University POLITEHNICA of Bucharest
Faculty of Electronics, Telecommunications and Information Technology

Implementation and Evaluation of a Hardware-Accelerated EtherCAT Slave Protocol

Dissertation Thesis

submitted in partial fulfillment of the requirements for the Degree of *Engineer*
in the domain *Electronics, Telecommunications and Information Technology*,
study program *Advanced Microelectronics*

Thesis Advisors
Prof. Dr. -Ing Andreas Koch
Dr. -Ing Andreas Engel
Conf. Dr. -Ing Zoltan Hascsi

Student
Brînduşa Mihaela Damian

2018

Statement of Academic Honesty

I hereby declare that the thesis *Implementation and Evaluation of a Hardware-Accelerated EtherCAT Slave Protocol*, submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of *Engineer* in the domain of Electronics, Telecommunications and Information Technology, study program *Advanced Microelectronics* is written by myself and was never before submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations or measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations or measurements. I understand that data faking is an offence punishable according to the University regulation.

Bucharest, September 2018.
Student: Damian Brînduşa Mihaela

.....

Table of contents

List of figures

List of tables

Acronyms

ASIC	Application Specific Integrated Circuit
AUI	Attachment Unit Interface
AXI	Advanced Microcontroller Bus Architecture
BRAM	Block RAM
CAN	Controller Area Network
CGRA	Coarse-Grained Reconfigurable Array
CoE	CAN application protocol over EtherCAT
CRC	Cyclic Redundancy Check
DIP	Dual In-Line Package
DSP	Digital Signal Processing
ENI	EtherCAT Network Information File
ESC	EtherCAT Slave controller
ESI	EtherCAT Slave Information Files
FCS	Frame Check Sequence
FCS	Frame Check Sequence
FF	Foundation Fieldbus
FoE	File Access over EtherCAT
FMC	FPGA Mezzanine Card
FMC	FPGA Mezzanine Card
FPGA	Field Programmable Gate Array
GMII	Gigabit Media Independent Interface
HART	Highway Addressable Remote Transducer
HDL	Hardware Description Language
ILA	Integrated Logic Analyzer
IP	Internet Protocol
IOBUF	Input/Output Buffer

IP	Intellectual Property
JTAG	Joint Test Action Group
LAN	Local Area Networks
LUT	Look-Up Table
LVDS	Low Voltage Differential Signaling
MAC	Media Access Control
MDIO	Management Data Input/Output
MDII	Medium Independent Interface
MII	Media-Independent Interface
MIIM	Media Independent Interface Management
MMCME2	Mixed-Mode Clock Manager
MMCM	Mixed-Mode Clock Manager
OSI	Open Systems Interconnection
PC	Personal Computer
pcap	Packet Capture
PL	Programable Logic
PLC	Programmable Logic Controller
PS	Processing System
RAM	Random-Access Memory
RGMII	Reduced Gigabit Media Independent Interface
SDK	Software Development Kit
SOES	Simple Open EtherCAT Slave
TEMAC	Tri-Mode Ethernet Media Access Controller
UART	Universal Asynchronous Receiver-Transmitter
VLAN	Virtual LAN

Chapter 1
Introduction

1.1 Project Motivation

Before the 1980s, when the Fieldbus Technology was introduced, the main communication technology in industrial systems was an analog, centralized control system [10]. In those systems, the measured data was sent as analog signal (e.g., 4 to 20 mA) via twisted cables to a control room, where the information was prossesed and sent back control data back. This approach requires many cables, thus implies high cost. An alternative are distributed systems, in which each actuator or sensor has the capability to individually process measured data, take control decisions and communicate to other devices in the network. The fieldbus technology provides this approach [11]. Digital communication replaces the analog one, providing a better accuracy and lower costs, as digital-analog converters are not required anymore [12].

Some examples of fieldbus technology are according to [10] the "WorldFIP, Foundation Fieldbus (FF), Controller Area Network (CAN), lonworks, DeviceNet, Profibus-DP, Highway Addressable Remote Transducer (HART) and Interbus". Recently, instead of using "proprietary" fieldbus PHYs, companies tend to implement Ethernet based systems [13]. Those protocols are called Industrial Ethernet and have some additional features compared to the common one, which is widely used in local area networks. They are caused by different requirements, e.g. providing a deterministic behavior in an industrial environment [14]. Additionally, according to [13], using the Ethernet protocol in industry is not trivial, because it poses high connection costs, high overhead when sending small packages and bad real-time capabilities due to software stacks. Some real time Ethernet protocols available on the market are Powerlink, EtherCAT, EtherNet/IP or PROFI-NET [15].

The EtherCAT technology can be defined as "Ethernet on the fly"[16]. The slaves receive data and pass information to the next slave while the frame goes through the device. Thus, the delay of the telegram is small (a few nanoseconds) [17]. Some other advantages of this protocol are that switches are not necessary and it does not require difficult handling of the MAC or IP addresses [18]. Besides advantages, the protocol poses also some drawbacks that must not be overseen. According to [15], one problem when using the EtherCAT technology is that it is not suitable for event-driven applications. In time-driven operation, messages are generated and sent within a certain period. This case is covered well by EtherCAT. However, when aperiodic behavior is required, the protocol leads to long cycle times.

Taking into consideration the above mentioned advantages, EtherCAT slaves are required in automation industry. Some software as well as hardware solutions are already available on the market. Some controllers used in automation industry might require high computation power. In this case, a hardware solution might be more suitable as computation resource compared to a software one. In order to have high flexibility, designing an Application Specific Integrated Circuit (ASIC) would be a good choice, which would however require extremely high (non-recurring) production costs. Only companies that sell enough chips, like Intel, afford this technology. As the number of the required EtherCAT Slaves might not cover the production costs, designing an ASIC might not come through as the best approach. A good compromise between efficiency and expenses would be using reconfigurable technologies, like a Field

Programmable Gate Array (FPGA), which gives the designer enough flexibility to obtain the desired microarchitecture without the need of selling a lot of chips to compensate the high production costs of an ASIC.

Some open source examples provide an implementation of an Ethernet Control Module using the processing system of some FPGAs provided by Xilinx [19]. However, the performance is lower in comparison to a possible full programable logic (PL) based design, which allows a tighter coupling to the PHY. Due to the fact that EtherCAT processes data on the fly, a PL based solution would be more suitable. For the last mentioned approach, some solutions are already available on the market as intellectual property (IP) cores, like the ones provided by the Beckhoff Company [20] or the HMS Industrial Network [21]. They give the designer the flexibility of choosing certain features of the module but they do not provide access to the HDL (Hardware Description Language) files. Thus, the flexibility of the design is limited. Additionally, in order to be able to use the netlists for production purposes, a payed license is required.

Besides the Field Programmable Gate Array (FPGA), described in Chapter 2.4, another reconfigurable technology that receives much interest in the academic area is the Coarse-Grained Reconfigurable Array (CGRA), described in Chapter 2.5. In comparison to the FPGA, which processes data at bit level, the later computes data at word level, thus requiring less configuration information. Therefore, the time needed to load a certain configuration is less and the principle of partial dynamic reconfiguration becomes feasible. More clearly, the user can change the configuration on a CGRA in each cycle, being able to reuse the same logic for different calculations.

Referencing the above mentioned considerations, a suitable solution in case of distributed systems requiring high computation power would be using CGRAs on each field device as computation resource. Because usually in distributed systems processes are interconnected, meaning that the process of a device in the network might influence another node of the network, having communication capabilities between the physically separated CGRAs is quintessential. In industrial automation, real time protocols are required. Compared to other such protocols, EtherCAT has some advantages. For example, it is faster than PROFINET IO and has a master less complex than EtherNet/IP [22].

An example network is shown in Figure 1.1. In a company fabricating printed circuits boards, there is an entire process with various machines, from cutting the boards, drilling them, printing the circuits or coating them. In order to control the process, sensors and actuators are required. For example, when coating a board, the speed of the running band is essential. A camera could check the quality of the coated board and depending on that, the speed could be adjusted. For such calculations, the computation parallelism offered by a CGRA could be used to detect faults faster. As the process is a line production, it is highly probable that process variations in one machine would influence the next machine. Thus, the necessity of sending data between the machines arises. EtherCAT could be used.

The goal of this project is designing Hardware Description Language (HDL) modules for an EtherCAT slave that is able to communicate with a CGRA. Thus, the flexibility of designing an interface between the CGRA and the EtherCAT slave is quintessential. As the license based solutions do not give access to the source files, it is required to design an own EtherCAT slave, which motivates the necessity of this project. The functionality and performance of the developed modules will be analyzed, trying to obtain an optimized design.

