionality is limited might save a lot of time searching for the reason for incompletely supported functions during full system verification.

7.2 Outlook

This thesis showed the needs for an evaluation network card for the use in high-speed Ethernet environments as a node supporting high-precision clock synchronization. It further showed that the developed hardware is prepared for the integration in a Gigabit Ethernet environment. All preparations to allow for delay measurements of Gigabit Ethernet over copper and optical fibre links were made. As soon as a a Gigabit Ethernet capable MAC unit is at hand, the hardware can be used to perform these measurements. During the description of the design, some possibilities for further improvements of the whole system as well as of the bus-switching unit in particular were mentioned already. It is obvious that the current design can be enhanced by tackling these issues.

The most important enhancement to the developed hardware is the integration of the newly available, third party Gigabit Ethernet MAC. Only the integration of this MAC enables the developed hardware to deliver the desired measurement results for clock synchronization over Gigabit Ethernet. As the new Gigabit MAC is designed for the same platform as the original legacy MAC was, again some steps as described in section 5.2 will have to be performed in order to enable the unit to work on the Altera Stratix II GX FPGA. Additionally, it can be expected that the incorrectly implemented AHB interfaces will be corrected. This requires the adaptation of the bus-switching interfaces to not only work with the incorrectly implemented interfaces, but also with the correct implementation.

Another possible improvement concerns the system design of the interconnection interfaces. The TX-FIFO in the bus-switching unit and its control logic is only needed because the AHB connection has to be rate-matched to the PCI Express interface. If the AHB for DMA burst transfer between the MAC and the bus-switching unit was changed to a 64-bit wide data connection, which is possible with the specification of AHB, the complete logic for rate-matching could be saved. This would save resources, it would increase the data throughput and decrease the transmission delay for TLPs because the TLPs could be sent directly without the need for buffering before transmitting. The bus-switching unit thus would become simpler, smaller and faster. As this improvement incorporates the legacy

hardware as well, it is a question of system design and specification. The requirements for this diploma thesis did not allow for such a change, but it is recommended to perform this conceptual change in the future.

A different approach can be made to increase the speed and data throughput of the bus-switching unit. The simple restrictive locking scheme for mutual exclusion is easy to understand and to maintain, but it greatly reduces the performance of the hardware. If a more sophisticated locking and arbitration mechanism would be used, the maximum data throughput could be greatly increased. On the other hand, this would make the logic larger and more complex, introducing new error sources. As the intended application of the hardware is the use in test and measurement networks for clock synchronization, and as the driving argument for the use of Gigabit Ethernet are the delay jitter properties and not the data throughput, the bus-switching unit is not required to be optimised for high throughput and performance. However, it is possible to enhance the design towards high throughput.

On behalf of the simulation and verification, the simulation environment can be extended to also support programmable AHB master and slave simulation modules. By doing this, the bus-switching unit could be verified completely separated from the rest of the system. This would enable testing for all possible inputs from the AHB side as well as from the PCI Express side. Doing so, a 100% code- and branch-coverage of the bus-switching unit simulation could be achieved. Additionally, further automatic verification could be implemented in the simulation environment.

Finally, the intended measurements on delay and delay jitter for Gigabit Ethernet over copper and optical fibre connections can be performed. The precision of the clock synchronization algorithm over these connections can be investigated and verified by measurements. Test networks for clock synchronization over long haul optical fibre based Ethernet can also be implemented.

List of Figures

Acronyms

AHB:	Advanced High-Performance Bus
AMBA:	Advanced Microcontroller Bus Architecture
AP:	Access Point
ASIC:	Application Specific Integrated Circuit
CERN:	Conseil Européen pour la Recherche Nucléaire, European Organization for Nuclear Research, the world's largest particle physics laboratory
COTS:	Commercial Off-The-Shelf
CRC:	Cyclic Redundancy Check
CSC:	Clock Synchronization Cell, responsible for timestamping
CSMA/CD:	Carrier Sense Multiple Access/Collision Detection
DMA:	Direct Memory Access
DUT:	Device Under Test
DW:	Double Word, 4 bytes, 32-bit of data
ECRC:	end-to-end CRC
EOP:	End Of Packet
ε-WiFi :	Embedded Position Determination and Security in Wireless Fidelity networks
FCS:	Frame Check Sequence
FIFO:	First In First Out memory structure
FIT-IT:	Forschung, Innovation, Technologie – Informationstechnologie
FPGA:	Field Programmable Gate Array, the acronym FPGA is used for a device of the respective technology

FSM:	Finite State Machine
GMII:	Gigabit Media Independent Interface
IC:	Integrated Circuit
ID:	Identification
IP:	Intellectual Property, third party hardware cores are called IP
ISO:	International Organization for Standardization
LAN:	Local Area Network
LED:	Light Emitting Diode
LSB:	Least Significant Bit
MAC:	Media Access Control
Mb:	Megabit
MAN:	Metropolitan Area Network
MDC:	Management Data Clock, the clock signal for the MDIO interface
MDI:	Media Dependent Interface
MDIO:	Management Data Input/Output, a tristate bus management interface between the Ethernet PHY and the station management part of the MAC
MHz:	Megahertz
MII:	Media Independent Interface
MIIS:	MII Scanner
MM:	Memory Mapped
MPGA:	Mask Programmable Gate Array, non reprogrammable, production mask defined chips
MOSFET:	Metal-Oxide-Semiconductor Field-Effect-Transistor
MSB:	Most Significant Bit
NIC:	Network Interface Card
OSI:	Open Systems Interconnection
PC:	Personal Computer
PCB:	Printed Circuit Board

