An FPGA Implementation for a High-Speed Optical Link with a PCIe Interface

Edin Kadrić

Department of Electrical & Computer Engineering

McGill University

Montreal, Canada

October 2011

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

Abstract

This thesis describes the design and implementation of an optical fiber based high speed interface between two computers. The system is particular in that the data transits in a Field Programmable Gate Array (FPGA) situated between each computer and the optical fiber link. The measured full duplex speed for exchanging data between two C programs running on two different computers is over 8Gbit/s, including encoding, protocol and software overhead. This design is suited for applications requiring high bandwidth between two computers, and since an FPGA sees all the data being exchanged, it can be used as a fast and flexible data processing tool: Error correction, debug support, data analysis, encryption and compression are all possible uses where the FPGA can save the Central Processing Unit (CPU) an important amount of computing cycles.

Résumé

Cette thèse décrit la conception et l'implémentation d'une interface à grande vitesse entre deux ordinateurs, basée sur un lien en fibre optique. La particularité du système est que les données transitent dans un "Field Programmable Gate Array" (FPGA) qui se trouve entre chaque ordinateur et les câbles en fibre optique. La vitesse de transmission bidirectionnelle de données d'un programme en C à un autre via cette interface a été mesurée à plus de 8Gbit/s. Ceci inclus les ralentissements dus à l'encodage, le protocole de communication et le logiciel. Cette conception convient à des applications demandant une bande passante importante entre deux ordinateurs, et comme un FPGA observe toutes les données échangées, ce dernier peut être utilisé comme un outil de traitement de données rapide et flexible: La correction d'erreur de transmission, le support de débogage, l'analyse, le chiffrement et la compression de données sont toutes des applications potentielles où le FPGA peut alléger la tâche de l’unité centrale.

Acknowledgements

I would like to thank my supervisor, Professor Željko Žilić, for his incredible support, guidance and insightful advice. His great vision and knowledge allowed me to make rapid progress and deliver tangible results.

Equally important, I must thank my co-supervisor, Professor Naraig Manjikian from Queen's University. His great knowledge, experience and dedication to this project helped me fix and avoid major issues and saved me valuable time.

I would like to thank the other students in McGill's VLSI Design lab for their motivation and advice, with a special mention to Bojan Mihajlović for helping me set up the workstation, together with Logan Smyth and Omid Sarbishei who have explored this project before me and provided me with useful information when I was just starting.

Although the writing of this thesis does not make it obvious due to a focus on other features of the system, setting up a working and efficient driver to interface with the hardware was a complex task. Still, it was certainly made much easier by the publicly available source code of a basic driver written by Leon Woestenberg and Nickolas Heppermann. I would also like to thank the (few) people who answered the half a dozen help messages I wrote on Altera forums.

In general, I would like to thank all the teachers and Professors I have had, as well as everybody who taught me something and contributed to my education.

Very special thanks go to my mother, my father and my brother, who have always shown an amazing support and a great dedication to my success.

Finally, I have to thank McGill University, Hydro-Quebec, the governments of Canada and Quebec, my supervisor, and the “Fonds Québécois de la recherche sur la nature et les technologies” (FQRNT) for the scholarships, fellowships, and financial support they provided me throughout my undergraduate and graduate studies.

Contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

1.1.1 Faster optical fiber links and applications

Increasing demand for faster internet access and higher quality multimedia transmission has led to a fast growth of computer network bandwidth capabilities. Ethernet over a twisted-pair cable has long been the link of choice between nodes in Local Area Network (LAN) communication, and has more recently started being replaced by not always faster, but more convenient Wi-Fi technologies. Ethernet was developed between 1973 and 1974 by the Palo Alto Research Center (PARC), and was first commercially introduced in 1980 [5]. First designed for speeds of 10Mbit/s, later standards increased this value to 100Mbit/s, and then 1Gbit/s. The latter technology, also called Gigabit Ethernet (GbE or 1 GigE), was introduced in 1999, and has a bandwidth of 1Gbit/s. But traditional copper-based physical implementations cannot follow the most recent demand for high speed networks, and due to their higher attenuation and interference, they are continuously being replaced with optical fiber cables. This is the case for some 1 Gigabit Ethernet standards, and for the more recent 10 Gigabit Ethernet one, first published in 2002, as well as 40 Gigabit Ethernet and 100 Gigabit Ethernet, first proposed in 2008 and ratified in June 2010. Different types of optical fibers are used depending on the functioning of the cable (single mode or multimode), and on the targeted communication distance.

Moore’s Law states that the number of transistors available at low cost on ICs doubles every two years. This trend has traditionally been satisfied by a reduction of transistor size, which in turn enabled a continuous increase of processors’ clock frequencies. But as current technologies reach their limits, transistor density and clock frequencies are not increasing at the same rate as before. More recent efforts consist in increasing computing power by moving to multicore architectures. In high-performance distributed computing applications, where the nowadays common 2, 4 or 8 core computers are not enough, several computing nodes can be used, often consisting of different computers. These nodes are not part of the same machine, and thus one critical element of the overall computing system is the interconnect between the different nodes. Ethernet is one standard that can be used to connect different nodes together. InfiniBand is another popular example of an industry-standard high-performance interconnect that aims at minimizing latency and maximizing bandwidth.

The evolution in bandwidth capabilities of these interconnects must also be accompanied by Input/Output (I/O) improvements between the interconnect and the computer's memory: This is achieved through standards such as the Peripheral Component Interconnect express (PCIe). This industry standard has known several evolutions since it was first introduced as PCI in 1993. The different generations kept improving on bandwidth capabilities to meet the increasing I/O performance demands of various applications. PCIe 2.0 is used in this work, capable of achieving bandwidths of 5.0Gbit/s per lane. A published study [27] evaluates the performance of Quad Data Rate (QDR) InfiniBand with PCIe 2.0 and stresses the importance of such interconnects in the area of high-performance computing.

1.1.2 Motivation for the presence of an FPGA in the link

In the work described in this thesis, two nodes are concerned, and a high bandwidth is to be achieved between them. The optical fiber interconnect used is based on the 1000base-X family (which can be used for 1 Gigabit Ethernet implementations). The optical fiber channels are linked to an interface that is situated in an FPGA chip. In this design, the channels are controlled by transceivers running at a bandwidth of 6.25Gbit/s. An Altera Stratix IV FPGA is used, and although it supports 10-gigabit Ethernet, as well as a number of other high-speed interface standards, including HyperTransport and RapidIO, the design presented in this thesis is more customized.

In advanced environments where CPU cycles become a scarce resource, having an FPGA seeing the data passing through two nodes can significantly lower the load on the CPU, especially for common FPGA-efficient parallelizable tasks such as error correction, encryption and compression. For example, the high bandwidth and monitoring capabilities of the design suggests uses in gate-level logic simulations that see a high number of state changes, and that are a crucial element of today's electronics design, test and commercialization cycles.

A similar FPGA and optical fiber based data acquisition system has been implemented and presented in [3] and [4]. The architecture and communication protocol are based on a master-slave system where the PCIe card is a master controlling a scalable number of front-end cards (the slaves). The PCIe links a computer to the FPGA board that is also equipped with fiber optic links to the front-end cards. The authors report stability with data rates of 1.6Gbit/s. These high data rates are to be achieved to support the development of a new accelerator Facility for Antiproton and Ion Research (FAIR) in Germany. The performance of their PCIe link is roughly the same as in the work presented in this thesis, only twice slower because they use the first generation of PCIe, whereas Gen2 is used here. Their design also demonstrates possible uses of an FPGA in communication links, since it is used for various tasks such as pulse shape analysis.

1.1.3 The role of the PCIe interface

PCIe is a standard port that gets integrated in modern, high-performance multicore configurations. The standardization and wide use of PCIe are important: they suggest that the design described in this thesis can easily be integrated in various configurations since the link between the interface and the computing node only consists of the standard PCIe 2.0 interconnect. For example, PCIe 2.0 is integrated in the recent fourth generation UltraSPARC T3 SoC processor [28]. In fact, the PCI standard's transparency to the implemented PCI software has been an important element influencing its wide-scale adoption in industry. As another example, the PCIe interface is also a critical component in IBM's supercomputer designs. This is illustrated in [30], where a customization of the PCIe interfaces' channel variables improves the performance of their Roadrunner petaflop supercomputer without modifying the overall system architecture.

The implementation described here takes the standard approach of using PCIe as a link between the host and an I/O subsystem, accompanied by a custom optical fiber link for host-to-host communication. Indeed, PCIe is not deemed efficient for directly linking two hosts and a dedicated cluster is usually used in the interface, typically consisting of an InfiniBand or an Ethernet link. Other approaches to creating such interfaces have been explored such as the Dolphin Express [29], which tries to address the shortcomings of PCIe when directly used for host-to-host communication, and compares the potential performance of a PCIe-only link versus a 10GigE-based link. Even though [29] uses a first generation PCIe implementation, it achieves a better performance than 10GigE. In the work described here, PCIe 2.0 is used, thus achieving double the bandwidth for the same number of lanes. The performance is close to that of [29] and the differences can be attributed to the processor and memory speeds of the computers used.

1.2 System setup

The design consists of two computers connected to Altera's Development and Education board 4 (DE4) through a PCIe Gen2 link with x4 lanes. The DE4 is equipped with a Stratix IV FPGA that temporarily stores the data being transmitted in its internal memory. A High Speed Mezzanine Connector (HSMC) connects the DE4 to a daughter card with 8 Small Form-factor Pluggable (SFP) slots. Four of them are used to connect four two-way optical fiber cables. Figure 1.1 shows a block diagram of the configuration.

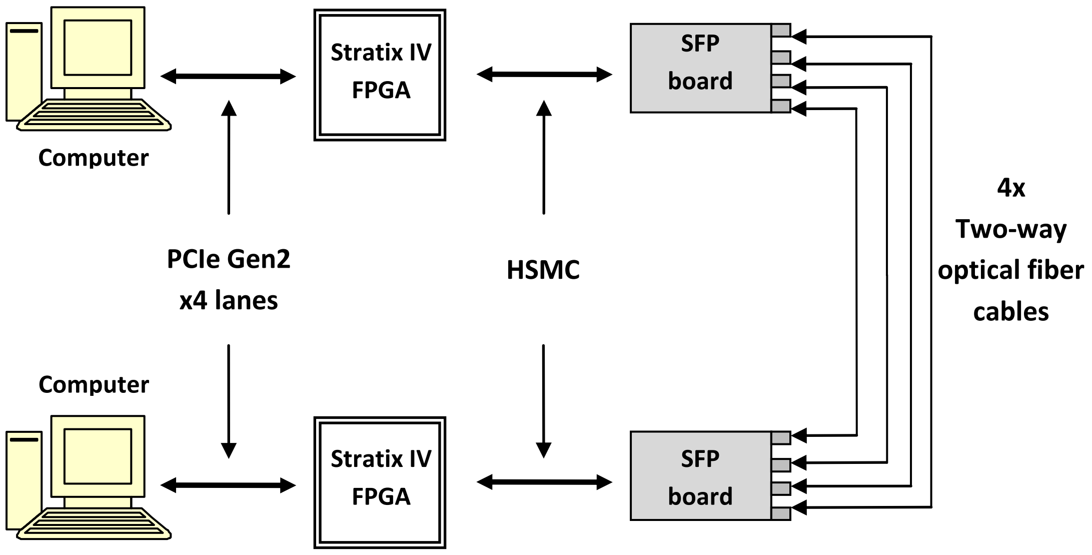


Figure 1.1: High level view of the hardware setup

For the PCIe link, the theoretical achievable data rate without overhead is 4Gbit/s per lane. For the optical fiber, it is 5Gbit/s per channel. Using 4 PCIe lanes and 4 optical fiber cables thus brings the two links up to 16Gbit/s and 20Gbit/s respectively. Also, the optical fiber link experiences more overhead due to its interface with software, and it can be expected to be the ultimate bottleneck of the design, thus controlling the overall system's performance.

The work started from Altera's PCIe Hard IP (Intellectual Property) block. It was interfaced with the *altpciechdma* (ALTera PCIE CHaining Direct Memory Access) driver coded by Leon Woestenberg and Nickolas Heppermann, and available in the Linux kernel. The other main piece of the design is the optical fiber interface that was based on the source code of the loopback demonstration provided by Terasic for its SFP HSMC daughter card. The driver and the PCIe interfaces both required important changes in order to allow for faster speed, more flexible communication with the FPGA's internal memory and more convenient use at the software level. The optical communication logic was modified to include 8b/10b encoding, internally controllable reset, channel bonding, channel alignment and start/end of transmission. It was also modified to support communication in a real setting between two nodes instead of a loopback test only, by introducing a custom synchronization mechanism between the two nodes. The system also required the implementation of an interface between the PCIe logic, the optical communication logic and the FPGA's internal memory. Those modifications are described in more detail in section III.

1.3 Outline

Section II presents background information on the major elements of the system. It provides a perspective on the development of FPGAs, the PCIe protocol and optical fibers, together with outlining their basic functioning. Section III describes the design in three main steps. After giving a high level view of the system, the PC/FPGA interface is presented, including the PCIe and driver designs. Then, the optical fiber interface linking the two nodes is explained. Finally, the link between the two previous interfaces is explained. Section IV summarises and analyzes the experimental results. A conclusion of the work and potential future improvements are found in Section V.

Chapter 2

Background

2.1 Field Programmable Gate Arrays

2.1.1 History of FPGAs

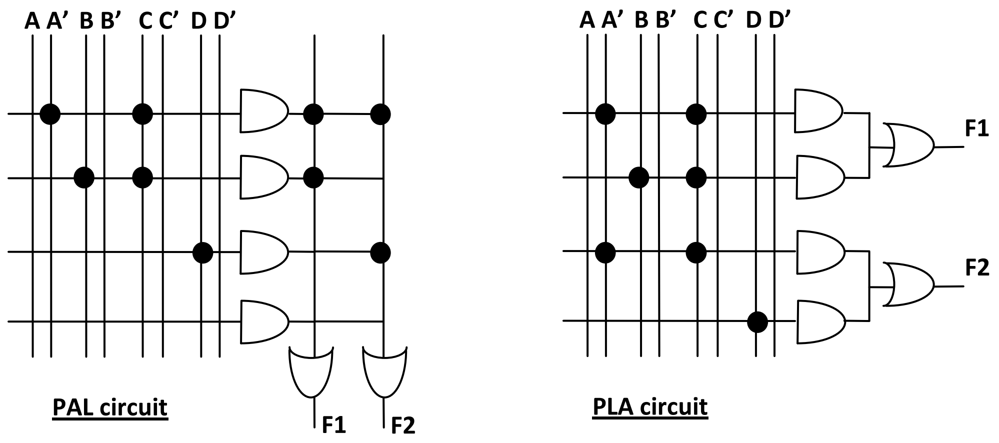
The transistor was invented at Bell Telephone Laboratories in 1947 by John Bardeen and Walter Brattain [6]. It is mainly used for the amplification and switching of electronic signals, and its ability to scale down in size and up in quantity has since then been crucial in it being the technology of choice for developing electronics components.

An Integrated Circuit (IC) uses the properties of semi-conductor materials, mainly silicon because it is inexpensive and widely available, to create devices with transistor properties, mainly MOSFETs (Metal-Oxide-Semiconductor Field-Effect Transistors). Compared to discrete transistors, the IC ones are smaller and quicker. Furthermore, the ability to reliably mass produce IC chips with more and more transistors ensured lower costs to the user, and propelled the electronics industry in a new era. For example, the microprocessor transistor count increased from about 2300 in 1971 (Intel 4004, 10µm transistor technology) to about 42 million in 2000 (Intel Pentium 4, 0.18µm technology). In 2006, it was common to find chips with 1 million transistors per mm^2, with state-of-the-art technologies enabling the production of chips containing a total of approximately 1.7 billion transistors (Intel Dual Core Itanium 2, 90nm technology) [7]. One major development that permitted the exponential growth of the number of transistors per chip is the advent of Very-Large Scale Integration (VLSI) techniques in the late 1970's. Together with Computer-Aided Design (CAD) tools, VLSI introduced more systematic design approaches that accelerated and facilitated the design of more complex systems. CAD tools permit design simulation, verification, layout generation and design synthesis at the different complexity levels [8].

Nowadays, ICs can be "full-custom" designs, where the transistors and their interconnection layouts are directly specified by the technicians or engineers. This introduces high labour, testing and time costs, but optimizes the utilized area, consumed power and operational speed of the system. This approach is only viable for products with very high sale volumes, such as Intel's microprocessors.

Several alternatives to full-custom design exist. Application Specific Integrated Circuits (ASICs) are often described at a hardware description language (HDL) level, and implemented using standard cell libraries. This approach trades off performance for lower cost and faster time to market. Another approach is to use a *gate array* technology: prefabricated silicon chip circuits containing transistors and basic logic gates, with no particular connection between them. The designer then lays out the appropriate interconnections to make the gate array perform a more specific function.

Introduced to target a higher number of potential customers and reduce production costs, programmable logic devices (PLDs) illustrate another way of implementing electronic circuits, where the device’s function is undefined at the time of manufacture, and the user has to program the device to make it perform a specific function. One early example is Texas Instrument’s Programmable Logic Array (PLA), where the programming consisted of altering the metal layer of the IC. The output of PLAs results from connections between an AND array followed by an OR array. As an example, basic logic functions in sum of products form could be implemented. Programmable Array Logic (PAL) devices were later introduced, removing the need of the OR array and resulting in faster designs. Figure 2.1 shows the structures of PLAs and PALs.



Both circuits realize the same functions F1 and F2: F1 = A’C + BC F2 = A’C + D

Figure 2.1, PAL and PLA structures

Complex Programmable Logic Devices (CPLDs) are based on a PAL architecture, but are much bigger in size and are used for larger, more complex designs. They can be reprogrammed several times and often integrate additional features such as bi-directional outputs, feedback lines and storage devices (flip-flops).

Extending on the previous ideas, Field Programmable Gate Arrays (FPGAs) have been invented. An FPGA contains programmable logic components and reconfigurable interconnects, and contrary to the previously mentioned technologies that use arrays of logic gates, FPGAs are based on reprogrammable Look-Up Tables (LUTs). FPGAs stress out the trade-offs discussed earlier to a higher extreme: Although the performance of an FPGA is lower than with full-custom or standard cell library approaches, its much lower non-recurring engineering cost makes it the technology of choice for many applications. FPGAs are faster than software and more flexible than hardware: they sit between these two paradigms, combine some of the advantages of each and thus suit many purposes and applications.

2.1.2 Functioning of FPGAs

FPGAs contain programmable logic blocks and reconfigurable interconnects to combine them together. Other enhancements such as multipliers and memory blocks are also integrated into FPGAs. Large FPGAs with all of these features enable the implementation of complex digital systems. Furthermore, analog features can be embedded directly on FPGA chips, such as differential comparators on input pins to protect against electromagnetic interference and potential ground offsets when differential signaling is used. Logic blocks often consist of 3-LUT or 4-LUT components [12]. An n-LUT device takes an n-bit binary input and produces a 1–bit output out of a predefined set of 2^n stored values, one corresponding to each input pattern. In a simplified view, a logic block consists of an LUT, a memory element (typically a D flip-flop), and a multiplexer that chooses which of the registered or unregistered outputs of the LUT are to be used as the output of the logic block. This logic block structure for a 4-LUT design is shown in figure 2.2.