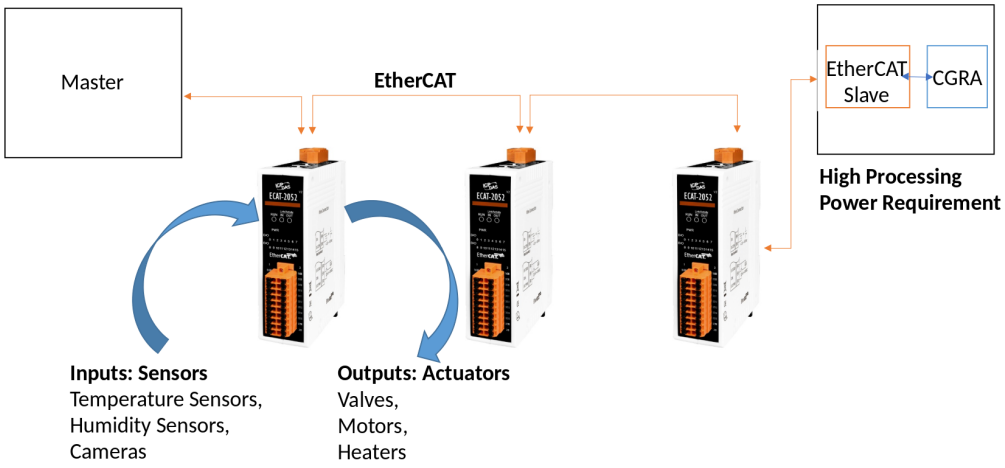


Figure 1.1: Example architecture of a simple EtherCAT network. (ECAT-2052 [1])

1.2 Task Description

The goal of this project is implementing a PL-based EtherCAT slave that would fulfill basic tasks of the protocol stack, such as topology scan or data exchange. A review of the already available implementations of EtherCAT slaves shall be done. Additionally, the project must conduct the concept and implementation of a lightweight interface to the application layer of the slaves. Functional evaluation of the implemented modules is required as well as its performance characterization.

1.3 Required Hardware and Software Tools

The architecture of the required system can be visualized in Figure 1.2. In order to fulfill the task, an Ethernet FMC board and a Zedboard are used. The first one provides four Ethernet plugs and four PHYs, which implement tasks of the first OSI layer. Although the Zedboard already provides one Ethernet port, the Ethernet FMC board is needed in order to implement and test a linear EtherCAT requiring at least two slaves with two PHYs each. It is connected to the Zedboard through an FPGA Mezzanine Card (FMC) connector. A Zynq 7 FPGA is mounted on the Zedboard and this will be used to implement two EtherCAT controllers, which have to fulfill requirements of the upper part of the second Open Systems Interconnection (OSI) layer, as the Media Access Control (MAC) is part of the Ethernet protocol. The EherCAT Master is implemented on the compuer using the open source “Simple Open EtherCAT Master ”project [23].

The main tool used for this thesis was Vivado, version 2017.2. The Software Development Kit (SDK) tool was also used for loading the design implying the processing system of the Zedboard and described in Chapter 3.1. The Wireshark tool was used for analyzing network traffic and testing the implemented designs.

1.4 Document Structure

The document starts in Chapter 2 with an introduction of technical terms and theoretical aspects of the project. As the EtherCAT technology is based on the Ethernet protocol, the project first implements an Ethernet slave, which is described in Chapter 3. Further on, an EtherCAT slave based on the AXI Ethernet IP is implemented and evaluated in Chapter 4.

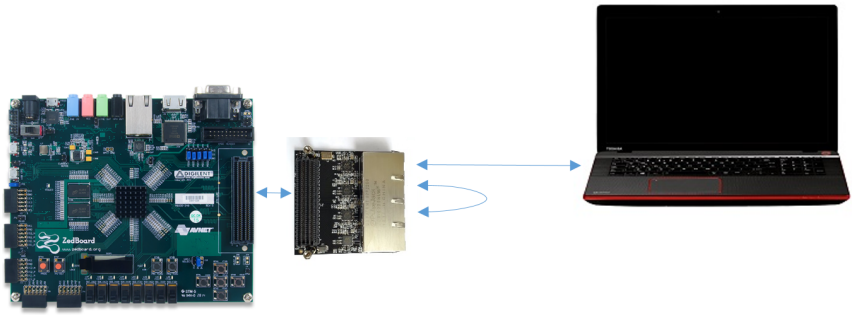


Figure 1.2: The hardware used for this thesis: one Zedboard, one Ethernet FMC, one PC and connection cables. [2, 3, 4]

In the following Chapter the core provided by Xilinx is removed in order to improve efficiency. Chapter 7 provides a description, implementation and evaluation of an interface between the EtherCAT slave and the CGRA. The setup of a linear EtherCAT topology is described and tested in Chapter 5. Last but not least, a summary of the implemented designs and their performance is given in Chapter 8. Additionally, possible further improvements of the current project are proposed.

Chapter 2
Technical Backround

This Chapter provides the technical background required for implementing the tasks.

2.1 Ethernet

The Ethernet Technology was initially designed by the Xerox Corporation in 1970 [24] and is a Local Area Networks (LAN) protocol. Two types of Ethernet protocols can be distinguished, the clasic one, which can be used at rates of maximum 10 Mbps, and the switched one that can reach 10 000 Mbps. The later version, which uses a switch to handle high loads, is the most common LAN protocol used today [25].

Ethernet implements features of the first two layers of the OSI Model. In the physical layer, fiber optics, coaxial cable or twisted pair can be used. The protocol imposes a maximal length for its segments, which can be overcome by using repeaters [25]. This rises a problem in automation industry, where field devices of a fabric are sometimes placed far away from each other. The Ethernet implements also the MAC sublayer. Its main task is to allocate a multiaccess channel to its users.

Further on, the frame format is shortly described, as it will clarify better the EtherCAT frame format introduced in section 2.2. According to IEEE 802.3, the classic Ethernet data contains the following information:

The field containing information regarding the length of the frame can also be used to indicate the type of message, if its value is greater than 0x600. For example, the value 0x0800 signals an IPv4 packet.

The checksum, or the Frame Check Sequence (FCS) field is required in order to detect errors in the transmitted message. It is based on the CRC computation.

The fast Ethernet uses the same frame format, but reduces the bit time. The gigabit Ethernet is also compatible with previous Ethernet versions and has an unacknowledged datagram service. The definition of the frame size, its format and the adressing scheme remained the same [25].

The Ethernet FMC board, having four Ethernet connections, provides 4 Gigabit Ethernet PHYs from Marvell, namely 88E1510. The PHY modules fulfill tasks of the first OSI layer and send data to the data link layer controller through the Reduced Gigabit Media Independent Interface (RGMII) interface. The controller must bring up correctly the PHYs according to its needs. Control and status data are sent using a Management Data Input/Output connection.

The media-independent interface (MII) was defined in Fast Internet and is equivalent with

	Preamble	Start of Frame	Destination Address	Source Address	Length	Data	Pad	Checksum
Nr. of Bytes	7	1	6	6	2	0 to 1500	0 to 46	4

Table 2.1: Ethernet Frame Format according to IEEE 802.3

the Attachment Unit Interface (AUI). It connects the PHY device to a controller [26]. Current variants are [27]:

- Reduced Media Independent Interface
- Gigabit Media Independent Interface
- Reduced Gigabit Media Independent Interface
- Serial Gigabit Media Independent Interface
- or 10 Gigabit Media Independent Interface

As stated above, the interface used by the Marvell 88E1510 is RGMII. It reduces the number of pins. If the GMII interface needs for receive as well as transmit busses 8 bits, the RGMII interface uses 4 bits for receive and transmit data, which is compensated by the double data rate [28]. Table 2.2 explains the functionality of the required signals in the RGMII interface and is based on [29].

For the Marvell 88E1510, the following characteristics apply according to the datasheet [29]:

- The allowed frequencies for the clock signals are 125 MHz, 25 MHz or 2.5 MHz.
- The data buses are double data rate. The first nibble corresponds to the least significant bits [3:0] of the sent or received data.
- In case of sending data, the enable control signal must be set on positive edge

An important theoretical aspect regarding the RGMII interface represent the timing constraints [5]. As described by the Reduced Gigabit Media Independent Interface (RGMII) standard, a clock skew is required in order to correctly sample the data. It must be introduced either by the receiver, the transmitter or the PCB traces, as shown in Figure 2.1. According to [30], if the transmitter has a data to clock output skew of 0 ps, the receiver has a typicall 1.8 ns data to clock input skew.

Besides the RGMII interface, another important connection is given by the MDIO communication, used for management purposes. Registers of the PHY are read and written using this signals. The MDIO interface, also known as Media Independent Interface Management (MIIM) is composed of one clock pin and one data pin for sending as well as receiving data [6]. Information is sent serially.

The data sent through the MDIO must comply with the frame given in Table 2.3 [6].

In order to establish a correct communication, some further theoretical aspects must be taken into consideration:

- The preamble contains 32 bits of one.