PCI:	Peripheral Component Interconnect
PCIe:	PCI Express
PHY:	Physical Layer, the acronym PHY is used for a physical layer interface device
PLL:	Phase Locked Loop
PTP:	Precision Time Protocol
QoS:	Quality of Service
RAM:	Random Access Memory
RGMII:	Reduced Gigabit Media Independent Interface
RMII:	Reduced Media Independent Interface
RX-FIFO:	Receive FIFO
SDS:	Space Division Switching
SERDES:	Serializer/Deserializer
SFD:	Start-of-Frame Delimiter, indicates the start of an Ethernet frame
SFP:	Small Form-Factor Pluggable, modular hot-pluggable optical transceivers
SFTP:	SSH File Transfer Protocol
SMA:	Sub-Miniature-A
SOP:	Start Of Packet
SOPC:	System on a Programmable Chip
SSH:	Secure Shell
TCP:	Transmission Control Protocol
TDoA:	Time Difference of Arrival
TDS:	Time-Division Switching
TLP:	Transaction Layer Packet
TX-FIFO:	Transmit FIFO
VHDL:	Very High Speed Integrated Circuits Hardware Description Language
WLAN:	Wireless LAN

Bibliography

[1] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[2] Excerpts from a conversation with gordon moore: Moores law, 2005.

[3] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, July 2008.

[4] Zafer Sahinoglu, Sinan Gezici, and Ismail Guvenc. *Ultra-wideband Positioning Systems*. Cambridge University Press, 2008.

[5] David J. Law, editor. *IEEE 802.3 (tm) Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. IEEE, December 2005.

[6] ARM. *AMBA (tm) Specification*, 2.0 edition, May 1999.

[7] R. Holler, T. Sauter, and N. Kero. Embedded SynUTC and IEEE 1588 clock synchronization for industrial Ethernet. In *IEEE Conference Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03*, volume 1, 2003.

[8] Georg Gaderer. *Fault Tolerance Enhancements to Master/Slave Based Clock Synchronization*. PhD thesis, Vienna University of Technology, Institute of Computer Technology, November 2008.

[9] Patrick Loschmidt, Reinhard Exel, Anetta Nagy, and Georg Gaderer. Limits of Synchronization Accuracy Using Hardware Support in IEEE 1588. In *ISPCS 2008, International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 12–16, Ann Arbor / U.S.A., September 2008.

[10] Martin Horauer. *Clock Synchronization in Distributed Systems*. PhD thesis, Vienna University of Technology, Institute of Computer Technology, February 2004.

[11] Faraj Nassar. PCI Express based Embedded System. Master's thesis, Vienna University of Technology, Institute of Computer Technology, October 2007.

[12] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 250–256, New York, NY, USA, 2000. ACM.

[13] Mathias Hein. *Switching-Technologie in lokalen Netzen.* Internat. Thomson Publ., 1996.

[14] MG Hluchyj and MJ Karol. Queueing in high-performance packet switching. *IEEE Journal on Selected Areas in Communications*, 6(9):1587–1597, 1988.

[15] ISO/IEC. *ISO/IEC 7498-1 Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, November 1994.

[16] RMII Consortium. *RMII (tm) Specification*, March 1998. Revision 1.2.

[17] Hewlett Packard Company, 3000 Hanover Street, Palo Alto, CA. *Reduced Gigabit Media Independent Interface (RGMII)*, January 2002. Version 2.0.

[18] How does ahb differ from ahb-lite?, September 2008.

[19] PCI-SIG. *PCI Express (tm) Base Specification*, March 2005. Revision 1.1.

[20] Altera Corporation. *PCI Express Compiler User Guide*, 8.0 edition, May 2008. chapter 5.

[21] R. Holler, M. Horauer, G. Gridling, N. Kero, U. Schmid, and K. Schossmaier. SynUTC-high precision time synchronization over Ethernet networks. *CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH-REPORTS-CERN*, pages 428–432, 2002.

[22] IEEE Std. 1588 - 2002 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2002*, pages i–144, November 2002. Replaced by 61588-2004.

[23] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. *Special Issue on the Challenge of Global Time in Large ScaleDistributed Real-Time Systems*, 12(2):173–228, March 1997.

[24] M. Mastretti, M.L. Busi, R. Sarvello, M. Sturlesi, and S. Tomasello. Vhdl quality: synthesizability, complexity and efficiency evaluation. In *Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC '95., European*, pages 482–487, September 1995.

[25] T.J. McCabe. A complexity measure. In *IEEE Transactions on Software Engineering*, volume SE-2, pages 308–320, December 1976.

[26] Altera Corporation. *Quartus II Version 7.2 Handbook – Volume 4: SOPC Builder*, October 2007.

[27] Altera Corporation. *PCI Express Compiler User Guide*, 8.0 edition, May 2008. pages 2-6ff, 5-66–5-72.

[28] Altera. *Stratix II GX PCI Express Development Board Reference Manual*, 1.0.1 edition, April 2007.

[29] Marvell. *88E1111 Datasheet Integrated 10/100/1000 Ultra Gigabit Ethernet Transceiver*, October 2006.

[30] Altera Corporation. *Stratix II GX PCI Express Board*, c-1 edition, September 2006.

[31] Altera. *PCI Express Development Kit, Stratix II GX Edition - Getting Started User Guide*, 1.0.1 edition, August 2006. page A-1.

[32] Altera Corporation. *PCI Express Compiler User Guide*, 7.2 edition, October 2007. page 5-11.