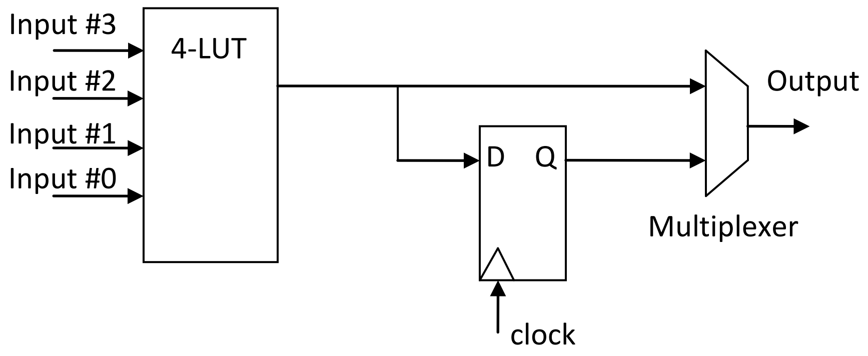


Figure 2.2: Simplified logic block structure of an FPGA

Figure 2.3 shows how 3-LUTs (each associated with a D flip-flop) and programmable interconnects can be laid out on an FPGA chip. As shown, the programmable interconnects account for most of the area consumption.

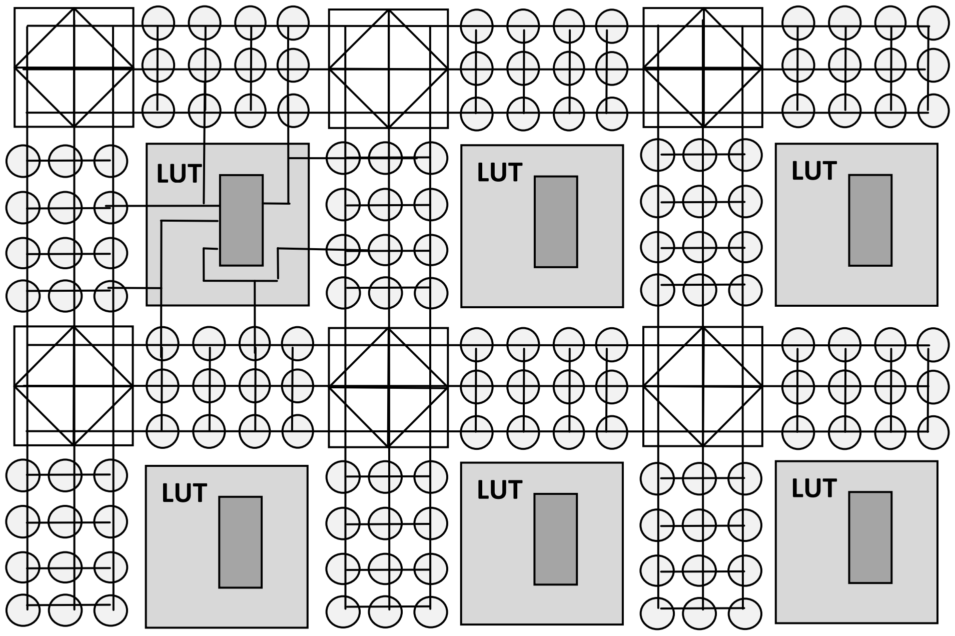


Figure 2.3: FPGA logic block and routing channels structure

In this scheme, the logic blocks are the LUTs, and the routing channel deals with groups of 3 wires. On the sides of the LUTs, some connections are made while other ones are not. Then, when a group of three horizontal lines intersect with a group of three vertical ones, a switch box is used to determine the appropriate connections. As illustrated in figure 2.4, a switch box consists of three programmable switches that can connect a wire entering a switch to the three other wires also entering it. With this structure, wires in channel #1 can only be connected to other wires of channel #1, channel #2 can only be connected to channel #2, and so on. The FPGA basically consists of a large array of such a structure, with input/output (I/O) pads placed around the rectangular layout, so that the appropriate signals that need to interface with external hardware are routed all the way to the edges.

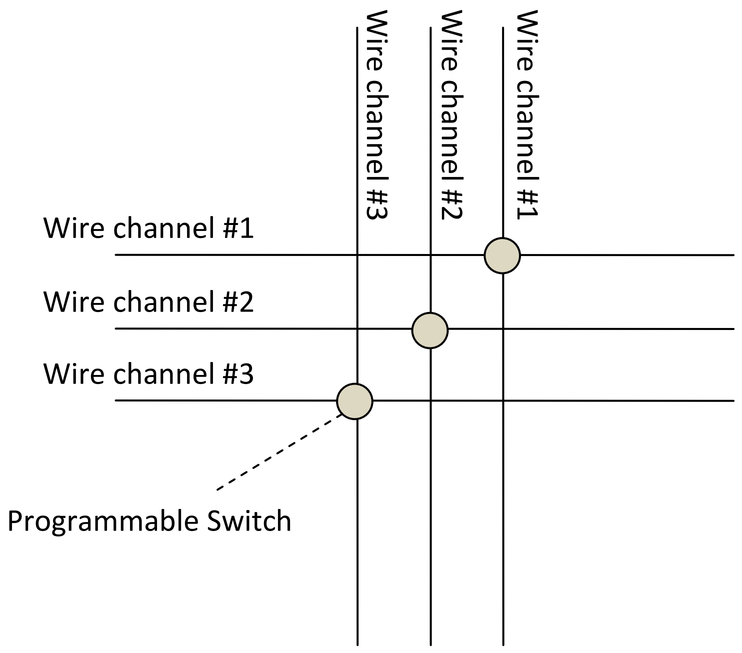


Figure 2.4: Switch box structure

In order to perform certain common tasks more efficiently, many common Digital Signal Processing (DSP) blocks, and even complete microprocessors are embedded as hardwired cores in modern FPGAs. Just like CPUs, FPGAs are meant to be programmed to execute a particular task of interest. Although an FPGA typically runs at a relatively lower clock rate, on the order of hundreds of MHz compared to several GHz for CPUs, the inherent parallelism of its resources makes it a candidate of choice for certain high performance computing tasks that it can execute much faster than a CPU. These include convolution, filtering or Fast Fourier Transform (FFT) operations. The low clock rate and the direct low level implementation of the functions on the data also allows for less power consumption. FPGAs are for example suitable in applications such as code breaking, since encryption/decryption algorithms can be implemented faster on an

FPGA than a CPU, and because the input data is inherently highly parallelizable in a brute-force attack. One such attack has been implemented with *Copacobana* [14] on 56-bit Data Encryption Standard (DES). The same FPGA-based infrastructure has also been used in different applications such as bioinformatics [15].

FPGAs perform operations by spatially organizing the primitive operations, whereas CPUs perform operations by sequentially organizing them in time and storing intermediate results in a limited number of registers. Although the FPGA may then end up with a few cycles of latency, the pipelined structure ultimately permits a faster processing compared to the CPU that takes more cycles for the same operation. This spatial organization allows for less instruction overhead and more active computations on the same chip area. Together with the inherent ability of FPGAs to work directly at the bit level due to their low-level hardware implementation, as opposed to CPUs that only access words then waste cycles on isolating the desired bits, this allows FPGAs to perform many tasks an order of magnitude faster than a CPU counterpart.

2.2 Peripheral Component Interconnect Express

2.2.1 History of PCIe

Peripheral Component Interconnect Express (PCIe), is a computer extension card standard, designed as an improvement over the PCI and PCI-X (PCI-eXtended) standards [17]. Although backwards compatible with both of them, it significantly differs in its functioning. It is a serial, packet-based, point-to-point link between two devices. PCIe itself has known three generations so far, each one improving on the previous one's bandwidth capability. Figure 2.5 summarizes the achievable bandwidths for the standards mentioned, as well as the older Accelerated Graphics Port (AGP) standard.

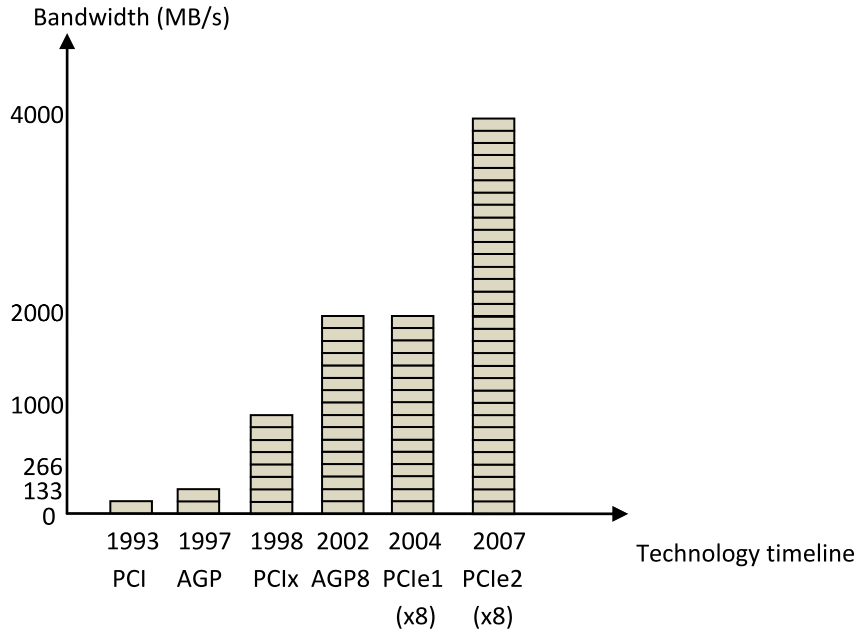


Figure 2.5: Evolution of PCI-like standards

Note that PCIe is capable of transmitting data in full duplex mode, and since the data in figure 2.5 is quoted per each direction, the PCIe values mentioned could effectively be doubled with respect to the other ones.

PCI has been created by Intel. PCIe was then also developed by Dell, IBM and HP. The PCIe standard is being developed and maintained by the PCI-Special Interest Group (PCI-SIG), regrouping many companies. Each new generation saw a double increase in bandwidth. In Gen2, this was mainly due to the doubling of the base clock rate with respect to Gen1: 5GHz instead of 2.5GHz. For the most recent Gen3, the 8b/10b encoding used in Gen2 and explained in section 3.3.1 from the optical fiber interface perspective, is being replaced by a more advanced “scrambling” technique, and uses 128b/130b encoding, thus reducing the encoding overhead by a factor of 13.

Today, PCIe is widely used as a motherboard-level interconnect and as an expansion card interface for custom hardware boards linked to the computer’s motherboard.

2.2.2 Functioning of PCIe

In conventional PCI, all the devices share the same 32 or 64-bit parallel bus. In PCIe, the link is based on couples of dedicated, unidirectional, serial, point-to-point connections, called lanes. A four-lane structure is illustrated in figure 2.6.

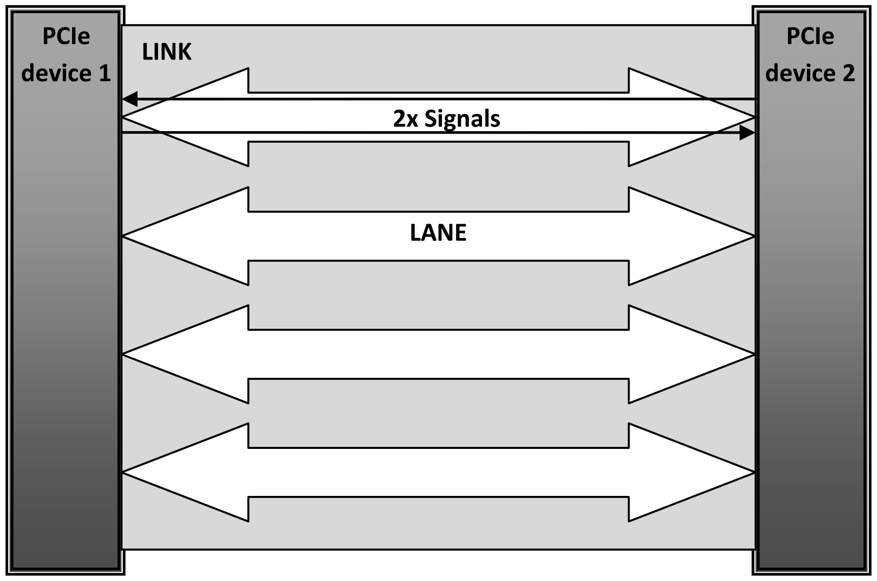


Figure 2.6: PCIe link structure

PCIe connectors are composed of a number of double-sided pins, used to transmit signals such as a 12V power supply, a reference clock and Joint Test Action Group (JTAG) signals. Each lane is assigned two of those pins to transmit data, one in each direction, hence the dedicated connection. Contrary to the conventional PCI case, there is no dedicated interrupt line, and together with control messages, interrupt requests are transmitted in place of the data through a Message Signaled Interrupt (MSI) mechanism. Just like in most high-speed communication systems, the clock information is embedded in the transmitted signal. At the receiver end, in order to accurately retrieve this information despite electric coupling along the communication link, the data first has to be encoded using a line code. PCIe Gen2 uses 8b/10b [18], which produces a 20% overhead. Therefore, although the speed is quoted at 5.0Gbit/s, the effective data rate is limited to 4.0Gbit/s per lane.

In order to limit the noise picked up throughout the communication channel, PCIe uses differential signaling, with a transmitter differential peak voltage of V_{diff} = 0.4 – 0.6V, and a common mode voltage of V_{cm} = 0 – 3.6V.

Within the Open Systems Interconnection (OSI) model, PCIe specifies three layers: The transaction layer, the data link layer and the physical layer, as illustrated in figure 2.7.

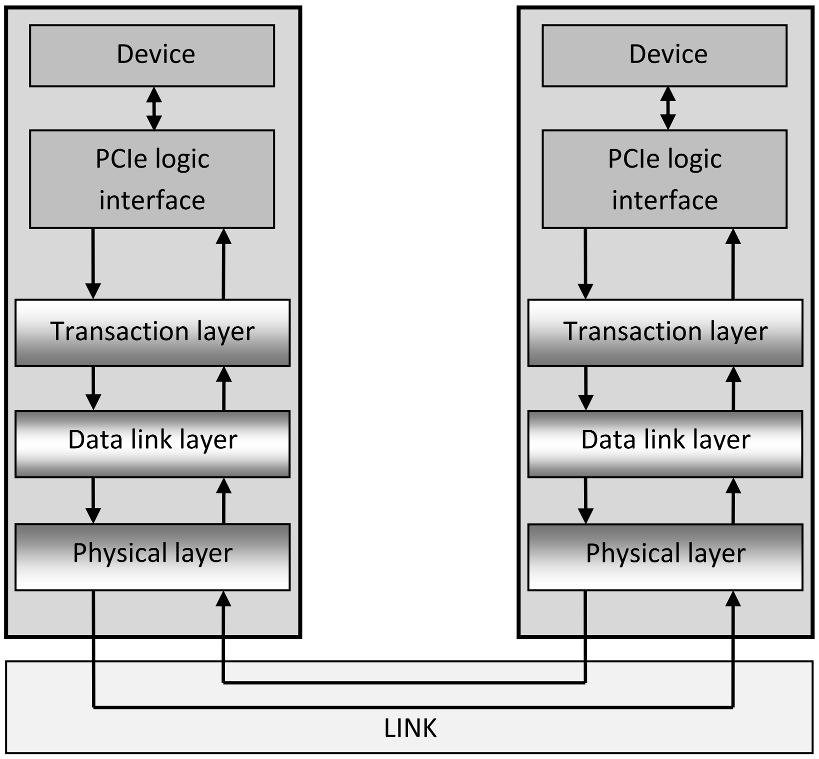


Figure 2.7: Structure of the PCIe transaction layers

Physical layer

The physical layer (PHY) specification is divided into an electrical and a logical part, which defines analog circuitry definitions, such as the serializer/deserializer (SerDes). At the electrical level, each double-sided pin transmits a differential pair of data at

5.0Gbit/s in one direction (for Gen2). Two pins form one lane, which can simultaneously transmit data in both directions at 5.0Gbit/s. Devices can have a number of lanes of x1, x2, x4, x8, x16, or x32. A larger slot is allowed to accommodate a smaller device, thus permitting a higher compatibility between different hardware components. A PCIe device with one lane uses 18 double-sided pins. A PCIe x4 device, such as the one implemented in this design, uses 32 of them.

Transaction layer

In PCIe, a transaction request does not need to be directly followed by its response, so that during the time the target device processes the received information and replies, other data packets can be sent. The transaction layer generates Transaction Layer Packets (TLPs) that are then processed by the Data link layer. Because of the physical 5.0Gbit/s dedicated pin connection, PCIe2 Gen2 is quoted to have a speed of 5.0Gbit/s per lane. With encoding overhead, this figure drops to 4.0Gbit/s for the actual data bytes that can be transmitted in the link. But this does not all consist of useful data. Depending on the higher-level protocol and software overheads, some of it will be wasted. For example, Cyclic Redundancy Check (CRC) and packet acknowledgement introduces an important overhead that further downgrades the speed capability of the protocol. The transaction layer supports four transaction types. **Memory Read or Write** transactions transfer data from or to a memory mapped location. **Input/Output (I/O)**

Read or Write transactions transfer data from or to an I/O location. **Configuration Read or Write** transactions fetch device capabilities and check the status of the PCIe configuration space. **Messages** are used for event signaling and general purpose messaging.

Data link layer

The data link layer sequences the TLPs generated by the transaction layer. Their reliable delivery is ensured with an acknowledgement system that requires sending unacknowledged TLPs again until they are received. A sequence number and a 32-bit CRC code are included in each TLP to ensure error-free transmission, or at least considerably decrease the probability of an undetected error. At the receiver side, packets that fail the CRC check or contain an inconsistent sequence number produce a negative-acknowledge (NAK) signal, indicating to the transmitter that the TLP has to be sent again.

The structure of a TLP is shown in figure 2.8, where DW is a double word (32 bits). ECRC is the End-to-end CRC generated by the transaction layer and LCRC is the Local CRC appended by the data link layer. SOF is the Start of Frame and EOF is the End of Frame, both appended by the physical layer.

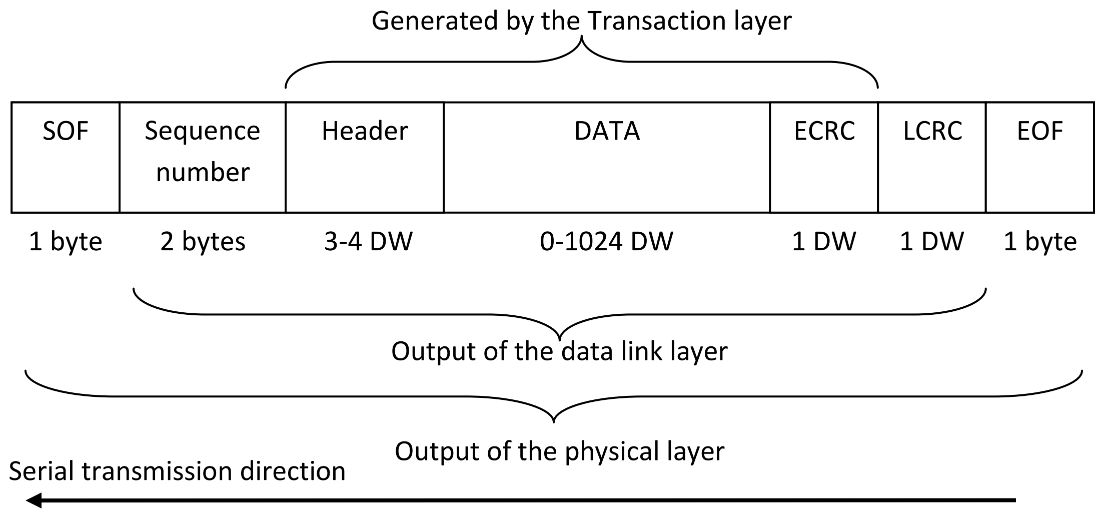

Generated by the Transaction layer
SOF
Sequence number
Header
DATA
ECRC
LCRC
EOF
1 byte
2 bytes
3-4 DW
0-1024 DW
1 DW
1 DW
1 byte
Output of the data link layer
Output of the physical layer
Serial transmission direction

Figure 2.8: TLP Structure in the PCIe protocol

2.3 Optical Fibers

2.3.1 History of optical fibers

Optical fibers are based on the transmission of a signal in form of light, guiding it by repeated reflections. This was first demonstrated in the 1840s by Daniel Colladon and Jacques Babinet. After a number of developments, optical fiber telecommunications were ultimately triggered in the 1970s, when Robert D. Maurer, Donald Keck, Peter C. Schultz, and Frank Zimar demonstrated optical fibers with attenuation below 20dB/km, based on doped silica glass [13]. Optical fibers have since then become an integral part of communication systems. Widely used, from linking two computers locally to transoceanic links, they have evolved into a major industry. Nowadays, optical fibers are capable of achieving faster bandwidths than copper cables, as well as less attenuation, cross-talk and electrical interference [1]. Although their price has long been their main drawback, they tend to slowly replace copper links. This is the case, for example, in LAN Ethernet connections. In fact, optical fibers have long been better suited and more widely used in long-distance communication, because the low attenuation rate allows for fewer repeaters to be used.