Signal Name	Bit Width	Description
rgmii_txd	4	RGMII transmit data.
rgmii_tx_ctl	1	RGMII transmit control.
rgmii_txc	1	RGMII transmit clock.
rgmii_rxd	4	RGMII receive data
rgmii_rx_ctl	1	RGMII receive contol
rgmii_rxc	1	RGMII receive clock

Table 2.2: RGMII Interface

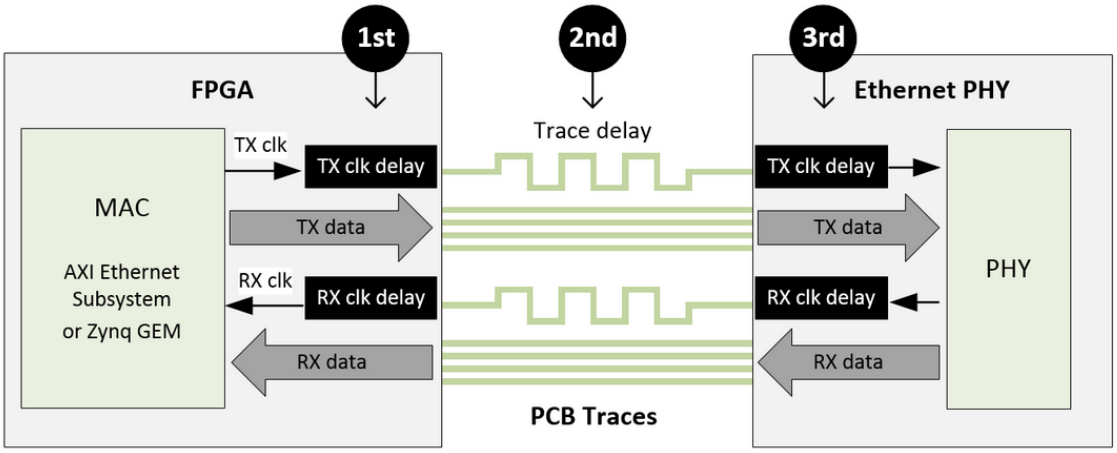


Figure 2.1: Possible clock skew sources.[5]

Preamble	Communication Start	Read or Write	PHY Address	Register Address	Turnaround	Data
32 bits	2 bits	2 bits	5 bits	5 bits	2 bits	16 bits

Table 2.3: MDIO signal pattern.

- The start is signaled through a low signal followed by a high one.
- The write command has the same pattern as the start signal, while the read command signal has opposite polarity.
- When setting the PHY address, the user must take into consideration that the PHYs on the Ethernet FMC board are hard-wired with the address 0.
- The registers of the Marvell 88E1510 are organized in pages. In order to write data into the correct register, the correct page must be selected by writing register number 22.

Some further important aspects regarding the connectivity between the Zedboard and the Ethernet FMC are:

- The board is connected to the Zedboard through the FMC connector.
- The signal pins include according to the schematics only clock signal, reset, RGMII interface and MDIO connection.
- The power supply is given by the Zedboard, which must provide 2.5 V to the Ethernet FMC.

Further on, an already available solution for implementing the Ethernet link layer is described, as this will be used for implementing the EtherCAT slave required by the task. As the used hardware has a Xilinx FPGA, a solution provided by this vendor is searched. The AXI Ethernet Subsystem (complete name AXI 1 G/2.5 G Ethernet Subsystem) is an IP hard macro provided by Xilinx. It can be parametrized by the designer, who does not have access to the HDL description though. It eases the workload of the designer in implementing the data link layer of an Ethernet-capable module. The information of this Section is based on the datasheet of the IP [31].

The block diagram of this macro is shown in Figure 2.2.

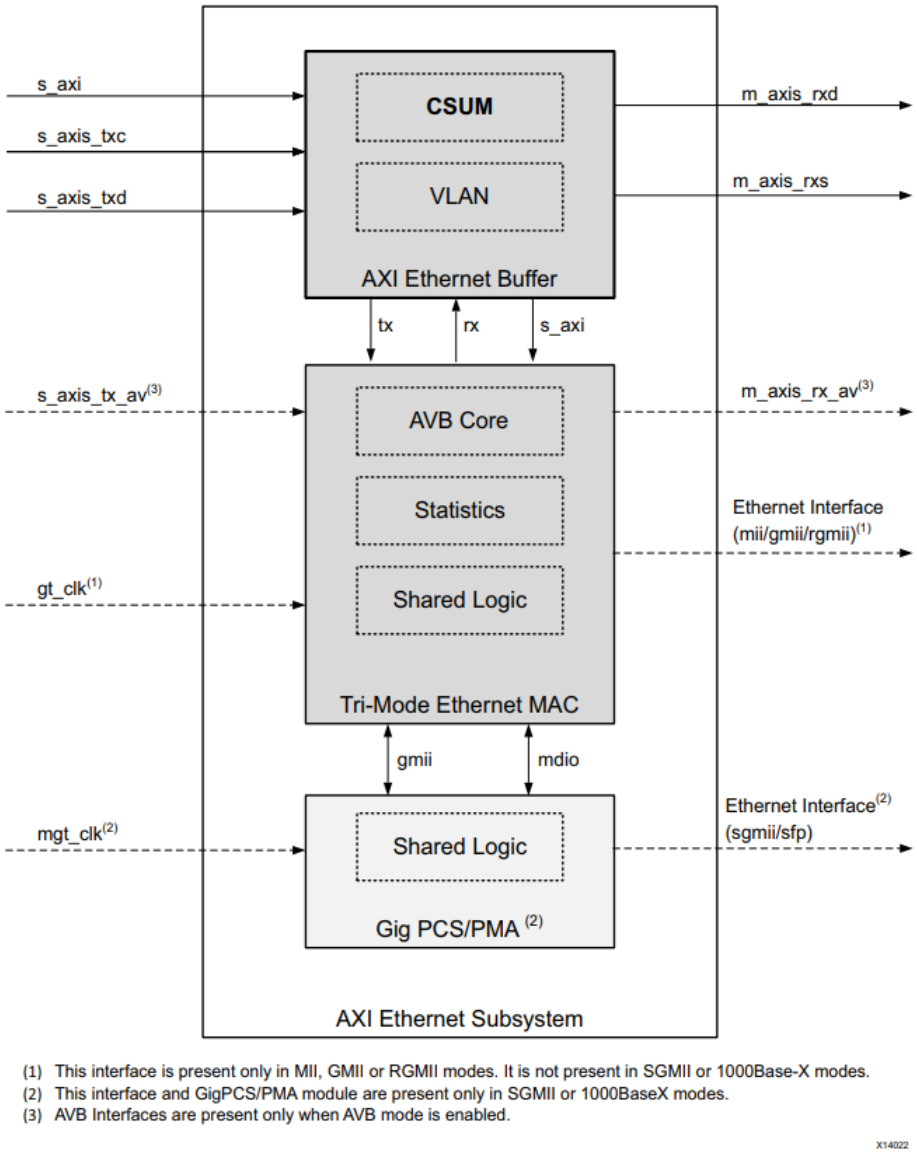


Figure 2.2: Block Diagram of the AXI 1 G/2.5 G Ethernet Subsystem [6]

The module can be controlled through the AXI Lite bus. The AXI Stream Bus is used in order to send and receive data from the PHY, via the RGMII. In order to write the registers of the PHY module, the MDIO connection is used, part of the Ethernet interface shown in Figure 2.2. The AXI Ethernet Buffer has capabilities such as TCP/UDP Checksum offload or TX and RX VLAN stripping. As the name suggests, the Tri-Mode Ethernet MAC provides the Ethernet Media Access Control. This is the most important component of the subsystem and requires a purchased license.

As stated above, the AXI 1G/2.5G Ethernet Subsystem is a macro, which means, the designer can select between several function modes of the IP. For the physical interface, the user can choose between: MII (Media-Independent Interface), GMII (Gigabit Media-Independent Interface) and RGMII (Reduced Gigabit Media-Independent Interface).

The user can also adjust the frequency of the AXI communication or he can choose to enable VLAN tagging, stripping or translation.

Another important parameter that can be set is choosing the source of the reference clock.

After generating the IP, the user has the option of using an example design provided by Xilinx. This creates a design with the architecture shown in Figure 2.3.

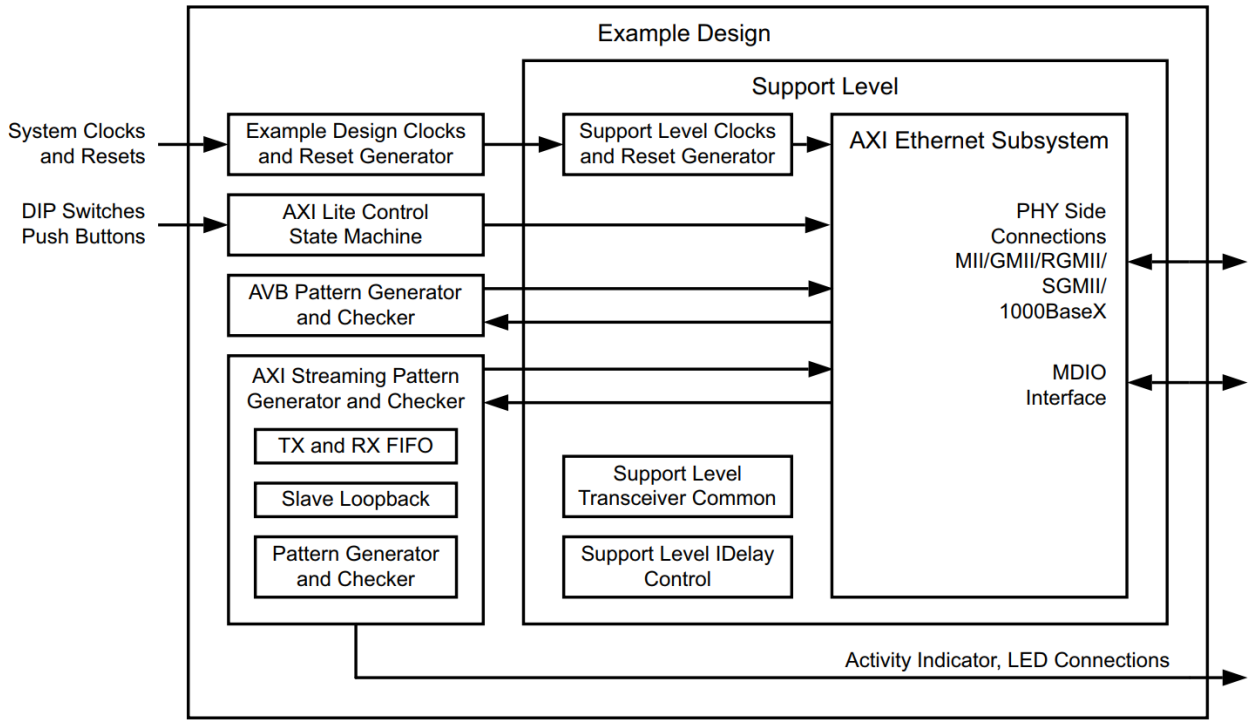


Figure 2.3: Architecture of the AXI Ethernet Submodule Example Design. [6]

The "Example Design Clocks and Reset Generator", as the name suggests, generates the clocks required by the design.

The "AXI Lite Control State Machine " sends through the AXI Lite interface control commands to the IP and writes its registers according to the user needs.

The example design comes also with a testbench. By setting up correctly some input signals, the design can have two different working modes: "DEMO" mode and "BIST" mode. In the "DEMO" mode, the testbench sends packets to the IP and the data is passed through the AXI Stream interface to a FIFO. The data is than looped back and sent through the IP to the testbench. In the "BIST" mode, the loopback is on the PHY side, while the hardware is able to generate Ethernet packets.

The "AXI Streaming Pattern Generator and Checker" module provides a FIFO for storing the incoming packets. It communicates with the IP via the AXI Stream interface. It is also able to generate packets if the "BIST" mode in testbench is selected.

2.2 Field Bus Technology and the EtherCAT Protocol

According to [32], the "Fieldbus is a digital two-way multidrop communication link between intelligent field devices". Before introducing this technology, analog signals were used in industrial automation. Common standards for sending data from sensors in the field to the control units were 3 to 15 psig or 4 to 20 mA [33]. However, in noisy environments, analog signals are prone to loss of accuracy, which was an important reason for replacing them with digital ones.

HART is an example of an industrial network, which uses digital as well as analog signals. The first one is superimposed on the second one. Usually the process variable is sent using the analog signal, while configuration is done using the digital one [34]. When it comes to transmission of digital signals, PROFIBUS is a faster solution. Its origin dates back to 1987 and until now, several versions were developed, which makes it easy to be adapted to different types of devices. For example, in hazardous environments, PROFIBUS PA can be used, while

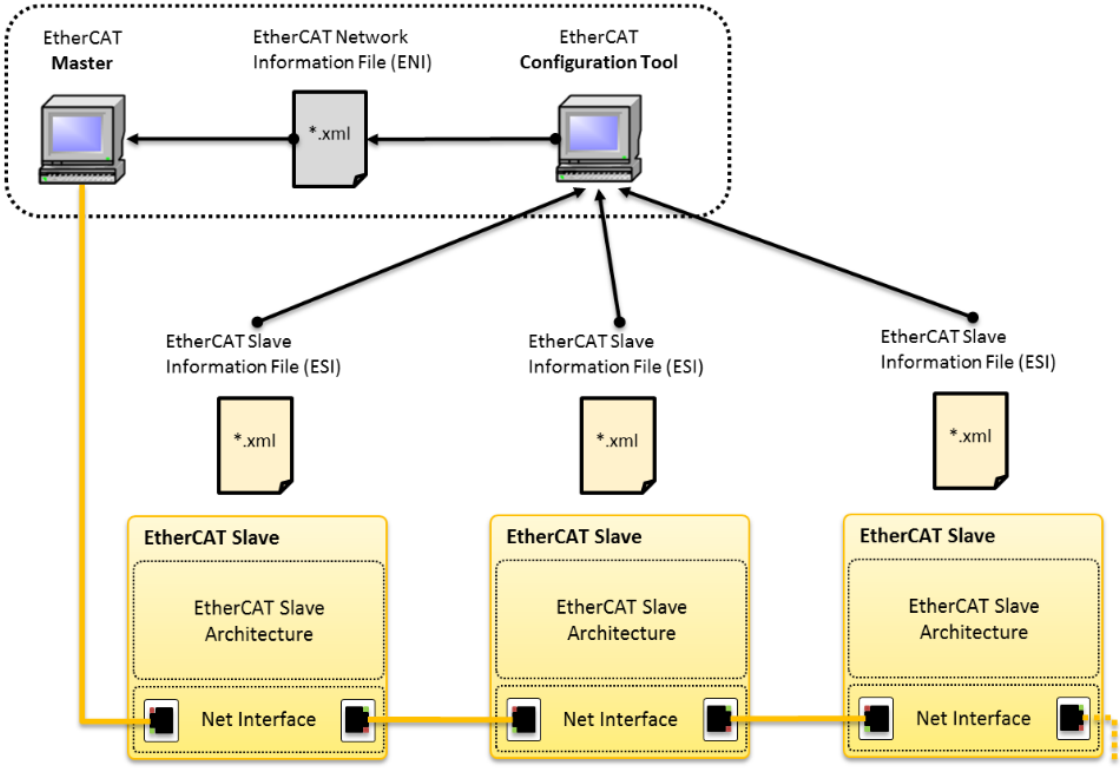


Figure 2.4: Example Architecture of an EtherCAT Network. [7]

PROFIdrive was developed for motion control applications, where the precision of the motors is essential [35]. Compared to HART, PROFIBUS provides more complex diagnosis [34]. In 1995 a new important fieldbus technology was defined, namely Foundation Fieldbus (FF) [36]. While HART is commonly used for configuration and viewing internal variables, FF can additionally implement real-time, digital, closed loop control [37]. Compared to PROFIBUS, it is a deterministic protocol **??**.

Using Ethernet in fieldbus technologies can bring advantages like low hardware costs or easy configuration compared to other competing protocols [38]. Some challenges when using Ethernet in industrial automation are performance and deterministic behaviour in real time applications [16]. Some examples of industrial Ethernet protocols are EtherCAT, EtherNet/IP, Modbus TCP, Powerlink, Profinet or SERCOS [38].

This thesis focuses on the EtherCAT protocol, which is based on the Ethernet protocol. Therefore, any Ethernet device can be connected to the switch port and in case only EtherCAT devices are present on the network, no switches are required.

Data is processed on the fly and the protocol has transmission rates of 2 x 100 Mbit/s in Fast Ethernet, Full-Duplex mode [16]. Through a clock synchronization mechanism, the protocol suports distributed clocks [16], providing high-precision synchronization [39].[39] gives as example two distributed devices with 300 nodes and 120 m cable length, where “the clocks are simultaneous to within much less than 1 µs of each other”.

When it comes to the network architecture, this protocol supports the line, tree, star or daisy-chain configurations, being very flexible [39]. An example architecture can be seen in Figure 2.4.