Several research efforts have led into optical fibers becoming serious competitors to copper based equivalents in many applications. For example, for relatively short interconnects such as the one discussed in this thesis, bandwidth can be scaled efficiently using fiber ribbons, first introduced in 1975 by AccuRibbon. Ribbons increase the achievable bandwidth by increasing the number of cables used in parallel while maintaining a small area usage. This finds applications in computer data centers, equipment connections and outside plant cables. Also inspired by copper cable technologies, active optical fiber cables have been developed: the cable's performance is increased by inserting a chip that can deal with issues such as attenuation, crosstalk and group velocity distortion. This improves the bandwidth and allows for less reliable materials and processes to be used, thus decreasing the cable's cost. Finally, optical fiber transceivers are also being integrated in a Surface Mount Technology (SMT) manner on Printed Circuit Boards (PCBs), allowing for smaller components to be used, lower production costs and a better automation of the manufacturing process.

2.3.2 Functioning of optical fibers

Materials have a characteristic refractive index n = c/v, where c is the velocity of light in vacuum (about 300,000 km/s), and v is the velocity of light in the material. Light being the fastest in vacuum, n assumes values greater than 1. According to Snell's law, the angles of incidence and refraction of a light wave hitting a boundary between two materials with different refractive indices n_1 and n_2 is governed by $\frac{sin\Theta_1}{sin\Theta_2} = \frac{n_2}{n_1}$. Therefore, if the incidence angle is greater than a critical angle $\Theta_c = Asin(n_2/n_1)$, where light goes from a material with refractive index n_1 to one with refractive index n_2, and n_1 > n_2, then total internal reflection occurs: The wave continues its propagation without loss. Optical fibers are typically made of a long and thin high refractive index core, surrounded by a lower refractive index cladding. Typical values for the refractive indices are $n_1 = 1.62$ and $n_2 = 1.52$. A coating surrounding this structure is also used in order to protect the optical fiber from its environment, thereby increasing its strength, reliability and lifetime. Optical fibers transmit information by guiding light in their core, through a series of total internal reflections at the boundary between the core and the cladding, made possible by the contrast in refractive indices. This is illustrated in figure 2.9.

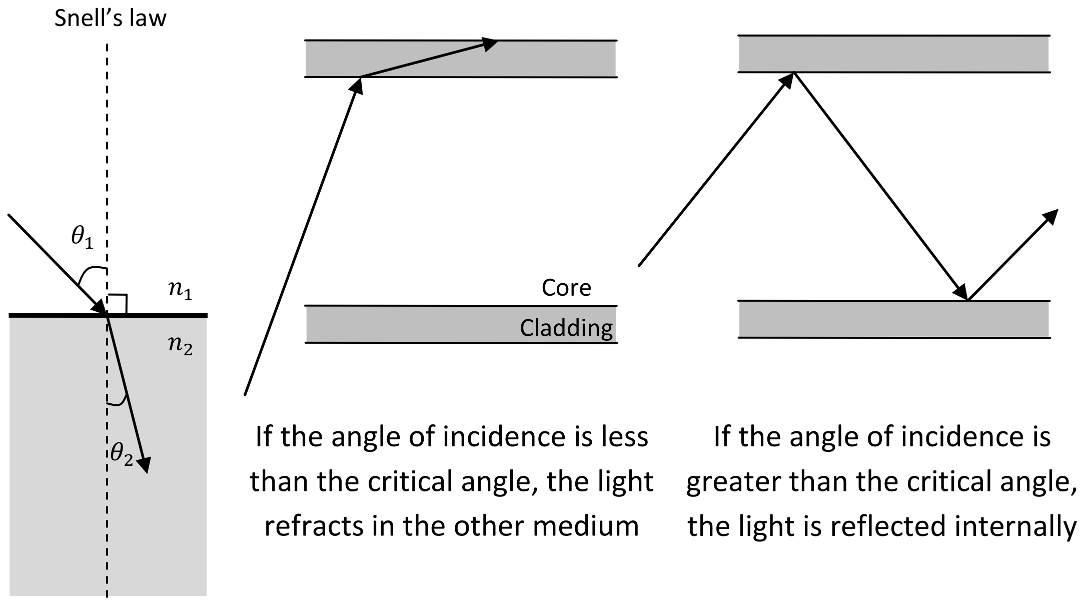

Snell's law
θ_1
n_1
n_2
θ_2
Core
Cladding
If the angle of incidence is less than the critical angle, the light refracts in the other medium
If the angle of incidence is greater than the critical angle, the light is reflected internally

Figure 2.9: The principle of total internal reflection in optical fibers

A typical optical fiber link consists of a transmitter and a receiver, an optical fiber cable, as well as a connector on each side. Pulse distortion results due to non-ideal characteristics of those components, which affects overall system performance [2]. Typically, the digital data is encoded with a line code such as Manchester or 8b/10b, and is transmitted as a stream of 1s and 0s. This produces a square-wave type of signal. But due to pulse distortion throughout the length of the fiber and at the connections between the different components of the link, the pulse broadens and may eventually send the wrong information, as illustrated in figure 2.10.

Figure 2.10 only illustrates the signal levels detected at the receiver end, which do not directly translate into data bits. In particular, since the application presented in this work uses a Non-Return-to-Zero (NRZ) scheme as shown, line codes such as 8b/10b encoding are further used in order to embed clock information and to provide a DC balanced signal, as explained in section 3.3.1. An alternative is to use a Return-to-Zero (RZ) scheme, where the signal returns to zero between each pulse. In this case, a positive signal indicates a logical 1, a negative one indicates a logical 0, and a zero signal is observed after every transmitted bit, thus producing a self-clocking signal. This nevertheless consumes twice the bandwidth of an NRZ scheme, and still does not provide DC balance.

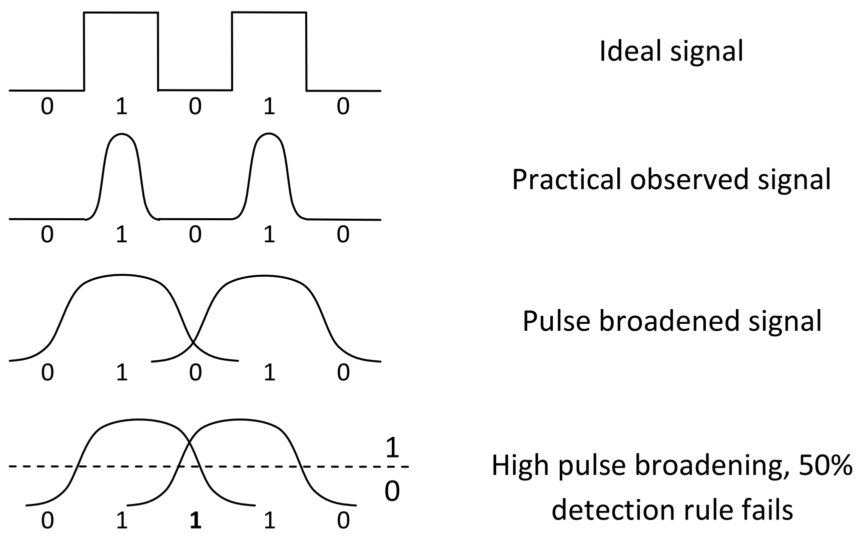

Ideal signal
0 1 0 1 0
Practical observed signal
0 1 0 1 0
Pulse broadened signal
0 1 0 1 0
1
0
High pulse broadening, 50% detection rule fails
0 1 1 1 0

Figure 2.10: Illustrating the effect of pulse broadening at the receiver end

In single mode operation, a coherent light of a single wavelength and mode is guided by the optical fiber. In multimode, multiple modes are excited in the core of the fiber. So in an ideal setting, multimode fibers have unlimited bandwidth, but in practice, distortion occurs and limits the achievable bandwidth. Those distortions are more pronounced in multimode fibers, and since the length of the fiber also affects the bandwidth, single mode fibers are preferred for very long distance transmissions. The work presented here concerns a short interconnect with a multimode fiber that is only 50cm long, thus allowing for very high bandwidths to be achieved. In general, multimode fibers also have larger core sizes, thus simplifying connections and allowing for the use of less expensive electronic components and light sources such as LEDs (Light-Emitting Diodes) and VCSELs (Vertical-Cavity Surface-Emitting Lasers).

Distortion occurs for various reasons. First, an instantaneous rise time of 0 is unachievable in practice. Hence, the ideal square wave signal is already distorted at the transmitter end, before being further distorted by the receiver. As an example, for the SFP components used in this design, maximum rise and fall time values are specified at 90ps [23]. Also, waves propagate with phase velocity $v_p = {}^{\lambda}/_{T}$, where T is the wave's period and λ is its wavelength. But the emitted light is composed of several modes that do not propagate at the same speeds within the fiber and cannot arrive at the receiver in a perfectly consistent temporal order with respect to the time they left the transmitter side. This effect is known as chromatic dispersion and leads to pulse broadening. The other effect that causes distortion of the signal and limits the achievable bandwidth of a multimode fiber is known as modal distortion. This occurs because the different light components are sent through different paths (or modes), which have a different length. Thus, the waves do not all travel through a path of the same distance.

Chapter 3

Design

This chapter aims at explaining the structure and different elements of the design. Section 3.1 presents the general structure of the interface, showing how the logics for PCIe and optical communication interface with the computer and the second board. A description of the internal memory structure of the FPGA is also presented, showing how the different elements of the system access it. Section 3.2 tackles the details of the PCIe link design, including the configuration of the Altera PCIe function, the structure of DMA requests and issues related to the software driver. Section 3.3 is concerned with the optical fiber interface design, with a focus on the implemented 8b/10b encoding scheme and the synchronization mechanism between the two nodes. Finally, section 3.4 explains the link between the two major sub-components of the design: the PCIe logic and the optical communication logic.

Since the design targets an Altera FPGA, the main Computer-Aided Design (CAD) tool used was Altera's Quartus II software. Several related tools were used, including the In-System Memory Content Editor, the SignalTap II Logic Analyzer and the MegaWizard Plug-In Manager to generate parameterized Intellectual Property (IP) modules.

3.1 High level view of the system

3.1.1 High level view of the communication protocol

Overall, the system can be viewed as a PCIe logic communicating with the computer, and an optical fiber interface logic communicating with the other board. They are linked with a custom communication protocol and exchange data through a Random Access Memory (RAM) structure as explained in section 3.1.3. A diagram illustrating this is shown in figure 3.1.

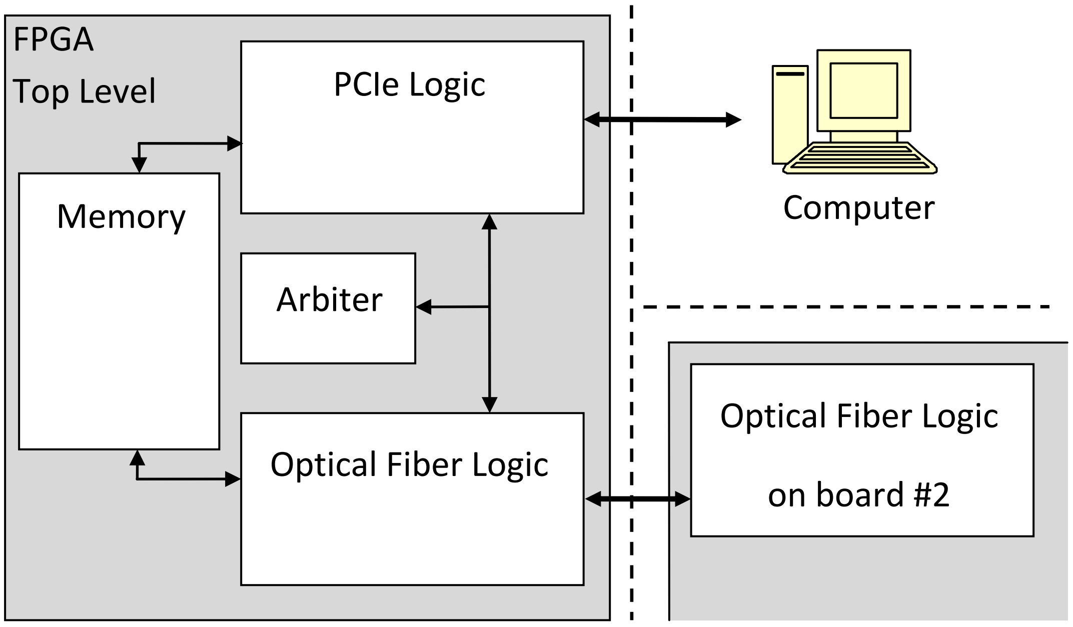


Figure 3.1: High level view of the system structure

3.1.2 High level view of the communication protocol

The two main links that constitute the system are the PCIe interface between the FPGA and the PC, and the optical fiber interface between the two nodes. The design uses x4 lanes for the PCIe. Each lane is quoted at a maximum physical throughput of 5Gbit/s, thus forming a 20Gbit/s link. Each optical fiber module is run at 6.25Gbit/s, thus forming a 25Gbit/s link when 4 channels are used. When protocol, encoding and software overhead are introduced, the available raw bitrate for the payload is reduced. Still, the optical fiber link remains faster, and the PCIe interface can be considered the bottleneck of the design.

Although it is possible to have the optical fiber link wait for PCIe DMA completion, and similarly to have the PCIe DMA wait for optical fiber transfer completion, such an approach is clearly inefficient. This motivates the use of at least two buffers on each node, such that on DMA completion, no cycle is wasted waiting for node to node transfer: The next chunk of data can be transferred to another buffer, while the data that has just been written to the FPGA can be transferred to the other node. Since the optical fiber is faster, it is expected that the PCIe will always be busy processing DMAs, while the optical fiber will not be used to its full capacity, since it will regularly be waiting for DMA completion before it triggers a new transfer. This scheme ensures that the overall speed of the system does not fall below that of the slower component: The PCIe link.

In this design, three buffers are used instead of only two. This reserves more time to eventually process the data, at the expense of requiring more area and increasing the latency. Each buffer is further divided into two subparts. The first subpart has its data written by the optical fiber and read by the PC, while the second one is read by the optical fiber and written by the PC. In hardware, this is stored into two RAM blocks, each one divided into 3 buffers, as illustrated in figure 3.2. The first part of *Buffer1* is stored in the first RAM, while its second part is stored in the second RAM. This is the case for *Buffer2* and *Buffer3* as well.

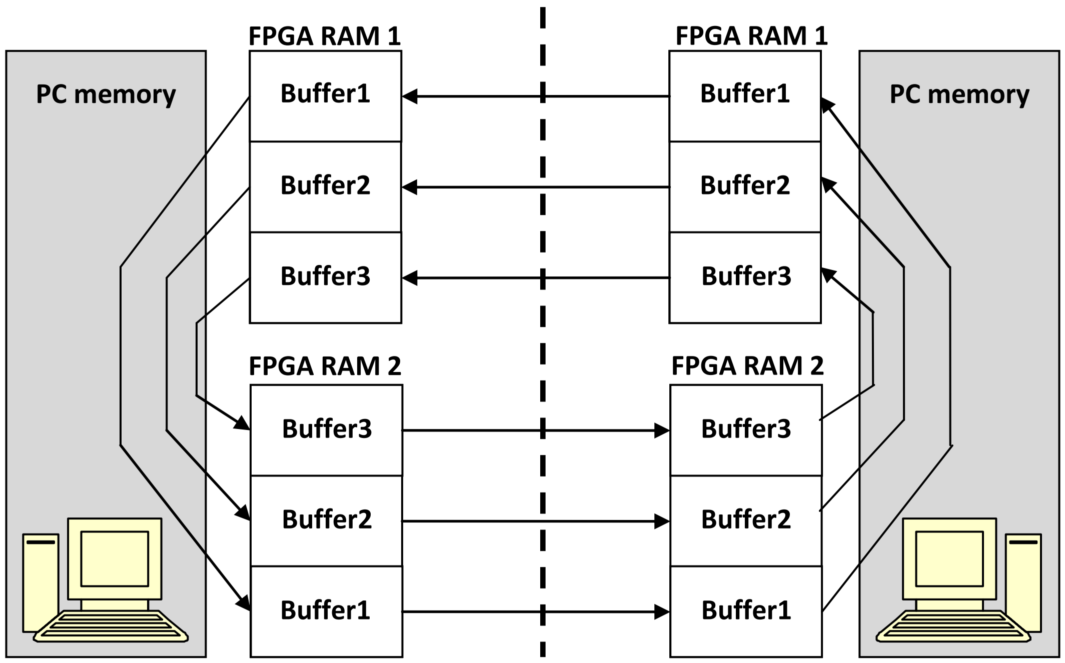


Figure 3.2: High level view of the communication structure

3.1.3 Memory structure

With respect to the PCIe DMA transfers, the FPGA's internal memory is denoted as the Endpoint (EP) memory. Each FPGA needs three memory areas for the three buffers used in the design, so the EP memory could be structured as one RAM divided into three areas. In this design, it was chosen to divide the RAM into four areas in order to keep multiples of two in the structure. The fourth area has predefined pseudo-random values that are transmitted when the optical fiber is in an idle state as mentioned in section 3.3.3, although this is not required and is mainly the result of previous debugging efforts during the implementation of the design. Furthermore, because of the different clock domains, two RAMs have to be used instead of one. The first RAM is written by the optical fiber interface logic at the receiver's clock speed, whereas it is read by the PC at the PCIe clock speed. On the other hand, the second RAM is written by the PC at the PCIe clock speed, and it is read by the optical fiber interface logic at the transmitter's clock speed. The PCIe accesses the data in 64-bit words, whereas the optical fiber interface logic, that uses channels with 128 bits of data, accesses one 128-bit word on every clock cycle. The RAMs therefore need to have different sizes for the input and output data, together with different input and output clocks, which is supported by Altera on-chip RAMs. This is illustrated in figure 3.3.