Each device has an EtherCAT Slave Information Files (ESI) provided by the vendor and this is used by the network master to generate the EtherCAT Network Information File (ENI) file. This files are not taken into consideration in this thesis. The shown network has a linear topology. In order to pass the frames through the network with minimal latency, each EtherCAT

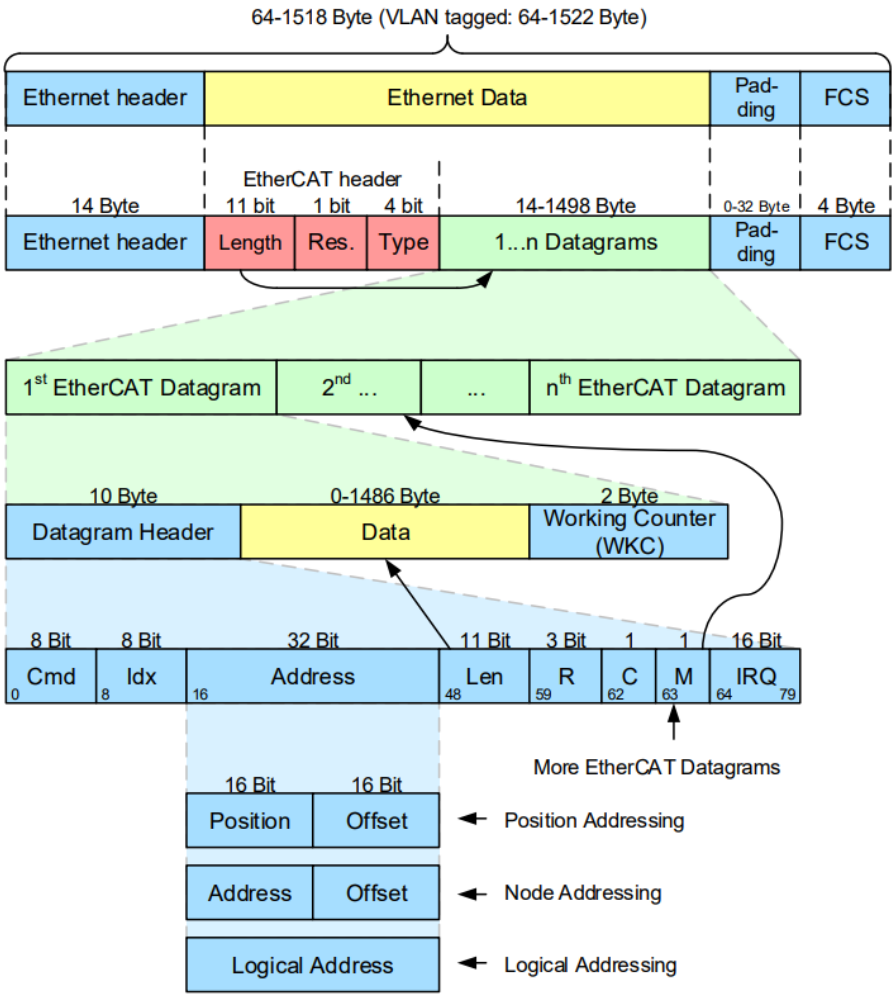


Figure 2.5: Structure of the EtherCAT frames.[8]

slave has two Ethernet ports. As each slave receives parts of the frame, it processes them on the fly and forwards the data to the next slave. The frame is based on the one used in the Ethernet communication. While Ethernet implements the physical layer as well as the MAC sublayer, EtherCAT implements the upper part of the second OSI layer. Therefore, the payload contains now several datagrams, each having its own header with control information.

EtherCAT packages can be recognized by checking the "Type" field of the Ethernet frame and it shall have the value 0x88A4. The format of an EtherCAT frame is shown in Figure 2.5.

The Ethernet Data consists of the EtherCAT Header and the Datagrams. Thus, the Ethernet header is followed by the EtherCAT Header consisting of two bytes. It gives information regarding the length and type of the message. The datagrams must have between 12 and 1498 Bytes. Each datagram consits of a header, data and a working counter. The header contains information regarding the datagram, such as the command type, length of the message or address. It is used by the data link layer controllers in order to process correctly the payload. The working counter is incremented or decremented by each slave depending on the command type. This field can also be used by the master in order to scan changes in the network topology. A full description of the frame can be found in [7].

2.3 Already available technologies for EtherCAT slaves

As the project proposes an implementation of an EtherCAT slave, a research regarding already available technologies must be conducted. The are several open source as well as license based projects available, which can be used in software or hardware solutions.

One example is the Simple Open EtherCAT Slave (SOES) project implemented in C programming language and available at [40]. It is an easy portable code, which can be used in embedded applications. It implements an EtherCAT state machine. Additionally, it provides mailbox interfaces, which are used according to [41] for asynchronous accesses. It also supports CAN application protocol over EtherCAT, where Controller Area Network (CAN) is another type of real time protocol, originally used in automotive industry [42]. Another supported protocol is File Access over EtherCAT (FoE).

There is also an already available solution for the widely used Arduino microcontrollers. ArduCAT is a board implementing an EtherCAT slave using the ATMega2560 processor and a LAN9252 adapter. It can be connected to an Arduino Mega board. To ease the development of the application code, users are given an open source code available at **ref40!** (**ref40!**).

As some controllers used in automation industry require high computation power, a solution implemented in hardware might be preferred. In case of Xilinx boards, two examples of EtherCAT slaves available for hardware designs are provided by the Beckhoff Company [20] or the HMS Industrial Network [21], both being license based. In case of the first solution, the user can configure some parameters of the Intellectual Property (IP). He can set for example the number of the PHY ports or the interface type (**RMII!** (**RMII!**), RGMII or Media-Independent Interface (MII)). Additionally, the user can choose between several types of interfaces between the EtherCAT Slave controller (ESC) and the processor. The size of the RAM is also configurable. A complete description of possible configurations is found in [**?**]. The second solution, provided by the HMS Industrial Network is called Anybus. It can be used for Zynq-7000 devices and reaches a maximum frequency of 143 MHz. It can be purchased as netlist and the IP is described at [43]. In case of prototyping projects, no fee is required.

2.4 Field-Programmable Gate Arrays

FPGAs are reconfigurable devices used for implementing digital designes. Basically, there are two categories of FPGAs: the ones that use lookup tables and the ones based on the multiplexor technology. The first type is the most common one and in this case, SRAM cells are used as main storage element [44].

The design configuration is defined in the content of the lookup tables (LUTs) and in the routing elements. This data is usually loaded from a FLASH memory that stores all the desired configurations [44].

FPGAs consist of logic blocks that are connected in an “Island Style”. The device used for this project is a Zynq 7000. It has an ARM Cortex-A9 based Application Processor Unit (APU), while the programmable logic mainly consists of:

- 53200 LUTs
- 140 Block RAMs (BRAMs)
- 200 Input/Output (I/O)
- 106400 Flip Flops (FFs)
- 220 Digital Signal Processing blocks (DSPs)

- Clock Managment MMCME2_ADV

The LUTs can be used to compute logic functions, but also as distributed RAM. As an alternative storage elements, BRAMs can be used. They can be configured as single or dual port memory. In the second case the designer must take into consideration phenomena like address collisions.

As clock source, the FPGA receives a 100 MHz signal, which must be taken into consideration when defining timing constraints.

The tool used in this Project for obtaining the configuration bitstream and programming the FPGA was Vivado 2017.2.

2.5 Coarse Grained Reconfigurable Arrays

CGRAs can be used as a reconfigurable technology and high parallization can be obtained. Its strong advantage is being able to change the configuration in each cycle [45]. It consists of an array of processing elements (PEs) that process data at word level and are reconfigurable. This project focuses on the CGRA design proposed in the AMIDAR project, which is developed by the "Computer Systems" group of the Faculty of Electrical Engeneering, within the Technical University in Darmstadt [46]. The CGRA has the architecture shown in Figure 2.6.

The "Context Control Unit" contains a context counter, which works similar to a program counter and keeps track of the current loaded configuration. The "Condition-Box" can compute the truth value of complex expressions that can be further on used also as prediction signals. The "Context Memory" stores the configuration of the PEs[6].

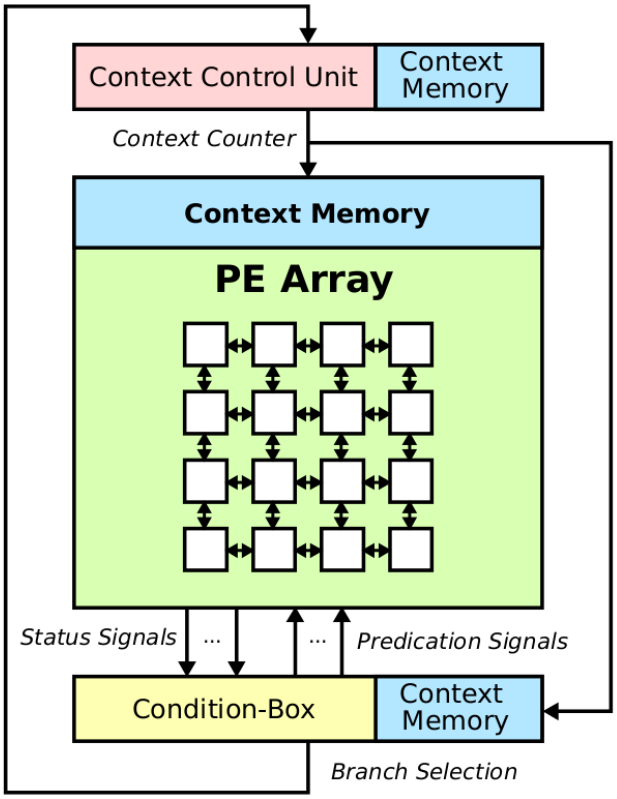


Figure 2.6: Architecture of the CGRA used in the AMIDAR project. [6]

Chapter 3
Designing an Ethernet Device

Before implementing the EtherCAT protocol, a basic Ethernet communication is established between the Personal Computer (PC) and the FPGA. The data is sent from the computer using the Packet Capture (pcap) library. The other endpoint is implemented on the FPGA. The design is based on the AXI Ethernet Subsystem and its parameters are explained in Section 2.1.

An already available open source project uses the processing system of some Xilinx FPGAs (Zynq-7000 and Zynq UltraScale+). This solution is described shortly in the first Section of this Chapter. As the final project shall not use the processing system, this design will not be described in detail. Some aspects that helped during implementation of the final project are highlighted however.

Xilinx also provides an example design implemented on the Programable Logic (PL). This will be used as reference design in order to implement the task of this thesis. As it was not developed for Zedboard and for the PHYs on the Ethernet FMC board, some modifications of the HDL files are required. These are explained in detail in Chapter 3.2. The example design gives some flexibility to the user, which requires additional logic. As the requirements for this thesis are clearly defined, this flexibility is not needed. The code is optimized by removing some of them and the corresponding modifications are explained in Chapter 3.3.

3.1 An available solution using the processing system

An example of an open source project implementing an Ethernet capable device can be found in the GitHub repository [19]. Although the final project should not involve the processing system, this will be used as an example to get a better understanding of how the system should work.

The architecture of the design can be visualized in Figure 3.1. Through an AXI Interconnection bus, the processor sends and receives data from four AXI Ethernet Subsystems, which are connected to the four PHYs. The example project is able to support only one port at a time. In order to select the desired port, the platform_config.h file must be modified, by changing the "platform_emac_baseaddr " parameter.

For testing the design, setting a static IP in the same subnet as the board is required. Additionally, some hardware adjustments must be conducted. The proper adjustable voltage VADJ is selected by putting a jumper on J18 for the desired value (in this case, 2.5 V). The jumpers M02 to M06 must be set correctly, depending on how the code is loaded into the FPGA (for this project, all jumpers are connected to GND and the FPGA is programmed through Joint Test Action Group (JTAG)).

In order to read the Universal Asynchronous Receiver-Transmitter (UART) messages, a terminal, like putty for example, is required. The FPGA is programmed using the SDK tool from Xilinx and the application is ran. This will output some status information in the terminal. There are more ways of testing the design, like using the ping command in a shell, the program Packet Sender or writing a script and sending packets using the pcap library. Wireshark was used to analyze the network traffic.

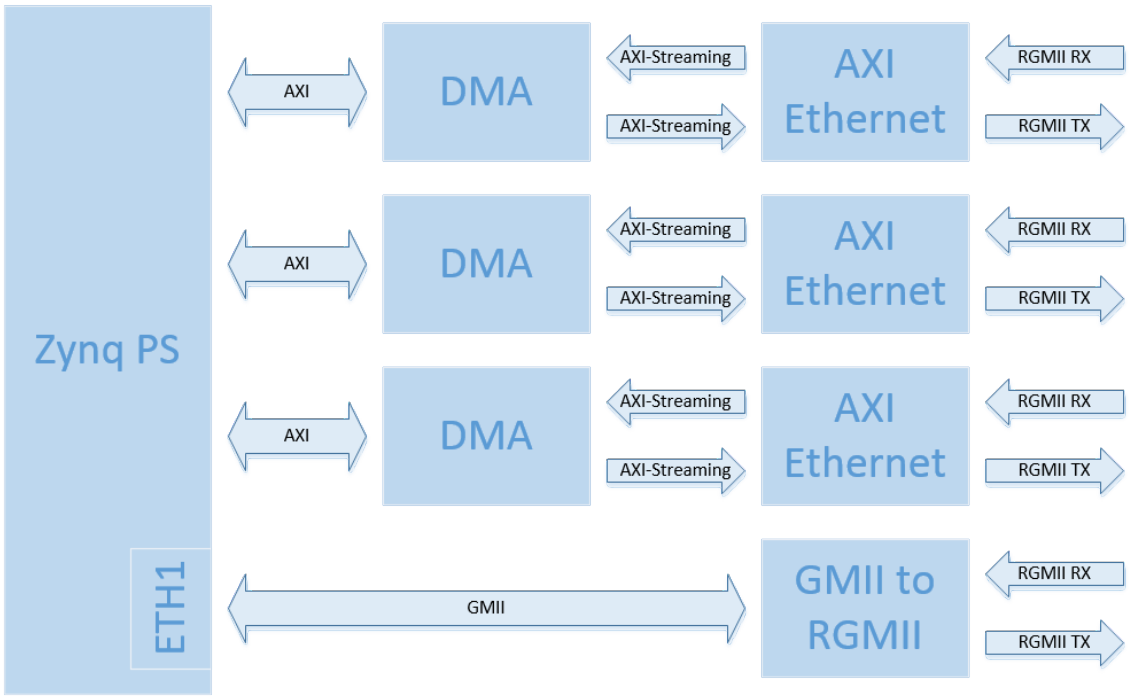


Figure 3.1: Architecture of the PS-Based solution. [9]

Although the final project must be implemented on the reconfigurable logic of the FPGA, using this example was useful in solving the following problems of the final project:

1. Setting up correctly the AXI Ethernet Submodule
2. Solving the error: IDELAYCTRL cells "~" and "~" have same IODELAY_GROUP "~" but their RST signals are different
3. Fast assignment of output pins

When implementing the solution in PL, one task was setting the control registers of the AXI Ethernet submodule correctly. As the Processing System (PS)-based example project was functional, it was used to analyze the commands sent to the AXI Subsystem module and they were replicated in a control state machine in hardware. This was done using an ILA module.

However, it must be noted that the final working version of the project needs less transfers to the IP in order to set it up correctly. Table 3.1 shows only some of the transfers conducted by the Zynq processor. Some of the accesses were needed in the working PL version, like writing the "TEMAC Receive Configuration Register" or resetting the receiver. Others however, like reading the interrupt status were not necesserily required.

As it can be seen in Table 3.1, some of the PS transfers are redundant. There are for example three read accesses from address 404_{16}, and this table enumerates just a part of all the sent commands. Another such example is a double read-write access to address 0.

The error mentioned above at point "2)" was flagged as the linear EtherCAT Topology was implemented (Chapter 5). Having more EtherCAT slaves meant instantiating more AXI Ethernet submodules, which raised this error. The problem was choosing whether to include shared logic in the core or not, an aspect discussed in the corresponding Chapter.

As stated above, this design based on the processing system was used only as example. The task is however implementing a PL-based design, which is described in the following Section.

Read/Write	Address	Data	Meaning
R	0XC	0X0000_01E4	Rd. Interrupt Status
R	0X404	0X1000_FFFF	Rd. TEMAC Receive Configuration Reg.
W	0X404	0X0000_FFFF	Set Pause Frame Ethernet MAC Address
R	0X10	0X0000_0000	Rd. Interrupt Pending Reg.
R	0X404	0X0000_FFFF	
R	0X408	0X1000_0000	Rd. TEMAC Transmit Configuration Reg.
W	0X404	0X1000_FFFF	Initiate a receiver reset
R	0X40C	0X6000_0000	Rd. TEMAC Flow Control Configuration Reg.
W	0X40C	0X6000_0000	Setup TEMAC Flow Control Configuration Reg.
R	0X0	0X0000_0000	Rd. Reset and Address Filter register TEMAC
W	0X0	0X0000_0000	
R	0X0	0X0000_0000	
W	0X0	0X0000_0000	
R	0X404	0X1000_0000	
R	0X408	0X1000_0000	
R	0X708	0X0000_0000	
W	0X708	0X0000_0000	Setup Frame Filter Control

Table 3.1: Some of the transfers conducted by the Zynq processor to set up the AXI Ethernet Subsystem.

3.2 Obtaining a PL-based functional design based on the AXI Ethernet Subsystem

In the past years the design gap, defined by [47] as the difference between the number of transistors that can be integrated on a current chip and the ones that are used effectively by an engineer in a design, has increased. In order to overcome this issue, the level of abstraction in designing circuits has improved. One way of overcoming this gap is by using IP cores or macros. The AXI 1 G/2.5 G Ethernet Subsystem is an IP hard macro and it eases the workload of the designer in implementing the data link layer of an Ethernet-capable module. The information of this Chapter is based on the datasheet of the IP [31].

The IP can be parametrized by the designer and the appropriate variables for this project are explained further on. As written in the specification of the PHY present on the Ethernet FMC board [29], the Ethernet PHY is connected to the Ethernet MAC designed in the FPGA via the RGMII interface. Thus, this type of communication was chosen for the AXI Ethernet Subsystem. The frequencies for the AXI Lite and AXI Stream interfaces were set to 100 MHz.