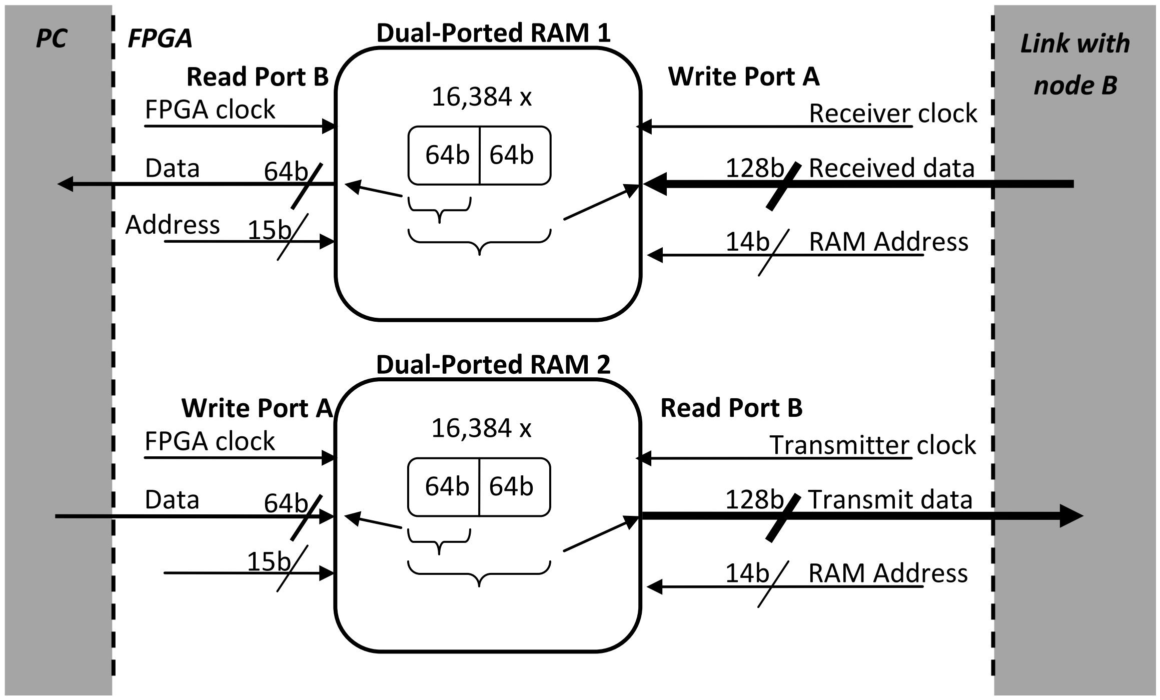


Figure 3.3: Memory structure of the design

When transiting through the FPGA chip, the data is stored in an internal memory RAM. The size of the RAM has to be carefully chosen because there is a trade-off between communication latency and FPGA area versus communication speed. The smaller the RAM, the faster a particular bit will travel from one computer to the other because the DMA chunks will be smaller and processed sooner. Also, smaller RAMs allow for the extra FPGA area to be used for other purposes. On the other hand, this would increase the PCIe protocol overhead since tasks such as fetching the DMA descriptor tables and checking for DMA completion have close to constant duration no matter what the size of the DMA is, so that two DMAs of size N would take more time to be completed than one DMA of size 2N. This tradeoff is further described in section 4.2.

The size of each RAM is chosen to be 2 Mbits, for a total of 4 Mbits of memory on each board. The PCIe logic sees the RAM as a group of 32,768 64-bit words accessible by a 15-bit address, whereas the optical fiber interface logic sees the RAM as a group of 16,384 128-bit words accessible by a 14-bit address.

For debugging purposes, each RAM is also associated with a companion “shadow” memory that is single-ported and that receives as input all the data that is being written to the RAM blocks. So the first shadow memory is written by the optical fiber interface logic at the same time as the first RAM, accessed as 128-bit words, while the second shadow memory is accessed as 64-bit words and is written by the PCIe interface at the same time as the second RAM. Since those shadow memories are single-ported, the second port can be used by the “In-System Memory Content Editor” of Quartus to monitor changes in the RAM through the JTAG port, and thus ease the debugging process by providing direct observation of the FPGA’s internal memory.

3.2 PCIe interface

3.2.1 Configuration of the PCIe interface

The PCIe compiler 11.0 provided with Quartus is used to generate a MegaWizard configurable PCIe design. It implements the transaction, data link and physical layer features of the PCIe specifications described in section 2.2.2. Part of Altera FPGAs, the PCIe hard IP block embeds those features and allows for important resource savings in terms of Logic Element (LE) usage. It also contains embedded memory buffers and consumes less power than an equivalent soft IP implementation.

The parameters can easily be configured with the MegaWizard, and some of them are shown in tables 3.1, 3.2, 3.3, 3.4 and 3.5.

Parameter	**Value**
PHY type	Stratix IV GX
PHY interface	Serial
Lanes	x4
Xcvr ref_clk	100 MHz
Application Interface	Avalon-ST 64-bit
Port type	Native Endpoint
PCI Express version	2
Application clock	250 MHz
Max rate	Gen2 (5.0 Gbit/s)
Test out width	9 bits

Table 3.1: PCIe system settings

BAR	**BAR type**	**BAR Size**
0	32-bit Non-Prefetchable Memory	256 MBytes - 28 bits
2	32-bit Non-Prefetchable Memory	256 KBytes - 18 bits
4	32-bit Non-Prefetchable Memory	256 KBytes - 18 bits

Table 3.2: PCIe base address registers

After the PCIe device has been mapped to the port or memory mapped I/O address space of the system, the software driver programs PCIe Base Address Registers (BARs) to inform the device of its address mapping. BAR0 and BAR4 map to the target memory block on the FPGA while BAR2 maps to the DMA control and status registers.

Register Name	Value
DeviceID	0x0004
VendorID	0x1172
SubsystemID	0x0004
Subsystem Vendor ID	0x1172
RevisionID	0x01
Class Code	0xFF0000

Table 3.3: PCIe Read-Only registers

Parameter	Value
Tags suported	32
Completion timeout range	ABCD
Error reporting	Off
MSI messages requested	4
MSI message 64-bit address capable	On
Link common clock	On
Implement MSI-X	Off

Table 3.4: PCIe capabilities parameters

Parameter	Value
Maximum payload size	512 Kbytes
Number of virtual channels	1
Retry buffer size	2 Kbytes
Maximum retry packets	64

Table 3.5: PCIe buffer setup parameter

3.2.2 Structure of the PCIe interface

The chaining DMA example design also provided by Altera, is used as a starting basis for the work [16]. Whereas a simple DMA can only transfer data that is stored contiguously in memory, a chaining DMA implementation makes use of a descriptor table that can store several memory transfer requests, possibly with non contiguous starting points. Although Altera's example design is used as a starting point for this implementation, the chaining DMA feature is not used to its full potential and data is always assumed to be contiguous.

Figure 3.4 illustrates the structure of the PCIe design. It has a DMA arbiter that contains a Root-Complex (RC) slave module, a DMA read and a DMA write module. Each DMA module has its own DMA descriptor table, so that DMA reads and writes can be performed at the same time, using the full duplex capability of the PCIe protocol. The DMA modules are controlled by a software driver, in turn controlled by a C program running on the computer. RC memory is the system memory (accessible by the PC), Endpoint (EP) memory is the FPGA memory where transactions take place. The EP memory consists of a RAM structure in the FPGA's internal memory. Contrary to the example design, and since this RAM block is also used by the optical fiber interface, it was brought up to the top level in order to integrate the final design more easily. Also, it was divided into two blocks, one for read transactions and one for write transactions, as explained in sections 3.1.2 and 3.1.3. The RC slave module is responsible for downstream requests to the EP memory: It programs the DMA control registers and reads the status registers. The DMA read engine implements operations that transfer data from the RC memory to the EP memory in the FPGA across the PCIe link, whereas the DMA write engine transfers data from the EP memory to the RC memory.

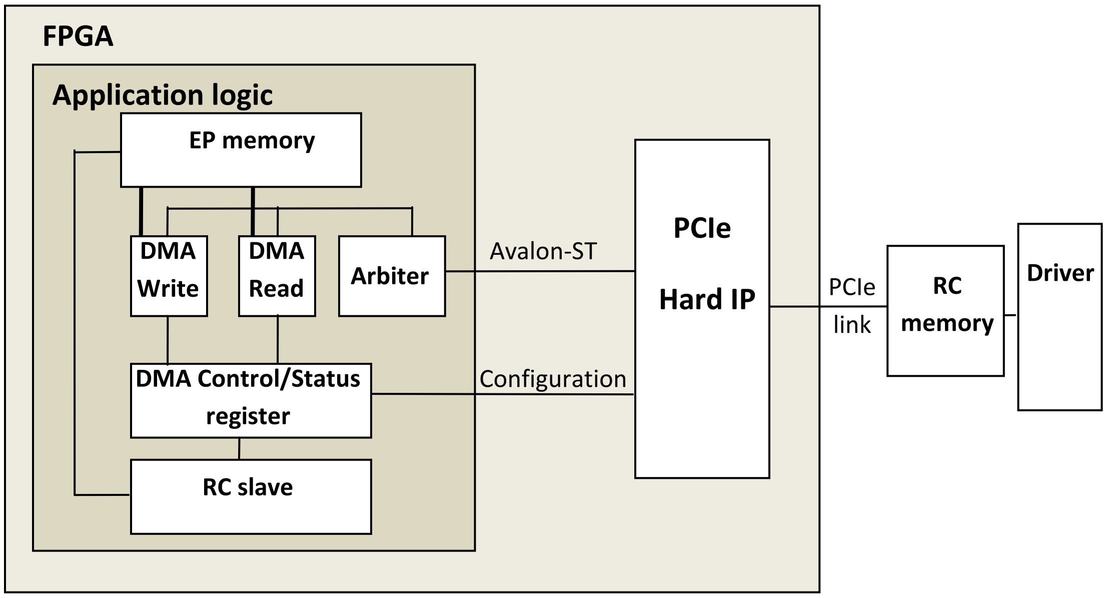


Figure 3.4: PCIe chaining DMA design structure, similar to [16]

The application logic hierarchy inside the FPGA is illustrated in more detail in figure 3.5. As mentioned previously, the EP memory RAM is brought out of the PCIe design itself, such that it later interacts with the optical fiber interface design more easily.

example_app_chaining: This is the top level module implementing Avalon Streaming (Avalon-ST) interfaces and sideband bus logic. Avalon interfaces ease the design of Altera FPGAs, by defining interfaces for streaming high-speed data, reading and writing registers and memory, or controlling off-chip devices. Avalon-ST supports the unidirectional flow of multiplexed streams, packets and Digital Signal Processing (DSP) data [9].

cdma_ast_rx: This module contains the Avalon-ST receive port for the DMA requests, which converts data at the Avalon-ST interface into the corresponding descriptors and data interface later used by lower level modules.

cdma_ast_tx: This module is responsible for implementing the same kind of transactions as the previous one, but for upstream communication: It contains the Avalon-ST transmit port that converts the descriptors and data interface from lower level modules to the corresponding Avalon-ST interface.

cdma_ast_msi: Similarly to the modules implementing the Avalon-ST transmit port; this module is responsible for converting MSI requests from lower level modules into Avalon-ST data.

cdma_app_icm: In Altera's example design, this module instantiates the EP memory of the design. Here, the corresponding connections are propagated to the top level, so that the interface with the optical fiber related logic is simplified. This module also arbitrates PCIe packets for the modules *dma_dt* and *rc_slave*.

rc_slave: This module is responsible for managing downstream requests. It instantiates the *rxtx_downstream_intf* and *reg_access* modules. These requests include the programming of the DMA control registers and reading of DMA status registers.

rxtx_downstream_intf: This module takes care of downstream read and write requests. BARs 0, 1, 4 and 5 are mapped to the EP memory, whereas BARs 2 and 3 access the chaining DMA control and status registers using the **reg_access** module. The latter accesses the control registers in **dma_prog_reg** (programmed by the software driver) and the status registers in **read_dma_requester** and **write_dma_requester**: For each descriptor, these modules transfer data between RC and EP memory by issuing PCIe transaction layer packets.

dma_descriptor: This module retrieves the DMA read or write descriptors written by the software driver and stores them in a First in First Out (FIFO) structure. Its data is then read by *read_dma_requester* and *write_dma_requester.*

dma_dt: This module plays the role of an arbiter that manages the PCIe packets issued by *dma_prg_reg, read_dma_requester, write_dma_requester* and *dma_descriptor*.

flagger: This module is not present in the example design. It is added inside the arbiter in order to monitor changes in the DMA status registers and inform the optical fiber interface hardware when it is allowed to transfer data.

Other modules are also used for tasks such as Cyclic Redundancy Check (CRC) calculation and synchronization, but they do not illustrate the hierarchy of the PCIe communication design and are not shown here.

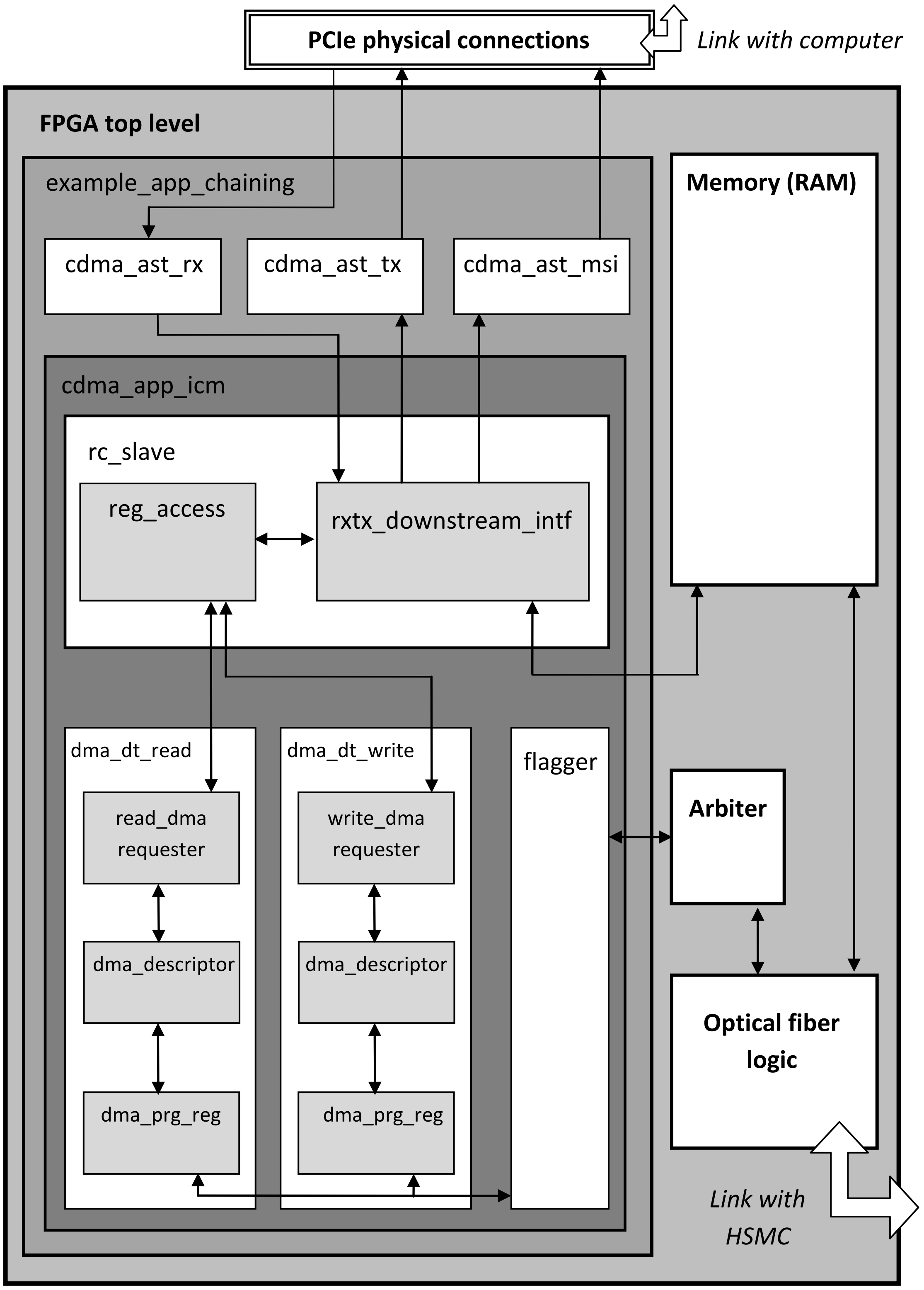


Figure 3.5: Application logic hierarchy of the PCIe design

3.2.3 Direct Memory Access

The descriptor table is a fundamental part of the chaining DMA implementation. It contains information on the length, address of the source and address of the destination of up to 256 transfers for a single DMA request. Since the starting addresses are specified for each entry in the table, the data need not be stored in contiguous memory locations. Each entry also contains control bits that set the handshaking behaviour between the software and the DMA engine [17].

The software driver writes the DMA information into shared memory, from which the DMA read and write modules continuously collect descriptor table information. The driver then programs the control registers with the number of descriptors to be processed, and triggers new DMA requests for read and write. Each descriptor is then processed in turn and the corresponding data transfer is performed. Note that read and write descriptors may be processed simultaneously.

The chaining DMA control registers are mapped into BAR2. At address 0x0, four 32-bit DWORDs constitute the DMA write control registers, whereas address 0x10 contains four 32-bit DWORDs for the DMA read control registers. Two of those 32-bit DWORDs are used to specify the lower and upper parts of the 64-bit base address of the descriptor table in RC memory. 16 bits indicate the number of descriptors to be processed, 16 bits indicate the index of the last descriptor to be processed, 16 bits are used as a control field, and the last 16 bits are unused.

On BAR2 address 0x20 and 0x28, chaining DMA status registers are stored for both the DMA write and the DMA read respectively. They indicate information such as the board number, maximum request size, the number of the last processed descriptor, and whether there are any descriptors left to be processed by the engine.

The 16-bit control fields of the DMA read and write control registers indicate how the DMA engine can inform the software driver of a DMA completion. The two main methods that could be implemented are MSI interrupts or polling of the last processed descriptor. In the first scheme, the driver enters a sleep state after issuing DMA requests. The DMA engine then issues an MSI interrupt and wakes up the driver on completion. In the second scheme, the driver polls the state of the "last descriptor processed" register, and knows that the request has been completed when the register's value is equal to the number of descriptors in the table, since they are processed from index 0 to index n-1, where n is the number of descriptors in the table. Both schemes have been implemented in software, but it was found that the system was faster with polling. Indeed, sending interrupts produces an overhead that can be circumvented by a parameterized polling strategy; experiments have been conducted to compare the two approaches and they are explained in section 4.2.

3.2.4 Interface with software and driver design

The design of a driver that interacts with the FPGA logic properly is a complex task in itself. The driver has been coded in C, and is based on the freely available software coded by Leon Woestenberg and Nickolas Heppermann, *altpciechdma.c*, available in the Linux kernel. This driver organizes the PCIe probing at system startup and implements a DMA test for both reads and writes, where completion is checked with polling. The driver required some major updates to customize the design, make it more efficient and interface it with a C program. Reference information about implementing linux drivers was found in [11].

Drivers are pieces of software that lie in the Linux Kernel space. They allow high level user applications such as a C program to interact with the computer's low level hardware. Drivers can access several built-in kernel functions to communicate with the hardware, and can be accessed by the user application in different ways. The two main types of device drivers in Linux are the *block* and *character* types. Block type devices implement random access to larger units of data for transfers to and from memory and are used for hardware such as hard disks, whereas character devices implement serial I/O transfers in the much smaller unit of a single character. Here the driver is loaded as a module instead of integrating it into the kernel and recompiling it. In Linux, drivers are seen as files from the point of view of the user application, and they can be accessed with such functions as *getchar* or *fread*. The file also needs to be opened beforehand and closed at the end of its use.

The userspace commands *insmod* and *rmmod* serve to load and unload a device driver from user space; they can be directly integrated in a makefile that also compiles the C code of the driver. Those functions make calls to the kernel space functions *module_init* and *module_exit* which are executed at the beginning and end of use of a driver. The latter functions serve various initialization purposes, such as reserving interrupts or memory pages, and then restoring the reserved resources when they are no longer needed by the driver. The driver needs to register itself in kernel space using functions such as *register_chrdev* with a given major number. This major number is then used in *user space* when linking a file in the /dev directory with the *mknod* command, to distinguish between the different drivers. Note that the appropriate *chmod* settings have to be granted to the device file, and the memory access functions should be called with the appropriate write and read rights.

In the case of the PCIe driver, after downloading the FPGA design to the board and restarting the computer, the PCIe device is already detected by the Operating System (OS) and can be observed in the list of PCIe devices by typing the command *lspci*. Then, on module insertion and initialization, a probing function is called in the driver. This function performs several tasks: It allocates memory, allocates and maps coherently-cached memory for the descriptor table of the chaining DMA design, enables the bus master capabilities of the device, enables and queries interrupt lines, sets a 32-bit DMA mask, maps PCIe BARs in memory and initializes the character device capabilities of the driver. The driver also requires the manufacturer and device ID numbers: 0x1172 for Altera, and 0x0004, arbitrarily chosen for this particular design's device number. This needs to match the configuration registers in the MegaWizard when generating the PCIe interface.

Due to the full duplex capabilities of the PCIe interface, DMA reads and writes can be performed at the same time. In this implementation, only *fread* calls are used in the C program to interface with the driver. Half the buffer is used to copy data from the FPGA to user space, whereas the second half of the buffer is previously filled with the data to be written to the board. Opening the device file with *fopen* (with read and right permissions), and calling *fread* in the C program calls a function *DMA_transfer* in the driver that performs several tasks. First, it polls the status of a 3-bit flag sitting in the PCIe's read header. Each bit corresponds to the current state of one of the 3 memory buffers in the FPGA. If the bit at the current index is 0, it proceeds with filling descriptor tables for both a DMA read and a DMA write, and triggers them both, otherwise it waits until the bit switches back to 0 before it proceeds with the DMA transfers. A value of 0 indicates that the exchange of data with the other node through the optical fiber logic has been processed successfully, that new data is ready to be read by the PC, and that the buffer is ready to be filled for future transfers. When a value of 0 is seen and after the DMAs are set up, the driver proceeds with polling for completion as explained in section 3.2.3. On completion, the driver sets the current flag bit to 1 to indicate that data is ready for the optical fiber to transfer it, and it increments the value of the current buffer/flag (legal values are 0, 1 and 2). An alternative, MSI based completion checking system has been implemented as well, as explained in section 3.2.3. In this scenario, the driver switches to a sleep state after triggering the DMAs, and is awakened by an Interrupt Service Routine (ISR), automatically called on DMA completion. The overall scheme is illustrated in figure 3.6.

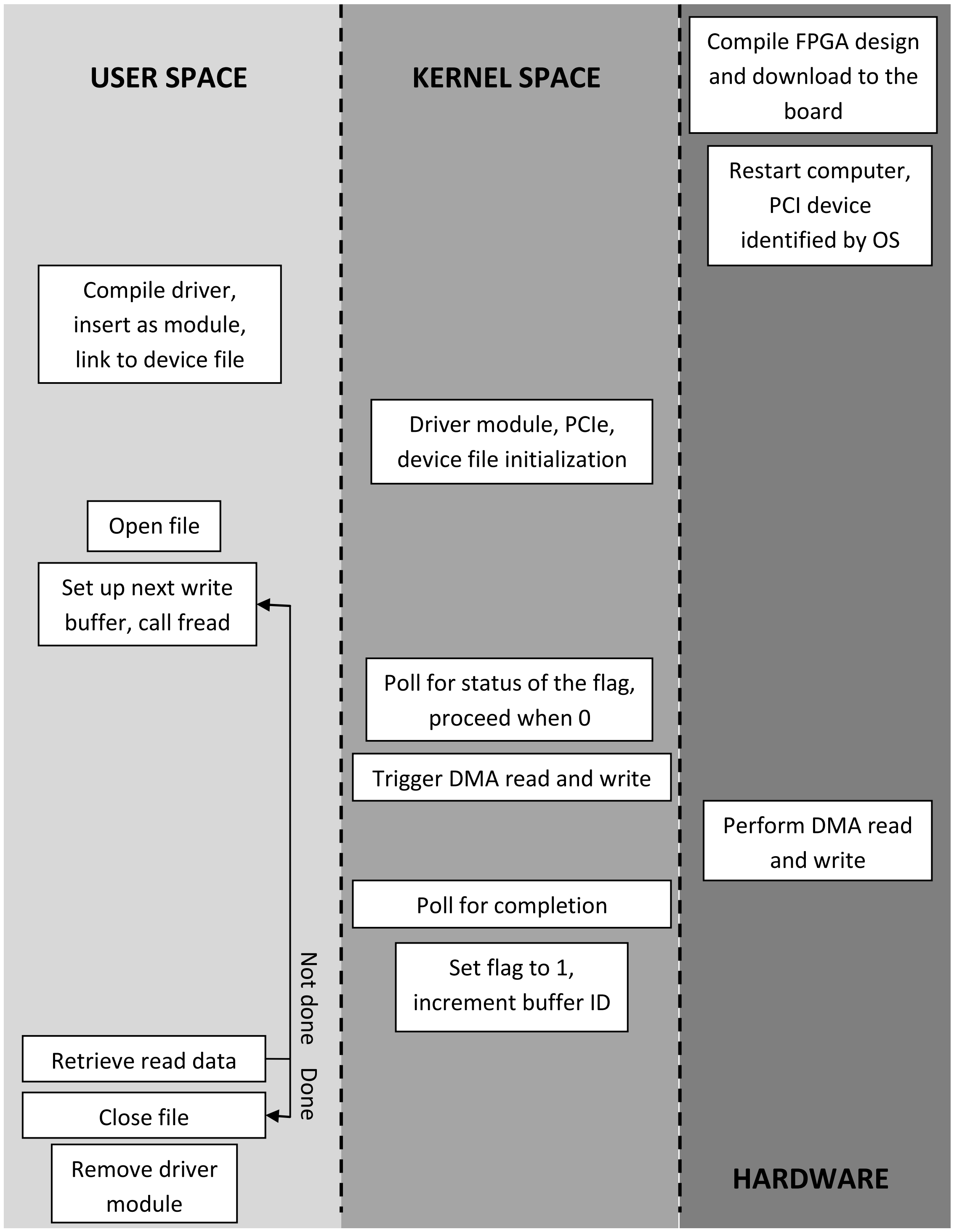

USER SPACE
KERNEL SPACE
HARDWARE
Compile FPGA design and download to the board
Restart computer, PCI device identified by OS
Compile driver, insert as module, link to device file
Driver module, PCIe, device file initialization
Open file
Set up next write buffer, call fread
Poll for status of the flag, proceed when 0
Trigger DMA read and write
Perform DMA read and write
Poll for completion
Set flag to 1, increment buffer ID
Retrieve read data
Not done
Done
Close file
Remove driver module

Figure 3.6: Steps for running the PCIe interface at different levels

When writing to the read header (to reset or trigger a DMA for example), it should be ensured that the current status of the three flag bits are not overwritten. Reading the 32-bit header with *io_read32*, updating the desired flags and then writing the 32-bit number back with *io_write32* is a risky option, since the flags could potentially be changed by the hardware side while the software is updating the register. Then, when writing the value back, the change made by the hardware could be overwritten and the system may fail. This option, although unlikely, has been avoided by a modification of the hardware: When the software attempts to write a 1 to one of those three bits, they are updated as expected, but writing a 0 has no effect. This way, only the hardware can switch the flags from 1 back to 0, and the driver does not need to worry about erasing those flags when writing to the read header.

3.3 Optical fiber interface

3.3.1 8b/10b encoding

In an electronic circuit, electric energy can be transferred between wires due to capacitive coupling. Although this phenomenon can be intentional in some designs, it can have harmful effects in the baseband transmission of digital data: Alternative Current (AC)-coupled electrical connections can lose the Direct Current (DC) information of the signal, as illustrated in figure 3.7. This translates to a channel with high-pass characteristics where the lower (DC) frequencies are attenuated.

Consider for example a system implementing binary phase-shift keying (BPSK) modulation, where the logical "0" bit of data is mapped into a negative voltage $-\sqrt{E_b}$, and the logical "1" is mapped into its positive counterpart $\sqrt{E_b}$. If the data's redundancy pattern produces a transmitted signal where 60% of the bits are logical 1s, the DC level of the transmit signal would be $0.2 \times \sqrt{E_b}$, which is its average value. After passing the signal through an AC-coupled channel, the received signal would lose the DC level information, and its average would drop down to 0.

This inevitably introduces errors in the transmission, and affects clock recovery information. In order to counter this, encoding schemes that produce a DC-balanced output out of possibly redundant input data have been designed.

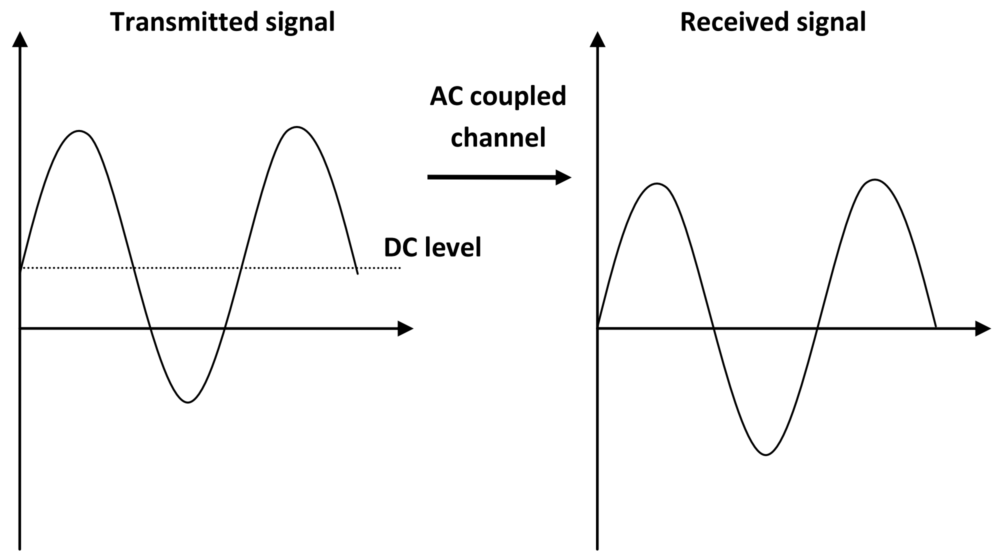


Figure 3.7: Effect of AC coupled channel on the DC component of a signal

The optical fiber interface design implements 8b/10b encoding. This is a line code with 20% overhead, as it maps each sequence of 8 data bits into a 10 bit symbol. It was first described in 1983 by Al Widmer and Peter Franaszek in [18].

The symbols are chosen such that DC-balance is achieved: No matter how redundant the data is, the total number of 1s and 0s transmitted only differ by at most 2 at all times. Also, except for a comma control signal used for synchronization, there are never more than five 0s or 1s transmitted in a row. This ensures proper clock recovery at the receiver end.

The 8 bits of data, denoted HGFEDCBA, are divided into two groups: The lower 5 bits (EDCBA), and the upper 3 bits (HGF). An 8-bit data string is denoted Dx.y, where x is the decimal value of EDCBA, and y is the decimal value of HGF. Control symbols are denoted

Kx.y. There are $2^{10} = 1024$ possible symbols, but only $2^{8} = 256$ possible data strings, so that only the symbols with less than 5 same consecutive bits are used. Also, some of the 8-bit data symbols can be encoded into two different 10-bit symbols, one of them having two more 1s than 0s and vice versa.

The x portion of the data is encoded into a 6-bit entity (abcdei), and the y portion is encoded into a 4-bit entity (fghj). The bits are then transmitted from least to most significant, as illustrated in figure 3.8.

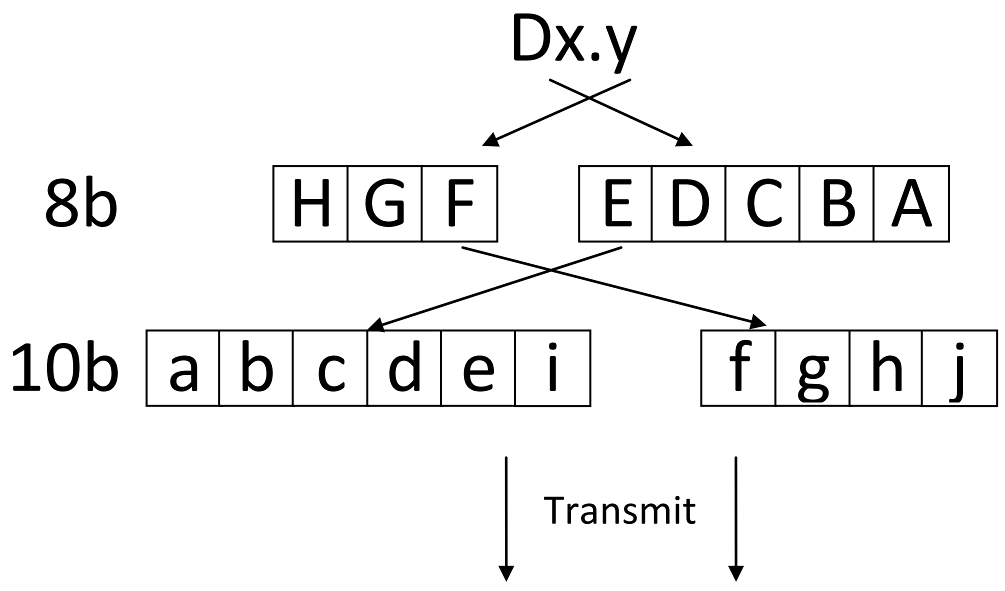


Figure 3.8: Bit ordering in 8b/10b encoding

The encoder needs to keep track of the disparity in the number of 1s and 0s sent. A value, called the Running Disparity (RD), holds this information. By convention, the RD starts at a value of -1. When the transmitted symbol has no disparity, the RD is left unchanged. In the case where the codeword to be transmitted is one that has two possible encodings, a disparity will occur. If the RD is -1, then the codeword with two more 1s is chosen and the next RD value is +1. If the RD is +1, then the codeword with two more 0s is chosen and the next RD is set to be -1. Therefore, this RD information has to be propagated and used as an input into the next encoding step. Note that the decoders do not require this information, since the backward mapping is unique. Tables 3.6 and 3.7 provide the mappings for encoding x and y. Table 3.8 provides the codes for common control symbols.

Input		Output	
		RD = -1	RD = +1
Notation	EDCBA	abcdei	
D.00.x	00000	100111	011000
D.01.x	00001	011101	100010
D.02.x	00010	101101	010010
D.03.x	00011	110001	
D.04.x	00100	110101	001010
D.05.x	00101	101001	
D.06.x	00110	011001	
D.07.x	00111	111000	000111
D.08.x	01000	111001	000110
D.09.x	01001	100101	
D.10.x	01010	010101	
D.11.x	01011	110100	
D.12.x	01100	001101	
D.13.x	01101	101100	
D.14.x	01110	011100	
D.15.x	01111	010111	101000
D.16.x	10000	011011	100100
D.17.x	10001	100011	
D.18.x	10010	010011	
D.19.x	10011	110010	
D.20.x	10100	001011	
D.21.x	10101	101010	
D.22.x	10110	011010	
D.23.x	10111	111010	000101
D.24.x	11000	110011	001100
D.25.x	11001	100110	
D.26.x	11010	010110	
D.27.x	11011	110110	001001
D.28.x	11100	001110	
D.29.x	11101	101110	010001
D.30.x	11110	011110	100001
D.31.x	11111	101011	010100

Table 3.6: Encoding of x in 8b/10b

Input		Output	
		RD = -1	RD = +1
Notation	HGF	fghj	
D.x.0	000	1011	0100
D.x.1	001	1001	
D.x.2	010	0101	
D.x.3	011	1100	0011
D.x.4	100	1101	0010
D.x.5	101	1010	
D.x.6	110	0110	
D.x.P7	111	1110	0001
D.x.A7	111	0111	1000

Table 3.7: Encoding of y in 8b/10b

Input		Output	
		RD = -1	RD = +1
Notation	HGFEDCBA	abcdeifghj	
K.28.0	00011100	0011110100	1100001011
K.28.1	00111100	0011111001	1100000110
K.28.2	01011100	0011110101	1100001010
K.28.3	01111100	0011110011	1100001100
K.28.4	10011100	0011110010	1100001101
K.28.5	10111100	0011111010	1100000101
K.28.6	11011100	0011110110	1100001001
K.28.7	11111100	0011111000	1100000111
K.23.7	11110111	1110101000	0001010111
K.27.7	11111011	1101101000	0010010111
K.29.7	11111101	1011101000	0100010111
K.30.7	11111110	0111101000	1000010111

Table 3.8: Control symbols in 8b/10b

For D.x.7, the preceding 6-bit code determines which one of the primary (D.x.P7) or the alternate (D.x.A7) code is to be sent. This choice prevents any sequence of 5 consecutive 1s or 0s (which is reserved for comma control symbols). Control codes K28.1, K28.5 and K28.7 are comma symbols used for determining the alignment of the 8b/10b encoded received signal.

In this design, the channel is 160 bits wide, and 8b/10b encoders are cascaded to map the 128 bits of data into the 160 bit encoded signal. In order to avoid an extra delay in data transmission, the running disparity is directly computed and propagated to the subsequent encoders without being registered. This allows all the encoders to produce their output on the same clock cycle, instead of waiting for the previous ones to finish. Only the RD output of the last encoder is registered and used as an RD input to the first encoder on the next clock cycle. This is illustrated in figure 3.9.

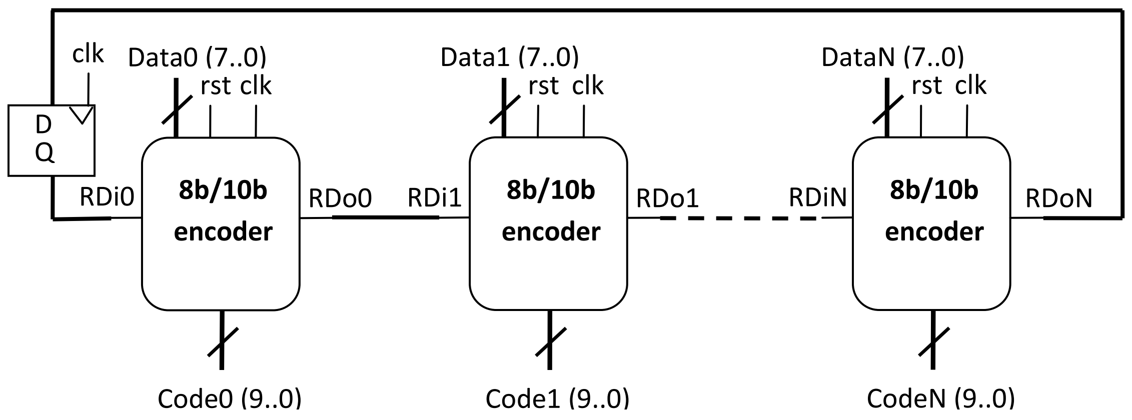


Figure 3.9: Cascading N 8b/10b encoders

Note that the errors introduced if no line code is used have been observed in a test setting. For example, when constantly sending a value of 0, the first few bits would be received properly, but the received signal would eventually flip to all 1s because the clock is not properly recovered anymore, and is seen with a delay. Also, transmitting a non-encoded pseudo-random pattern of 1s and 0s, using the full capacity of the channel without the 20% overhead, produces an error-free transmission, since in a random binary stream, it is rare that six 0s or 1s are generated consecutively, and the long term ratio of 1s to 0s is close to 1.