Another essential adjustment is the source of the ref_clk signal, which is the reference clock used for controlling the delay of the RGMII signal. This is an important aspect which must be taken into consideration when instantiating more such IPs. Otherwise, the Vivado Tool will

flag errors. Analyzing the open source code using the processing system showed that in case just one IP is needed, the user has to select the "Include shared Logic in Core Option". In this case, the module expects a reference signal and generates an output clock signal. This shall be used as input for further instances of the same module, which must have the option "Include Shared Logic in IP Example Design" checked. An explanation of this fact is given also in the datasheet of the subsystem [31]: "When multiple subsystem instances are targeted for I/O in the same bank, IDELAYCTRL needs to be shared", where IDELAYCTRL is a module generating certain delays in the design.

The user can also choose to enable Virtual LAN (VLAN) tagging, stripping or translation. For this project, none of them were chosen. The selected options can be seen in Figure 3.2.

An Ethernet capable endpoint can be designed either by controlling this core via the processing system or via the reconfigurable logic. There is an already available open source example implying the Zynq processor, as presented in Chapter 3.1. Because in the end the design must be capable of processing packets on the fly, a PL based solution is preferred. One reason would be the required determinism in a real time system, which can be achieved better in a hardware solution. Another reason would be increased computation power due to enhanced parallelism. (Is the argumentation correct?) After selecting the desired parameters of AXI Ethrnet Subsystem and generating it, Xilinx offers an example project implemented on the PL. A short theoretical description of the example design was given in Section 2.1.

The first modification conducted on the example project and the most important was modifying the control state machine implemented in the "AXI Lite Control State Machine" module. It has the task to write the proper data to the registers of the AXI Ethernet Subsystem and PHY. The design example provided by Xilinx implements a state machine for configuring the communication with a virtual PHY. It was obviously not designed exactly for the Ethernet FMC board. Starting from the example control module, some adjustments were made.

In order to simplify the understanding of this module, instead of implementing a big state machine, several, smaller ones, were designed. The first one goes through some states according to the input control signals. It triggers some flags in order to start an AXI transfer. The second state machine implements the MDIO interface. It is used in order to write correctly data into the PHY registers. The user first needs to write the data into the "Write Data Register" of the Tri-Mode Ethernet Media Access Controller (TEMAC) module. In order to start a transaction on the MDIO line, the "Control Register" needs to be written accordingly. For a read transaction, the user writes the "Control Register" and then reads the received data from the "Read Data Register". This behaviour is shown in Figure 3.3. A third state machine is implemented to generate a correct AXI Lite Transfer, depending on whether it's a read or a write.

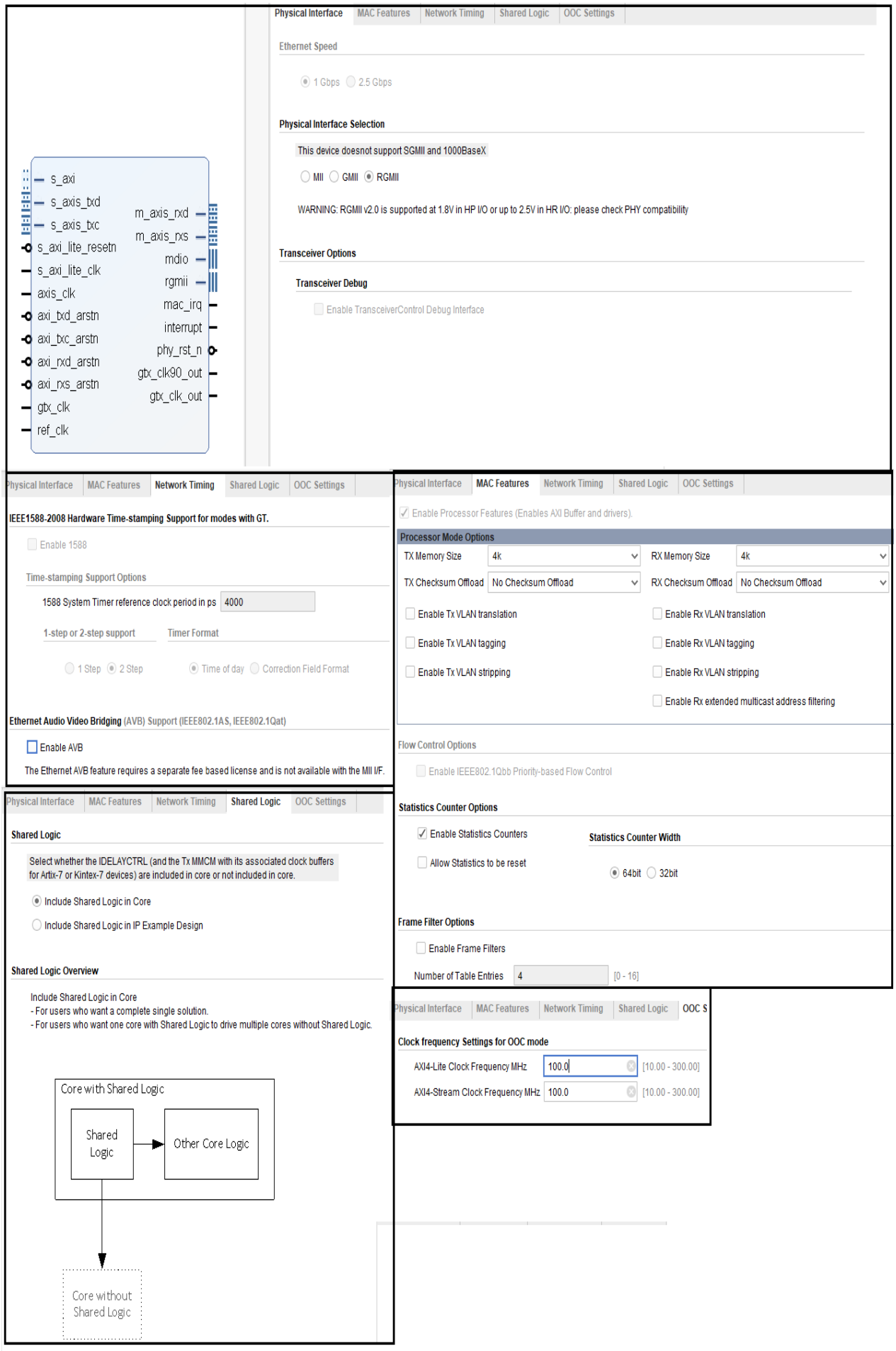


Figure 3.2: Settings of the AXI 1G/2.5G Ethernet Subsystem

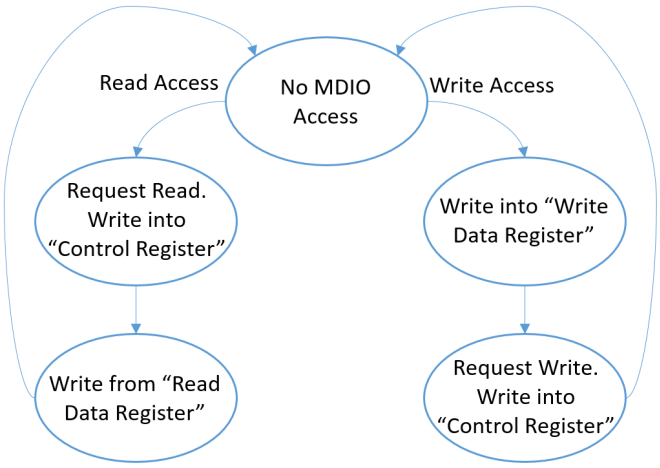


Figure 3.3: Read/Write MDIO accesses.

The original state machine was not functional. Some adjustments were made using the PS-based design as example and reading several forums. In order to set up correctly the Ethernet communication, the current state machine performs the following steps:

- Set the MDIO clock frequency and enable MDIO
- Read register zero of the PHY. Loop until the PHY responds
- Write in the register zero of the PHY 140_{16}; Select 1000 Mbps & Full Duplex
- Start auto-negotiation
- Disable receiver. Pause address = ffff_{16}
- Enable receiver.
- Enable transmitter
- Set the Unicast Address by writing the registers "Unicast Address (Word 0)"and "Unicast Address (Word 1)"
- Registers are organized in pages. Select page two of the register file in PHY
- By writing the "MAC Specific Control Register 2" register, set PHY so that the transmit clock is not internally delayed
- Select again page 0 of the register file in PHY
- Generate a copper software reset by setting bit 15 of the register 0 in PHY
- Set the MAC speed to 1 Gbps

While setting up the clock frequency of the MDIO communication, the equation 3.1 must be taken into consideration [31]. In this project, the AXI clock was set to 100 MHz and according to [31], f_{MDC} shall be lower than 2.5 MHz. For this project, a 2 MHz clock was used, thus the value 58_{16} was written into the "MDIO Setup Word" register. The first six bits correspond to the "Clock Divide" value, while the seventh bit enables the MDIO communication.

$$f_{MDC} = \frac{f_{\text{s\_axi\_aclk}}}{(1 + ClockDivide[5:0]) \cdot 2} \tag{3.1}$$

```
// start config signal
always@(posedge ref_clk or posedge sys_rst) begin
    if(sys_rst)
       start_config <= 1'b0;
    else if(&delay_reset)
        if(control_data!=4'd0)
            start_config <= 1'b1;
end

// control valid
always@(posedge ref_clk or posedge sys_rst) begin
    if(sys_rst)
       control_valid <= 1'b0;
    else
        if(&delay_reset) begin
            if(control_data_old!=control_data)
                control_valid <= 1'b1;
            else
                if(control_ready==1)
                    control_valid <= 1'b0;
        end
end

// control data
always@(posedge ref_clk or posedge sys_rst) begin
     if(sys_rst)
        control_data <= 4'd0;
     else
        if(&delay_reset) begin
            if(control_data==4'd0 && control_valid==0)
                control_data <= 4'd9;
            else
                if(control_data==4'd9 && multiple_counts_reg==5'd20 && control_valid==0)
                    control_data<=4'd1;
        end
end
```

Figure 3.4: Code replacing the DIP switches and push buttons.