3.3.2 Structure of the optical fiber interface

The Verilog source code of the loopback demonstration provided by Terasic is used as a basis for the design [19]. The code already implements the pin interface with the HSMC. It takes care of comma detection and channel bonding, triggered by user push buttons on the board. It also instantiates a complex board test system (BTS). The BTS generates pseudo-random data, sends it to the channel, and compares it to the received data after it is looped back through the optical fiber. Note that this example design is targeted at the Stratix IV “development kit”, different from the DE4 board that is used in this project. In particular, the clock circuitries and the pin assignments are different and have to be adapted to match the characteristics of the DE4.

For the purpose of this design, the pseudo-random data that is generated is replaced by the output of the Random Access Memory (RAM) in the FPGA's internal memory, and the channel bonding, comma detection and start/end of transmission are automated. Also, 8b/10b encoder/decoder modules are implemented as described in section 3.3.1.

The hierarchy and structure of the optical fiber interface logic is shown in figure 3.10. The main modules are the following:

nios_bts_port_core: This module instantiates the lower level submodules. It propagates signals from the top level and initializes parameters such as the data width and the number of channels.

altgx_s4gx_basic_x4_8p5G: This is a module that implements an instance of a customized transceiver channel based on the MegaWizard function *altgx* as described in [10]. Its ports are routed to the top level and attached to the pins that map to the HSMC board. This transceiver module thus implements the link to the other node of the design. A transceiver is a combination of a transmitter and a receiver that acts as the interface between the digital processing domain and the analog transmission domain.

Further, we distinguish between stages where digital data is encoded and processed to prepare it for better transmission (Physical Coding Sublayer, PCS), and the stages that are concerned with converting analog to digital or digital to analog signals and connecting to the physical medium (Physical Medium Attachment sublayer, PMA). For example, on the receiver side, the PMA contains the Clock and Data Recovery (CDR) unit. In this design, most of the PCS blocks that can be implemented directly in the transceiver module are here disabled and implemented outside, such as word alignment and 8b/10b encoding. This is because the algorithms that are automatically supported by the transceivers do not exactly fit optical fiber standards.

altgx_s4gx_xcvr_reconfig_x4: This MegaWizard generated module implements the *altgx_reconfig* module, which allows for dynamic reconfiguration of the associated transceivers. This permits easier debugging or later updates and maintenance of the system. For example, when an *altgx_reconfig* module is used, Quartus' *Transceiver Toolkit* can be used to automatically search for optimum parameters for the altgx function, to test for system performance or draw eye diagrams.

xs_wrapper_bts: This module acts as a wrapper around lower level components that synchronize the signals. Also, after the signals have been word-aligned and channel-bonded for the first time, a *pattern_sync* piece of data is sent to the other node. The same pattern is later received from the other node. xs_wrapper_bts is responsible for capturing it and ensuring proper synchronization between the two nodes.

altera_bts: In the example design, this module instantiates pseudo-random data generation and manages the start and end of the test. It has been modified to read data from and write data to the RAM in the top level. It also instantiates the 8b/10b encoders and decoders that process the data before it is being transmitted and after it has been received, respectively.

tx_wrapper_bts: This module determines what data is to be sent to the other node. It has a Finite State Machine (FSM) that switches between states where a repeating DC balanced pattern, a word alignment pattern, followed by the actual user data are being transmitted. This allows for word alignment and channel bonding.

channel_aligner and **word_aligner**: These modules perform channel alignment and word alignment respectively. This is required to ensure proper detection of word boundaries at the receiver side; otherwise the data could be delayed by a few bits in either direction.

On system reset, each board continuously sends a predefined synchronization pattern. After one second, it attempts to detect "comma" signals in the received stream in order to align it properly as explained in section 3.3.1. One second later, it performs channel bonding. Once this initialization process has been performed, the optical fiber system is ready to reliably exchange data with the other node. The one second wait allows for a human user to reset the two systems using a push button on the DE4 board since it is possible to release the two buttons within one second of each other.

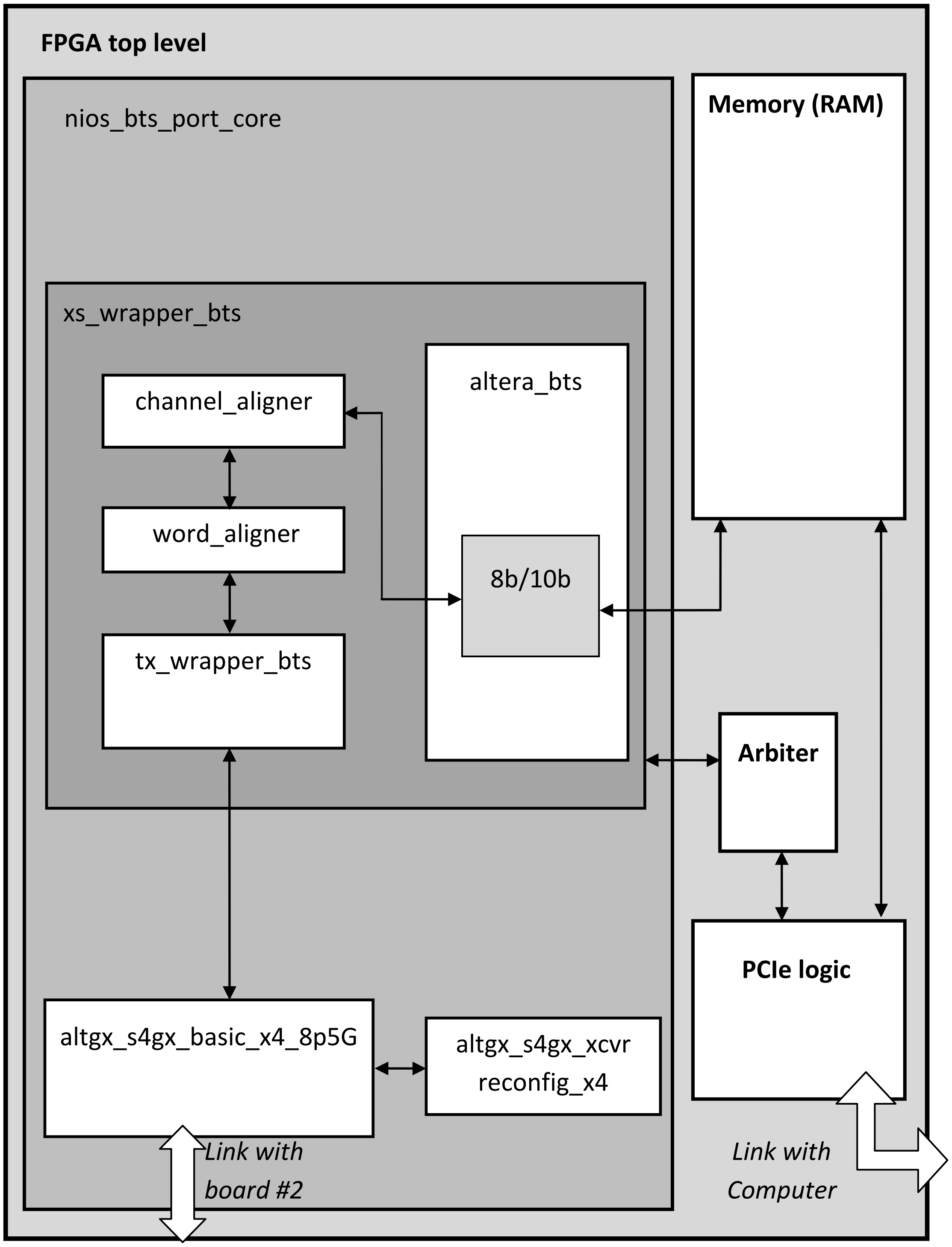


Figure 3.10: Application logic structure of the optical fiber interface

3.3.3 Synchronization between the two nodes

In order to keep the two end nodes synchronized, a mechanism needs to be implemented to signal to the other node that data is ready to be transmitted. Start and end of packet control signals are also useful to accurately determine the beginning and end of data. The only way to share such information is through the optical fiber link, at the cost of sending fewer data bits. But the overhead introduced is minor, and is not even seen by the whole system since the optical fiber link works faster than the PCIe one between the PC and the FPGA. This section explains the custom synchronization protocol that was implemented between the two nodes.

When no user data is ready to be transmitted between the two nodes, some predefined synchronization data is chosen to travel on the channel. As soon as node A is ready to exchange data with node B, it starts repeatedly sending *ready_on* signals. As soon as a *ready_on* signal is received from node B and node A is ready, it starts sending *start_on* signals. At the reception of a *start_on* signal, node A starts sending 128 bits of data on each transmit clock cycle. It also waits for the first received signal that is not a *start_on* signal to start recording 128 bits of received data on each receive clock cycle. A *start_on* state is required, because relying on *ready_on* only might result in node A sending user data before node B is ready to record it in case it has not seen the *ready_on* of node A yet. When all the data has been processed, an *end_on* signal is sent. On reception of an *end_on* signal, node A stops recording data. This scheme has the advantage of being completely symmetric, so that none of the nodes is solely responsible for triggering and controlling data exchanges. A and B can be used interchangeably in the explanation. Figure 3.11 illustrates the communication steps.

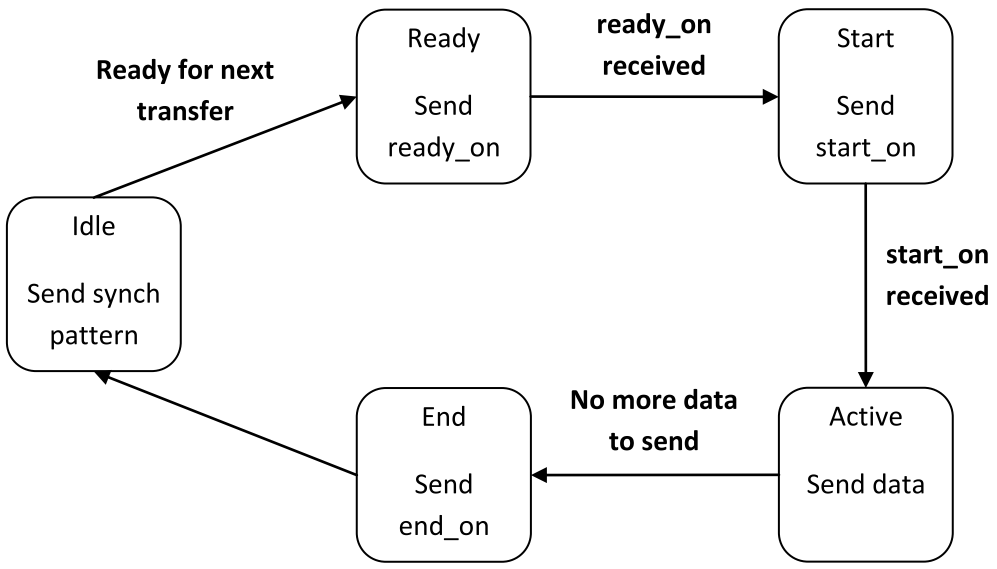


Figure 3.11: Communication steps to synchronize the two nodes

The three control signals are unique 8b/10b encoded signals: they cannot be confused with data symbols. In order to maintain a consistent RD, *ready_on* is chosen to be K28.0 repeated 16 times (in order to fill up the 16 x 10 = 160-bit channel), where even positions are the RD = +1 code, and odd positions are the RD = -1 code. *start_on* and *end_on* have the same structure, with K28.2 and K28.4 respectively.

ready_on = 0xC2CF4_C2CF4_C2CF4_C2CF4_C2CF4_C2CF4_C2CF4_C2CF4

start_on = 0xC28F5_C28F5_C28F5_C28F5_C28F5_C28F5_C28F5_C28F5

end_on = 0xC34F2_C34F2_C34F2_C34F2_C34F2_C34F2_C34F2_C34F2

3.4 Complete system

3.4.1 Interface between PCIe and optical fiber system

Although the PCIe link is the system's bottleneck and the optical fiber link does not always transmit useful data, it is only synchronized once on system startup, instead of performing synchronization for every new chunk of data to be transferred. On reset, comma detection takes place: It tries to identify sequences of five 0s and 1s in the 8b/10b control codes. Then, the four channels are bonded and data starts to be transmitted. Originally, only dummy data is continuously being transferred. When the PCIe link signals to the optical fiber interface that data is ready to be exchanged between the two nodes, synchronization takes place in the optical fiber link as explained in section 3.3.3 and user data starts being transmitted through the link. On completion, if another buffer of data is available, it will be transmitted as well, and so on. If no buffer is available yet and the PCIe is busy completing its DMA, then the optical fiber goes back to a state of transmitting dummy data until a new buffer is available.

In the PCIe application logic, changes in the value of the three reserved bits of the control field of the DMA read status register sitting on BAR2 are monitored. Those three bits are used as control bits for the three buffers storing data. Originally, the value of each bit is 0, and the driver can perform a DMA transfer. On completion of a DMA transfer on buffer #0, the software sets the corresponding flag to 1 and switches to buffer #1 for the next transfer. This rising edge is seen by the PCIe logic and a signal *flagAssertedOut*(0) is asserted. *flagAssertedOut* is a 3-bit signal: one for each flag and buffer. This signal is propagated to the *flagger* module to signal that data is ready to be transferred between the two nodes.

When the *flagger* module sees a high *flagAssertedOut* signal coming from the PCIe logic, it asserts the corresponding *SFPtransfer*(2:0) signal, propagated all the way to the design's top level. This signal informs the optical fiber interface logic that it can start a new transfer, and it remains high until completion of the transfer. On completion of the transfer, the corresponding *SFPflag*(2:0) signal, output of the optical fiber interface logic and input to *flagger*, is asserted. The corresponding *SFPtransfer* is then deasserted to acknowledge the reception of the signal, and *flagAssertedIn*(2:0) is asserted for one clock cycle to signal the PCIe logic that the flag can be switched back to 0.

The design's top level contains a register currentSFPtransfer(1:0). It has two bits in order to store 3 states, one for each buffer currently being used to read/write data with the optical fiber. When a rising edge of the current SFPtransfer control signal is seen, a new transfer is triggered and the proper protocol and connections are propagated further down the hierarchy. On completion of the transfer, the current SFPflag is asserted to signal completion to the *flagger* module, and *currentSFPtransfer* is incremented. It can now process other rising edges of *SFPtransfer* that possibly occurred during the previous transfer if the PCIe completed a transfer before the optical fiber did. When the *flagger* module sees a high *SFPflag*, it acknowledges It by setting *SFPtransfer* back to 0, which is in turn acknowledged by the top level that switches *SFPflag* back to 0 as well, going back to the initial state.

The hierarchical structure is illustrated in figure 3.12. The signal levels during such transitions are shown in figure 3.13; they are sampled at the transmitter clock rate and are obtained from experiments on the actual design using Quartus's SignalTap II Logic Analyzer. The waveform has been compressed in time to fit the page, the three black vertical lines show a longer period of time (during which the optical fiber transfer takes place). Note that the value of the signal *sendPattern* is low when the optical fiber has data to transmit, and it is high otherwise.

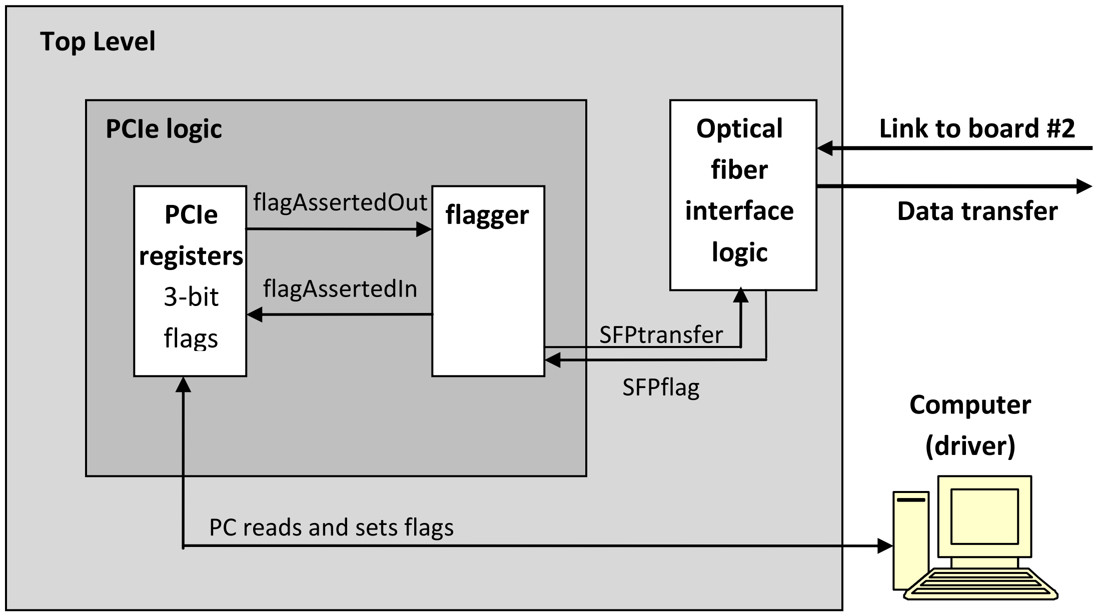


Figure 3.12: Links between the driver, PCIe logic and optical fiber interface logic

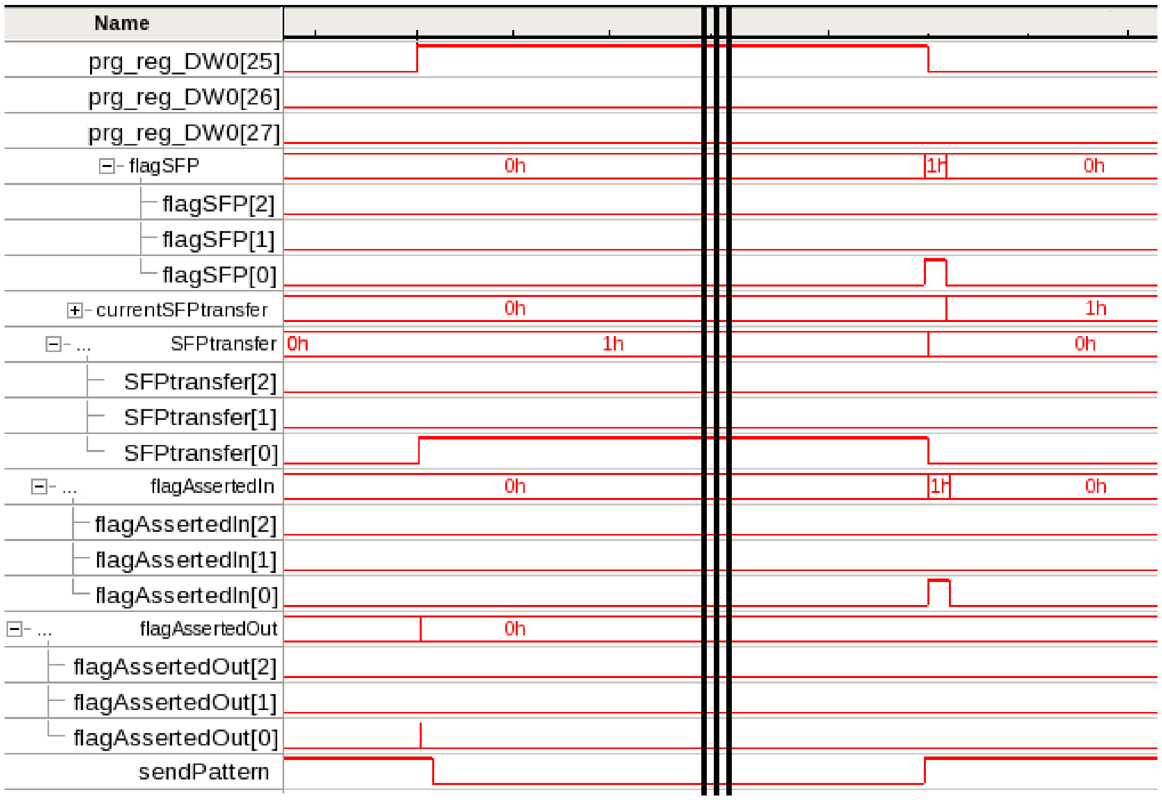


Figure 3.13: PCIe and optical fiber communication on SignalTap waveform

Although the main clock runs at 50MHz, a Phase-Locked Loop (PLL) is used to distribute a 125MHz clock to the optical fiber interface logic. This matches the 125MHz clock generated on the SFP daughter card that drives the transceivers. In addition to these, the PCIe logic itself uses several clocks (the main one is at 100MHz), and the receiver and transmitter logics both use a different clock.

Memory elements of a digital design are typically designed using flip-flops. These devices have inherent setup and hold times, and when they are not respected, the flip-flop can enter into a metasable state, where its state is not settled between neither 0 nor 1. The logic value might therefore switch back and forth until it finally settles, but if the rising edge of the clock occurs during that time, an erroneous value may be read and the system may fail. The control signals described in this section interact with logic in different clock domains. Thus, synchronizers must be used in order to ensure that proper values are read and to avoid unpredictable behaviour, at the cost of introducing a few clock cycles of delay in the protocol. The basic functioning of a synchronizer is illustrated in figure 3.14, where the signal in clock domain A is registered several times in cascaded registers in clock domain B before it is read by logic in B. With this scheme, if enough buffers are used, the probability of failure becomes very small and is suitable for most practical purposes.

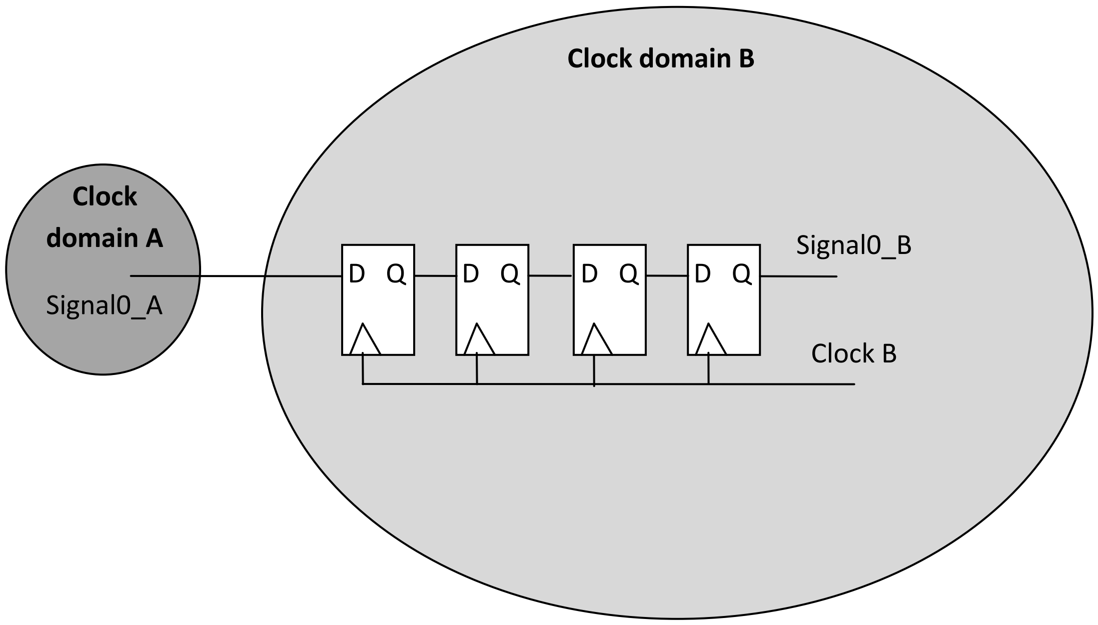


Figure 3.14: Synchronizing a signal to a new clock domain

3.4.2 FPGA resource usage

The compilation report returns the data shown in table 3.9.

Component	Value/Usage
Family	Stratix IV
Device	EP4SGX530KH40C2
Logic utilization	4%
… Combinational ALUTs	9,088 / 424,960 (2 %)
… Memory ALUTs	522 / 212,480 (< 1 %)
… Dedicated logic registers	11,616 / 424,960 (3 %)
Total registers	11,616
Total pins	275 / 888 (31 %)
Total virtual pins	0
Total block memory bits	8,405,376 / 21,233,664 (40 %)
DSP block 18-bit elements	0 / 1,024 (0 %)
Total GXB Receiver Channel PCS	8 / 24 (33 %)
Total GXB Receiver Channel PMA	8 / 36 (22 %)
Total GXB Transmitter Channel PCS	8 / 24 (33 %)
Total GXB Transmitter Channel PMA	8 / 36 (22 %)
Total PLLs	2 / 8 (25 %)
Total DLLs	0 / 4 (0 %)

Table 3.9: Compilation report data

In the above, ALUT stands for Adaptive LUT. GXB stands for Gigabit transceiver block. PCS stands for Physical Coding Sublayer and PMA stands for Physical Medium Attachment sublayer. Note that the logic utilization is low (4%), so that custom logic can be added on the FPGA in order to perform various processing on the data.

Chapter 4

Experiments

Section 4.1 describes in more detail the different hardware and software that were used to develop the system. Section 4.2 provides experimental results that justify some of the design choices such as the transfer size and driver/PCIe interface strategies. Bandwidth and latency measurements are also reported for different components of the system, and the performance of the complete system is compared to the expected performance of a similar system based on other modern optical fiber standards.

4.1 System setup and physical layer

Due to the complexity of the design, logic simulators such as Modelsim have not been used. On the other hand, Quartus's Signal Tap II Logic Analyzer has been crucial for determining issues with the design and debugging logical errors.

The compilation time of the design is about 25 minutes when using a computer with one 3GHz processor. Note that multicore computers do little to improve compilation time, since for most of the compilation process (Analysis & Synthesis, and Fitter (place & route)), only one core is being used.

The hardware design is described using a hardware description language (HDL). The most common ones are VHDL (VHSIC HDL, where VHSIC stands for very-high-speed integrated circuits) and Verilog. The two languages being very similar, either one could be chosen. Also, since they are ultimately equivalent at the component or module level of abstraction, both could be used in the same design. In this work, Verilog was chosen.

The system has been developed and tested between two Linux computers running Ubuntu 10.04 with Kernel version 2.6.32-33. They both total 8.2GBytes of RAM and are equipped with a quad-core AMD Phenom™ II X4 955 processor chip. Note that the performance of the system is the same whether it is set up between two computers or in a loopback mode, and it was found that it behaved faster on a different computer, with 3.2GBytes of RAM, a AMD Phenom™ II X6 1055T processor chip and the Fedora operating system with Kernel 2.6.34.8-68.fc13.i686. This computer runs the system at about 9Gbit/s instead of 8.38Gbit/s as measured here and explained later. There was no other such computer available to test the full system at this speed, but this loopback clearly shows the overhead associated on the PC side, and it implies that the measured speed of 8.38Gbit/s for the system can easily be further increased without changing the design, but just updating the computer's software and hardware.

The design has been processed with the Altera Quartus II software for a Stratix IV FPGA chip, specifically the EP4SGX530KH40C2N that is found on the Altera Development and Education board (DE4). The FPGA chip provides a logic capacity that is equivalent to approximately 530,000 logic blocks using 4-input LUTs. It provides 27,376K of memory bits, 4 PCI Express Hard IP (Intellectual Property) blocks, and 8 phase locked loops (PLLs) [20]. The DE4 board also provides many useful features, including integrated transceivers supporting up to 8.5Gbit/s, a PCIe x8 edge connector and two HSMC connectors [21]. Designs are downloaded to the board through a Universal Serial Bus (USB) cable and a Joint Test Action Group (JTAG) interface. HSMC is a low-cost and high performance physical interface designed by Altera to allow for third-party hardware to be added to the development boards and interfaced with the FPGA [22].

The PCIe interface between the DE4 board and the computer uses 4 lanes and thus achieves a maximum theoretical physical throughput of 20Gbit/s. The HSMC port is connected to a daughter card: Terasic’s Small Form-factor Pluggable (SFP) HSMC board. This board provides 8 SFP connectors: 4 Transceiver based ones, as well as 4 Low-Voltage Differential Signaling (LVDS) based ones [19]. The SFP transceiver modules for the optical fiber interface are Finisar’s FTLF8524P2BNV 1000Base-X model [23]. They are plugged into the 4 transceiver based SFP slots of the daughter card, and emit light of wavelength 850nm using Oxide VCSELs. The optical fiber cables are 50cm long; they have *Lucent Connectors* (LC) and are inserted in the SFP modules. The cables are of the

62.5/125 µm family: they are multi-mode fibers with a core size of 62.5µm and a cladding diameter of 125µm.

As mentioned in the specifications for the SFP modules [23], the case operating temperature ranges from -20°C to 85°C, and the link is expected to be reliable with cable lengths up to about 70m. Furthermore, for a single supply voltage of about 3.3V, the stressed receiver sensitivity is evaluated at about 0.55mW for all four channels combined. This parameter describes the minimum optical power required at the receiver to recover the signal with a Bit Error Rate (BER) of at most 10^{-12}, as required by many optical channel specifications, including 10 Gigabit Ethernet.

At the transmitter side, the optical modulation amplitude has 247µW for each channel. The optical rise and fall times are at most 90ps, the total jitter is at most 57ps, and the spectral width is at most 0.85nm: This is the amount of deviation from the nominal value of around 850nm observed on the optical wavelength, and is ideally equal to zero.

4.2 Determining the transfer size at the PCIe level

Note that the speeds reported are for a two-way communication, which is inherently possible with the hardware available (PCIe and two-way optical fiber links). For example, a speed of 1Gbit/s means that every second, 1Gbit of data is sent to the other node, and 1Gbit of data is received from the other node.

For both the PCIe link and the optical fiber link, a handshaking protocol takes place before each transfer. This protocol runs with almost constant time no matter the amount of data to be exchanged per transfer (as this has been fixed in the design). The protocol is an overhead, but is required since FPGA storing resources are limited, and the data generated by the high level C programs is not necessarily all available from the start of the application: If it is generated in real time it should be buffered until enough data is available to constitute a new transfer. The greater the size of each transfer, the smaller the time spent on protocol overhead, but the greater the latency for transferring one particular bit from the first node to the second.

Since the optical fiber is faster than the PCIe link, its protocol overhead does not ultimately matter, and deciding the transfer sizes should be based on analysis of the PCIe throughput. In order to maintain a lower system complexity, the transfer size is chosen to be fixed. If the data rate of the used application is too slow, dummy bits can be appended to the data in order to fill up the buffers more quickly and avoid communication latency between the two nodes. In the PCIe case, time is wasted in writing/reading data between the C program and the device driver's file. Here, the computer's memory capabilities and processing speed play an important role. Once the data is available to the driver, the communication protocol consists of setting up descriptor tables that are then fetched by the PCIe logic, then triggering a DMA request by writing to PCIe registers. On completion, interrupt handling or register polling are other tasks that further slow down the system. The PCIe protocol itself uses a packet acknowledgement system that significantly slows down throughput in the case of a high traffic interface. Figure 4.1 shows the evolution of the measured PCIe throughput for an increasing transfer size. The measurements have been performed in a loopback setting, where a C program exchanges data with the internal memory of an FPGA, without using any other node. Since measurements are done at the high level C program sitting on top of the system, software overhead plays an important role in reducing the theoretical maximum PCIe speed. Data is sent and received in the form of successive DMA requests of varying size.

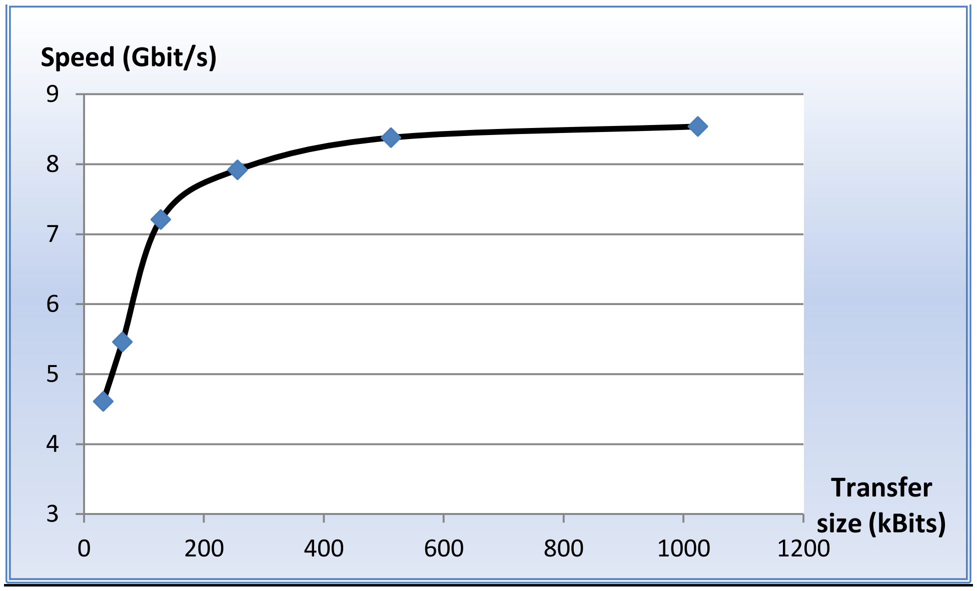


Figure 4.1: PCIe throughput versus DMA size

As expected, for greater transfer sizes, the PCIe link is faster. But the curve has an asymptotic behaviour, and we can clearly see that after a certain memory size, the protocol overhead is negligible and very little speed is gained by further increasing the memory, at the cost of increasing FPGA resource usage and latency due to packet size, as shown in figure 4.2.

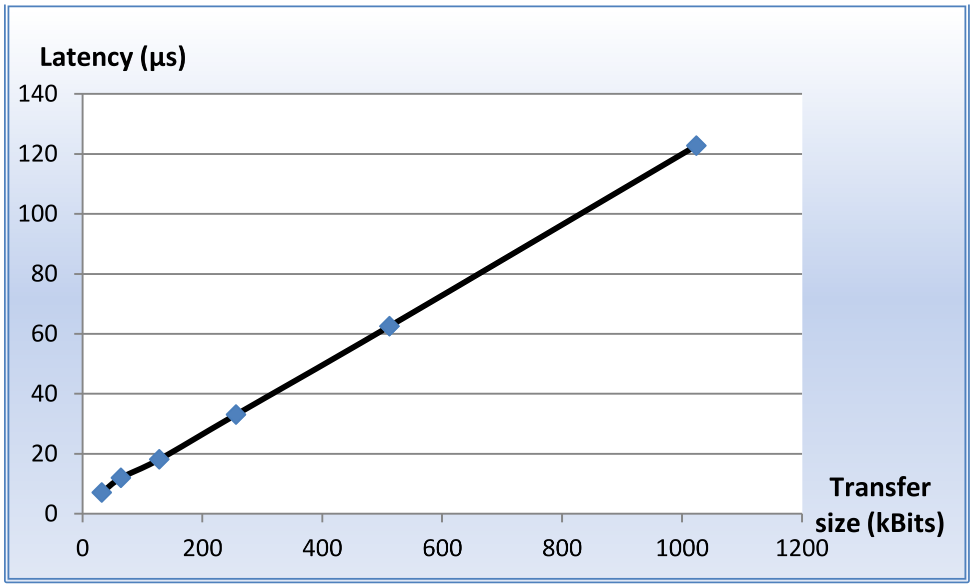


Figure 4.2: PCIe latency versus DMA size

When the transfer size is switched from 512 kBits to 1Mbit, the latency is approximately doubled, but the bandwidth improvement is minimal. Thus, for the purpose of this design, the transfer size was chosen to be 512 kBits. Also, this size leaves enough FPGA memory for possibly more than three buffers in the case of applications where heavier processing on the data is required, as explained in section 5.2.

4.3 Low-level speeds of the PCIe and optical fiber links

Measuring the PCIe speed for the physical connection only, ignoring some of the protocol overhead, can be done by comparing the values of a counter between enough sent/received packets during a transfer. This counter has its value incremented on every positive edge of the reference 50MHz clock. We observe 542 cycles of the 50MHz clock for 1940 64-bit data chunks successfully transferred during a DMA write, resulting in a speed of about 11.45Gbit/s, and 1892 64-bit data chunks successfully transferred during a DMA read, resulting in a speed of about 11.17Gbit/s. This measurement does not provide the direct physical speed of the interface, as it includes the 8b/10b encoding, the packet headers and some potential replayed packets, but it ignores the overhead in initializing and acknowledging the transfer between the driver and the PCIe logic.

We can do the same approximate calculation for the optical fiber link, by measuring the speed between two transceivers. We observe 491 cycles of the 50MHz clock for 1534 160-bit packets sent and received. This yields a physical speed of 24.99Gbit/s for the four channels. This speed corresponds to the expected 6.25Gbit/s per channel, and is a protocol-free measurement. Adding the 8b/10b encoding, the effective data rate after the handshaking protocol initiating each transfer is thus measured to be 5Gbit/s per channel, for a total of 20Gbit/s between the two computers.

4.4 Driver design considerations

In an attempt to better understand the point of slowdown on the software side, an experiment has been conducted where the data was directly available inside the driver instead of writing to it with a C program. This eliminates any overhead due to file input/output (I/O) between the C program and the device driver’s file. The increase in speed was only 5% with respect to the character-device driver implementation. This result indicates that even if a block device implementation had been implemented instead, only up to a 5% increase in speed would have been observed, which does not indicate a major bottleneck.

In comparing the two "check for completion strategies" in the PCIe link, it was observed that issuing an MSI interrupt and having the driver detect it and handle it takes a constant time regardless of the size of the data being transferred. In the polling case, a more customized strategy can be adopted. In fact, after issuing a DMA request, a polling scheme implies that the driver needs to repeatedly poll the status of the register indicating the last processed descriptor until it reaches a desired value. This process consumes resources and slows down the DMA transfer if it is still active. In order to minimize this overhead, it is desirable that polling occurs only once, right after the DMA transfer has been completed. Therefore, a *wait/sleep* statement can be used right after the DMA request has been issued, so that the driver waits an amount of time t_1 before its first poll. If the first poll is unsuccessful, then subsequent polling should be done within an interval of time t_2 smaller than t_1. Optimal values for t_1 and t_2 are determined experimentally, and they highly depend on the transfer size. The aim is to minimize both the number of polls and the time between completion and the first successful poll. On average, using optimal values for t_1 and t_2, polling was found to be about 10% faster than an MSI scheme.

4.5 Measuring system bandwidth and latency

In order to use the complete system, the two C programs need to be running at the same time. To keep things simpler, no protocol has been implemented to trigger the communication and take care of the fact that the C programs will not be started at the exact same time. Instead of this, when the driver first polls for the status of the three bits that indicate whether it can perform a DMA transfer, a long wait of 5 seconds is used in case of a failed polling. This gives enough time for the user to start the other program as well. Once the programs are started, data is continuously being exchanged in chunks of 512 kBits. 600,000 chunks are exchanged (for a total of 314,572,800,000 bits), and the experiment is repeated 5 times. The results are tabulated in table 4.1. According to these results, the average high level speed of the system can be set at 8.38Gbit/s.

Experiment #	1	2	3	4	5
Runtime (s)	37.55	37.51	37.54	37.55	37.52
Speed (Gbit/s)	8.377	8.386	8.380	8.377	8.384
				Average speed: 8.38 Gbit/s	

Table 4.1: System throughput measurements

The optical fiber link used in this design achieves a data rate of 20Gbit/s. Compared to the InfiniBand technology, this sits between the achievable speeds of 16Gbit/s in Double Data Rate (DDR) and 32bit/s achievable in Quad Data Rate (QDR) modes. Those speed values concern four channels and include encoding overhead but not other protocol requirements. This also approximately corresponds to the rate achievable using two 10G Ethernet cables. The link is therefore well suited for high-speed applications, since it permits faster transfers than many modern optical fiber standards. However, optical fiber links in general scale almost linearly in speed by just adding more channels, and the main advantage of this design is the presence of an FPGA that sees the transmitted data, and that can for example process it and use the extra throughput available in the optical fiber channels to send additional information (since the ultimate system throughput is limited by the PCIe connection, extra unused bandwidth is available in the optical fiber channels).

The designed interface is eventually intended to be used in a high-speed distributed simulation environment. Latency is especially critical in such an application, because the delay between the different cores or processing units is to be maintained as low as possible in order to avoid rollbacks, deadlocks and other events that affect the simulation's performance. As far as the optical fiber link is concerned, its latency mainly depends on transfer speed and packet size, but also potentially on the technology used, the cable's length and other physical factors. In this case, the length of the cable is 0.5m, contributing to a 2.5ns increase in latency, which is insignificant compared to what is measured for the rest of the system.

In fact, latency can be measured using a SignalTap waveform and counting the number of cycles of the transmit clock between the moment a signal is sent and the moment it is acknowledged on the receiver channel. The transmit clock speed is determined as explained previously; it has a value of 156.25MHz. Performing this measurement, the latency is found to be around 0.73µs. This measures the latency on the optical fiber link itself, including line encoding/decoding and the overhead due to the 160-bit packet size. This value closely corresponds to the expected latency on a similar InfiniBand link (in the order of 1µs), and is clearly better that what can be expected from the same experiment in an Ethernet connection: about 10-100µs [26].

Although this measurement supports the fact that this system can outperform other optical fiber standards, it does not provide an accurate view at the high level, end-to-end user application level. Indeed, the software overhead, the combination of a PCIe link, an optical fiber one and buffers to hold the data and maintain high bandwidth all further increase the system latency as seen by the C programs lying on top of the system. This high-level latency is easily measured by the software at around 260µs. This value is not a significant loss compared to the capability of the optical fiber link, and is thus acceptable for most high-speed applications. In particular, it is in the same order as Ethernet latency, despite the 3-buffer structure that can be used for other purposes and that make this design more attractive than an Ethernet counterpart achieving the same bandwidth. Note however that this latency could be significantly reduced should the application require it, by allowing a variable transfer size or a variable number of buffers for the data. Here, large transfer sizes were chosen to increase the bandwidth, at the cost of increasing latency. For example referring to [27], where the performance of PCIe 2.0 with QDR InfiniBand is analyzed, bandwidth calculations are performed on large message sizes: 1byte to 1Mbyte on a logarithmic scale, whereas latency calculations are performed on message sizes from 1byte to 1Kbyte, with a performance that starts degrading after message sizes of 64bytes. Furthermore, note that according to their experimental results, message sizes of 512Kbits = 64Kbytes also seem to be a point where a further increase of the size only yields a negligible improvement in bandwidth.

Chapter 5

Conclusions and Future Work

5.1 Achievements

Since the capabilities of single computers are too limited for most high-performance computing applications, these are usually being carried out by parallelizing the computational tasks and spreading the workload between several computing units. The interconnects between the different nodes of the system play a critical role in ensuring the efficient execution of parallel computation algorithms. In recent work, high bandwidth and low latency for these interconnects have typically been achieved through optical fiber based links such as 10-GigE and InfiniBand. The high-speed interface presented in this thesis links two computers together through four optical fiber channels at 25Gbit/s, and can provide better performance than Ethernet and InfiniBand counterparts. Furthermore, the optical fiber link is interfaced to FPGA chips situated on each computer. Each FPGA sees all the data before it is transmitted to the other node and after it is received from the other node, and it is interfaced with a computer through a PCIe x4 Gen2 link running at 20Gbit/s. PCIe is an industry standard protocol that provides good performance, reliability, scalability and that is widely used in I/O links between a device and a computer’s memory. The computer communicates with the hardware and triggers DMA transfers with the FPGA through a kernel level driver. Due to the triple buffering structure, the overall link speed does not drop below that of the slower interface: the PCIe. This buffering structure also gives time to the FPGA to process the data in parallel before it is being transmitted and after it is received. Different strategies have been explored to maximize the performance of the DMA transfers. Ultimately, including protocol and software overhead, the data rate of the design was measured to be about 8.38Gbit/s from one C program to another.

5.2 Future extensions to related areas

Keeping the design as it is, the optical fiber link has got some extra throughput that could be used to share extra information about the data being transmitted. This extra throughput can further be increased if the FPGA is used to compress the received data before transmitting it and then decompress the received data. This compression capability could also be used to cut down the physical optical fiber bandwidth itself, by using only two channels instead of four, and still keeping the same speed performance. This is of course feasible assuming an application with high enough redundancy in the data being transmitted, such that compression has a significant effect. Compression, as well as any other digital processing on the data such as convolution operations for filtering, does not affect the overall system bandwidth, its costs are in FPGA logic block area and in delaying the reception of transmitted data. Fewer optical fibers could also be used in order to introduce other nodes in the design, at the cost of having the optical fiber become the bottleneck. Still, the overall structure would remain the same, only the PCIe link would be waiting for optical fiber completions, and not the other way around as is the case in this design.

The data is currently stored in 3 memory buffers. This gives time to do processing on one buffer and transmit the result with the optical fiber. In case the operation is time expensive, additional buffers can easily be introduced at the cost of increasing system latency. In this design, only 14.8% of the available FPGA memory is actually being used by the buffers. Therefore, everything else remaining the same, and keeping buffers of 1 Mbit, up to 20 of them could be used, instead of 3 currently. The number of memory buffers and the transfer size could also be varied to achieve different bandwidth and latency trade-offs, depending on the application's requirements and performance.

If the amount of data required to start processing is greater than the size of one buffer, or greater than what can potentially be stored in the FPGA's internal memory, the data can easily be transferred to external memory available on the DE4 board: 64MB Flash memory, 2MB SSRAM or 8GB DDR2 SO-DIMM sockets (double data rate 2 small outline dual in-line memory module). A PCIe reference design to external memory is provided by Altera in [24].

Harnessing the full power of the FPGA in the interface is a task that remains to be explored, and that will certainly spark new ideas and provide many interesting challenges.

The interface was originally intended to connect two PCs that perform parallel logic simulations using Graphics Processing Units (GPUs). This FPGA based high speed interface provides additional powerful tools to communicate simulation data between the two nodes. With respect to this application, one improvement to the design would be to allow the GPU units to directly trigger DMA transfers on their own, without consuming CPU computational cycles. GPUs offer a high amount of parallelism due to their inherent design intended at manipulating large amounts of (graphical) data in parallel.

The design may eventually be further ameliorated by making use of the efforts and developments recently pursued by Altera to embed an optical fiber infrastructure directly on FPGAs [1]. The latest designed FPGAs are to be integrated with laser and photon detectors in order to provide range, power, cost and density advantages over equivalent copper-based interconnects that are reaching their bandwidth limits and cannot scale to future technological needs anymore. Figure 5.1 shows an example of such an FPGA, with a transmitter optical sub-assembly (TOSA) and a receiver optical sub-assembly (ROSA) placed on the corners of the chip.

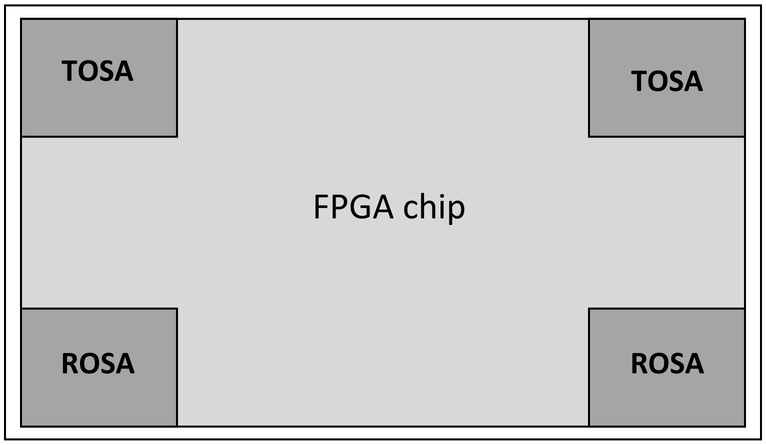


Figure 5.1: Integrated optical fiber infrastructure on an FPGA

The link developed in this thesis demonstrates an important tool to support other research activities, mainly due to its scalability, high bandwidth, low latency, and the presence of programmable logic monitoring the transiting data. For example, consider the NUMAchine [31][32], a large-scale shared-memory multiprocessor with tightly-coupled processing units implementing a CC-NUMA (Cache-Coherent Non-Uniform Memory Access). The use of programmable logic and high-speed interconnects is key in the successful implementation of such a multiprocessor system that requires high bandwidth and low latency between the different nodes.

Furthermore, in logic simulation and debugging, the FPGAs can provide debug support on top of the system to increase visibility and lead to more efficient circuit debugging. In particular, they can lower the cost of hardware assertion on-line monitoring and provide essential verification and debugging information without interrupting or slowing down the system [33][34].

This work also constitutes a good reference for the design of Network On Chips (NoCs) that require high bandwidth between processing elements. One major research effort is in making these designs fault-tolerant [35][36] and maintaining reliability as the size of integrated circuits decreases and the effect of process variations increases. High bandwidth reconfigurable links can be used in fault-reliable NoC routers, where errors can be monitored and their effect mitigated on the fly, thus decreasing packet latency in the network in case of failures. Combining the two previous points, [41] presents a debugging and monitoring infrastructure for the design of reliable and failure-free NoCs.

Testing of high-speed serial interfaces has also been the subject of various research efforts because of the expensive instrumentation, long test time and signal integrity issues [37]. The work presented in this thesis can play a major role in facilitating such expensive tasks in various ways. Since the data transits through an FPGA, data integrity can efficiently be checked at the system level using logic analyzers such as SignalTap, which is in fact done here for several calculations in section 4. Furthermore, tools such as Altera's Transceiver Toolkit also provide an easily accessed testing infrastructure with such features as automatic test data generation and BER calculation for transceiver components, but also EyeQ diagram generation to check signal integrity in more detail.

The presence of the FPGA in the link also suggests an extension to multiple-valued logic in order to increase the system's efficiency [38][39], as well as an improvement of the clock management system. GALDS (Globally Asynchronous, Locally Dynamic System) [40] provides a system-level framework for designing efficient, multiple-clock based digital systems. In fact, numerous clock domains are used as part of the design presented in this thesis: The FPGA logic, the PCIe protocol, the HSMC interface, the daughter card, the transmit and the receive transceivers all use different clocks. The clocks in one node are either generated independently or related through PLLs [25], and the different clock domains further need to be synchronized between the two nodes of the system.

References

[1] Altera Corporation. “Overcome Copper Limits with Optical Interfaces.” Internet: http://www.altera.com/literature/wp/wp-01161-optical-fpga.pdf, April 2011 [accessed September 2011].

[2] Corning Incorporated. “Optical Fiber Local Area Networks: Bandwidth, Data Rate, and Link Length - What Does It All Mean?” Internet: http://www.corning.com/docs/opticalfiber/wp410_10-01.pdf October 2001 [accessed September 2011].

[3] S. Minami, J. Hoffmann, N. Kurz and W. Ott, "Design and Implementation of a Data Transfer Protocol via Optical Fiber," *Real Time Conference (RT), 2010 17th IEEE-NPSS*, pp. 1-3, 2010.

[4] S. Minami, J. Hoffmann, N. Kurz and W. Ott, "Design and Implementation of a Data Transfer Protocol Via Optical Fiber," *IEEE Transactions on Nuclear Science*, vol. 58, no. 1, pp. 1816-1819, 2011.

[5] R.M. Metcalfe and D.R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Communications of the ACM*, vol. 19, no. 7, pp. 395-404, 1976.

[6] J. Bardeen and W. Brattain, “The Transistor, a Semiconductor Triode,” *Phys. Rev*., vol.74, p. 230, 1948.

[7] Intel Corporation. “60 years of the transistor: 1947 - 2007.” Internet: http://www.intel.com/technology/timeline.pdf, [accessed September 2011].

[8] J. M. Rabaey, A. Chandrakasan and B. Nikolic, "Introduction," in *Digital Integrated Circuits*, 2nd edition. Sodini, Ed. Upper Saddle River, NJ: Prentice Hall, 2002, pp. 3-34.

[9] Altera Corporation. “Avalon Interface Specifications.” Internet: http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, May 2011 [accessed September 2011].

[10] Altera Corporation. “Transceiver Architecture in Stratix IV Devices.” Internet: http://www.altera.com/literature/hb/stratix-iv/stx4_siv52001.pdf, June 2011 [accessed September 2011].

[11] J. Corbet, A. Rubini, G. Kroah-Hartman. *Linux Device Drivers*. Sebastopol, CA: O’Reilly Media, 2005.

[12] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171-210, June 2002.

[13] R.D. Maurer, "Glass fibers for optical communications," *Proceedings of the IEEE*, vol. 61, no. 4, pp. 452-462, April 1973.

[14] T. Güneysu, T. Kasper, M. Novotný, C. Paar and A. Rupp, "Cryptanalysis with COPACOBANA," *IEEE Transactions on Computers,* vol. 57, no. 11, pp. 1498-1513, November 2008.

[15] L. Wienbrandt, S. Baumgart, J. Bissel, C.M.Y. Yeo and M. Schimmler, "Using the reconfigurable massively parallel architecture COPACOBANA 5000 for applications in bioinformatics," *Procedia Computer Science 1,* vol. 1, no. 1, pp. 1027-1034, 2010.

[16] Altera Corporation. “PCI Express High Performance Reference Design.” AN-456-1.3. Internet: http://www.altera.com/literature/an/an456.pdf, August 2010 [accessed September 2011].

[17] Altera Corporation. “IP Compiler for PCI Express - User Guide.” Internet: http://www.altera.com/literature/ug/ug_pci_express.pdf, May 2011 [accessed September 2011].

[18] A.X. Widmer and P.A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code," *IBM Journal of Research and Development,* vol. 27, no. 5, pp. 440-451, 1983.

[19] Terasic Technologies Incorporated. “SFP HSMC – Terasic SFP HSMC Board User Manual.”Internet: http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=342&FID=c4b7d99c26cb60c9e3bdb4f2ecdf10dd, September 2009 [accessed September 2011].

[20] Altera Corporation. “Overview for the Stratix IV Device Family.” Internet: http://www.altera.com/literature/hb/stratix-iv/stx4_siv51001.pdf, June 2011 [accessed September 2011].

[21] Terasic Technologies Incorporated. “DE4 User Manual.” Internet: http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Langua

ge=English&No=501&FID=b1388d95cec372035605969d3b3727e4, August 2010 [accessed September 2011].

[22] Altera Corporation. “High Speed Mezzanine Card (HSMC) – Specification.” Internet: http://www.altera.com/literature/ds/hsmc_spec.pdf, June 2009 [accessed September 2011].

[23] Finisar Corporation. “Product Specification - 4.25Gb/s RoHS Compliant Short-Wavelength SFP Transceiver - FTLF8524P2xNy.” Internet: http://www.finisar.com/sites/default/files/FTLF8524P2xNy%20Spec%20RevJ.pdf June 2009 [accessed September 2011].

[24] Altera Corporation. “PCI Express to External Memory Reference Design.” AN-431-2.0. Internet: http://www.altera.com/literature/an/an431.pdf, May 2011 [accessed September 2011].

[25] Z. Zilic "PLL- and DLL-based Clock Management for Digital Circuits," *On-line Symposium on Electrical Engineering*, May 2001.

[26] HPC Advisory Council. “Interconnect Analysis: 10GigE and InfiniBand in High Performance Computing.” Internet: http://www.hpcadvisorycouncil.com/pdf/IB_and_10GigE_in_HPC.pdf 2009 [accessed September 2011].

[27] M.J. Koop, Huang Wei, K. Gopalakrishnan, D.K. Panda, "Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand," *16th IEEE Symposium on High Performance Interconnects, 2008. HOTI '08.*, pp.85-92, August 2008.

[28] J.L. Shin, Huang Dawei, B. Petrick, Hwang Changku, K.W. Tam, A. Smith, Pham Ha, Li Hongping, T. Johnson, F. Schumacher, A.S. Leon, A. Strong, "A 40 nm 16-Core 128-Thread SPARC SoC Processor," *IEEE Journal of Solid-State Circuits,* vol.46, no.1, pp.131-144, January 2011.

[29] V. Krishnan, "Towards an integrated IO and clustering solution using PCI express," *IEEE International Conference on Cluster Computing*, pp. 259-266, September 2007.

[30] P. Germann, M. Doyle, R. Ericson, S. Lewis, J. Dangler, A. Patel, "Pushing the limits of PCI-express: A PCIe application within an IBM supercomputing environment," *Electronic Components and Technology Conference 58th*, pp. 495-501, May 2008.

[31] A. Grbic, S. Brown, S. Caranci, R. Grindley, M. Gusat, G. Lemieux, K. Loveless, N. Manjikian, S. Srbljic, M. Stumm, Z. Vranesic and Z. Zilic, "Design and Implementation of the NUMAchine Multiprocessor," *Proceedings of 35th ACM/IEEE Design Automation Conference DAC `98*, pp.66-69, San Francisco, June 1998.

[32] S. Brown, N. Manjikian, Z. Vranesic, S. Caranci, A. Grbic, R. Grindley, M. Gusat, K. Loveless, Z. Zilic and S. Srbljic, "Experience in Designing a Large-Scale Multiprocessor using Field-Programmable Devices and Advanced CAD Tools," *Proceedings of 33rd ACM/IEEE Design Automation Conference DAC `96*, pp. 24-29, Las Vegas, June 1996.

[33] M. Boulé and Z. Zilic, "*Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring,*" Springer Verlag. 2008.

[34] M. Boulé, J-S. Chenard and Z. Zilic, "Debug Enhancements in Assertion-Checker Generation," *IET Computers and Digital Techniques,* Vol. 1, No. 6, pp. 669-677, November 2007.

[35] M. H. Neishaburi and Z. Zilic, "Enhanced Reliability Aware NoC Router," *Proc. Intl. Symposium on Quality Electronic Design*, ISQED 2011, 6 pages, March 2011.

[36] M. H. Neishaburi and Z. Zilic, "Reliability Aware NoC Router Architecture Using Input Buffer Sharing," *Proceedings of Great Lakes Symposium on VLSI*, pp. 511-516, May 2009.

[37] Y. Fan and Z. Zilic, "*Accelerating Test, Validation and Debug of High Speed Serial Interfaces,*" Springer Verlag. 2011.

[38] Z. Zilic and Z. G. Vranesic, "Multiple Valued Logic in FPGAs," *Proceedings of 36th Midwest Symposium on Circuits and Systems,* pp. 1553-1556, Detroit, Michigan, August 1993.

[39] Z. Zilic and Z. Vranesic, "A Multiple-Valued Reed-Muller Transform for Incompletely Specified Functions," *IEEE Transactions on Computers*, vol. 44, No. 8, pp. 1012-1020, August 1995.

[40] A. Chattophadyay and Z. Zilic, "GALDS: A Complete Framework for Designing Multi-clock ASICs and SoCs," *IEEE Transactions on Very Large Scale Integrated Circuits (VLSI)*, Vol. 13, No. 6, pp. 641-654, June 2005.

[41] J-S. Chenard, S. Bourduas, N. Azuelos, M. Boulé and Z. Zilic, "Hardware Assertion Checkers in On-Line Detection of Faults in a Hierarchical-Ring Network-On-Chip," *poster, Workshop on Diagnostic Services in Network-on-Chips, DATE 2007 Conf.*, Nice, France, April 2007.