In the original project, the user has to press some buttons in order to select the proper speed and configuration (DEMO versus BIST) of the design. The second modification conducted on the example project was setting up correctly these control signals. They are represented in Figure 2.3 by the "DIP Switches Push Buttons" signal. This is however not desired in the final project. By setting up the parameter TB_MODE = "DEMO", the testbench generates the proper signals corresponding to the DIP Switches and Buttons. The sequence of the signals was analyzed and the same approach was implemented in hardware. The added logic basically sets the speed of the RGMII communication and selects the DEMO mode. It generates a signal called "control_data" that is forwarded to the "AXI Lite Control State Machine" module. When in reset, this control signal is set to zero. As soon as the "control_data" is asserted a value different than zero, the "start_config" signals the state machine that a configuration process must be conducted. Each time the "control_data" changes its value, the "control_valid" becomes one and is reset by the "control_ready" signal. The code describing this behavior is shown in Figure 3.4.

As described by the RGMII standard, a clock skew is required. The AXI Ethernet Submodule inserts by default a clock skew on the transmission line. The PHY on the Ethernet FMC board also implements a clock skew. Thus, one of them must be disabled. Writing the register "MAC Specific Control Register 2" of the PHY disabled the skew inserted by the PHY.

Another important problem was writing data to the PHY via the MDIO line. After sending a write command, a read command from the same address did not yield the desired result. The received data was 0x1ffff. Varying the PHY address did not solve the problem. The approach that worked was requesting data from the PHY in a loop until the device responded. Most probable, this is neccessary as the AXI Ethernet Subsystem holds the "phy_rst_n" reset signal active for 10 ms and the PHY can not be accessed afterwards for another 5 ms [31].

A third step was modifying the clock scheme. The example design does not provide support

for the Zedboard. It was created for the Kintex board which has a 200 MHz Low Voltage Differential Signaling (LVDS) oscillator. The clock source on the Zedboard has however 100 MHz. A fast solution was instantiating a "Clocking Wizard" IP, using a Mixed-Mode Clock Manager (MMCM) primitive with 100 MHz input signal and a 200 MHz output clock, which was forwarded to the "Example Design Clocks and Reset Generator" module. In further optimizations of the design this IP must be removed and the "Example Design Clocks and Reset Generator" module shall be correctly adjusted. As selected in the AXI Ethernet Subsystem, the macro expects a 100 MHz clock on the AXI Stream as well as AXI Lite bus. Additionally, the reference clock "ref_clk" shall be set to 200 MHz for 7 series FPGAs, as the datasheet states [31]. Moreover, according to the datasheet, a 125 MHz clock, "gtx_clk" is required for the RGMII configuration to control the PHY.

The fourth step meant creating the constraints file. As stated above, the example design does not support the Zedboard. Thus, it does not offer a constraints file. In order to solve this problem fast, avoiding design mistakes, the PS-based version was analyzed and the signals required in the PL-version were selected.

A fifth modification was instantiating an ILA IP. Simulating the real PHY behavior in a testbench would have been time consuming. When configuring the AXI Ethernet Subsystem, analyzing the AXI Lite communication is enough. Thus, for debugging the problem only few signals are required and these can be easily analyzed during run time by using the ILA IP and the ChipScope tool.

A big problem in debugging the design was not being able to capture the RGMII and MDIO signals. These signals require strict delay and connectivity to the I/O Buffers and the constraints file of the AXI Subsystems prevents the user from observing them using ILA. A solution of this problem would be forwarding these signals to some output pins and measuring them using the oscilloscope. Luckily, the fault in the design was found before taking into consideration using the oscilloscope.

3.3 Optimizing the design

After obtaining a functional Ethernet slave and testing it, some further optimizations of the code can be done.

As described in Chapter 2.1, the "Example Design Clocks and Reset Generator" module manages the clock signals of the design. It expects a 200 MHz input clock and in the beginning this clock was generated by a "Clocking Wizard" IP. This is however suboptimal. Looking into the module provided in the example design, it can be found that a "MMCME2" cell generates the required signals. Adjusting its parameters and removing the additional "Clocking Wizard" IP is necessary.

When computing the new parameters, the following formula must be taken into consideration:

$$Desired\ Frequency = \frac{CLKFBOUT\_MULT\_F}{DIVCLK\_DIVIDE * CLKIN1\_PERIOD} \tag{3.2}$$

Table 3.2 illustrates some of the original parameters and the modified ones. When inserting the values computed in the middle column, the following error is flaged: "Error: [Unisim MMCME2_ADV-8] The calculated VCO frequency=500.000 000 MHz. This exceeds the permitted VCO frequency range of 600.000 000 MHz to 1600.000 000 MHz set by VCO-CLK_FREQ_MIN/MAX". Multiplying the 100 MHz input clock with 5 flagged the error. The parameters of the third column generated the clocks without problem and this optimization

Parameter	Original Values	Illegal values	Correct Values	Output Frequencies
CLKIN1_PERIOD	5	10	10	
CLKFBOUT_MULT_F	5	5	10	
CLKOUT0_DIVIDE_F	10	5	10	100 MHz
CLKOUT1_DIVIDE	8	4	8	125 MHz
CLKOUT2_DIVIDE	5	2.5	5	200 MHz

Table 3.2: Modifying the parameters of the "MMCME2" cell.

removes one clock management resource in the design.

Another optimization is removing the logic that emulates the DIP switches and push buttons. In the example project the user has to select between certain functional modes by pushing some buttons. Based on the simulation given by the testbench provided by Xilinx, the same behaviour was emulated in hardware in Section 3.2. This is however redundant. This additional logic must be removed and the control module is updated in such a way that it sets up the AXI Ethernet Submodule and the PHY immediately after reset, without taking into consideration any commands from user.

A third optimization implies removing most of the steps of the control state machine bringing up the AXI Ethernet Subsystem, thus speeding up the initialization phase. The current version only enables the MDIO communication, waits for a response from the PHY, disables the delay on the transmission line, generates a soft reset and sets the speed of the RGMII communication.

After implementing this changes, the design size reduces by 292 Look-Up Tables (LUTs), 291 registers and one clock management unit.

3.4 Design Evaluation

The functionality of the design is tested directly on hardware. In order to send Ethernet packets from the PC to the FPGA and check its response, the pcap library is used. The network traffic is analyzed using Wireshark. The design is evaluated as functional if the received data is equal to the sent one, having swapped the destination and source addresses.

Figure 3.5 shows the sent and received data prooving the functionality of the design. As expected, the addresses are swaped correctly while the rest of the data was received unchanged.

An important aspect during testing was observed. When sending small packages, with less than 46 bytes of data, the message is padded with zeros to be in accordance with the minimum allowed length. This padding can be disabled according to [31] by setting correctly the bits 25 and 29 of the "TEMAC Receive Configuration Word 1 Register". By choosing whether to perform Length/Type field check and to strip the FCS field, the AXI Ethernet module will pad zeros or not. If both of them are zero, the padding is not passed to user. In order for this to work, the Type/Length field must be populated with length information. While designing the module, this theory was tested and the result confirmed the theory.

According to the data recorded by ILA, the padded zeroes do not appear in the AXI Stream transfer. However, the additional bits were removed by writing the length of the message in the Length/Type field, instead of its type. In the final project, this field will signal an EtherCAT access. Thus, the padded zeroes will not be stripped. A solution would be manually stripping the zeroes after receiving the packet. For the moment, this issue is avoided by sending messages that are longer than 46 bytes.

Figure 3.6 shows the resource usage of the design. Half of the clocking management cells were used. One MMCM block was instantiated by the AXI Ethernet Subsystem in order to generate the clock signals for the RGMII and Medium Independent Interface (MDII) interfaces.

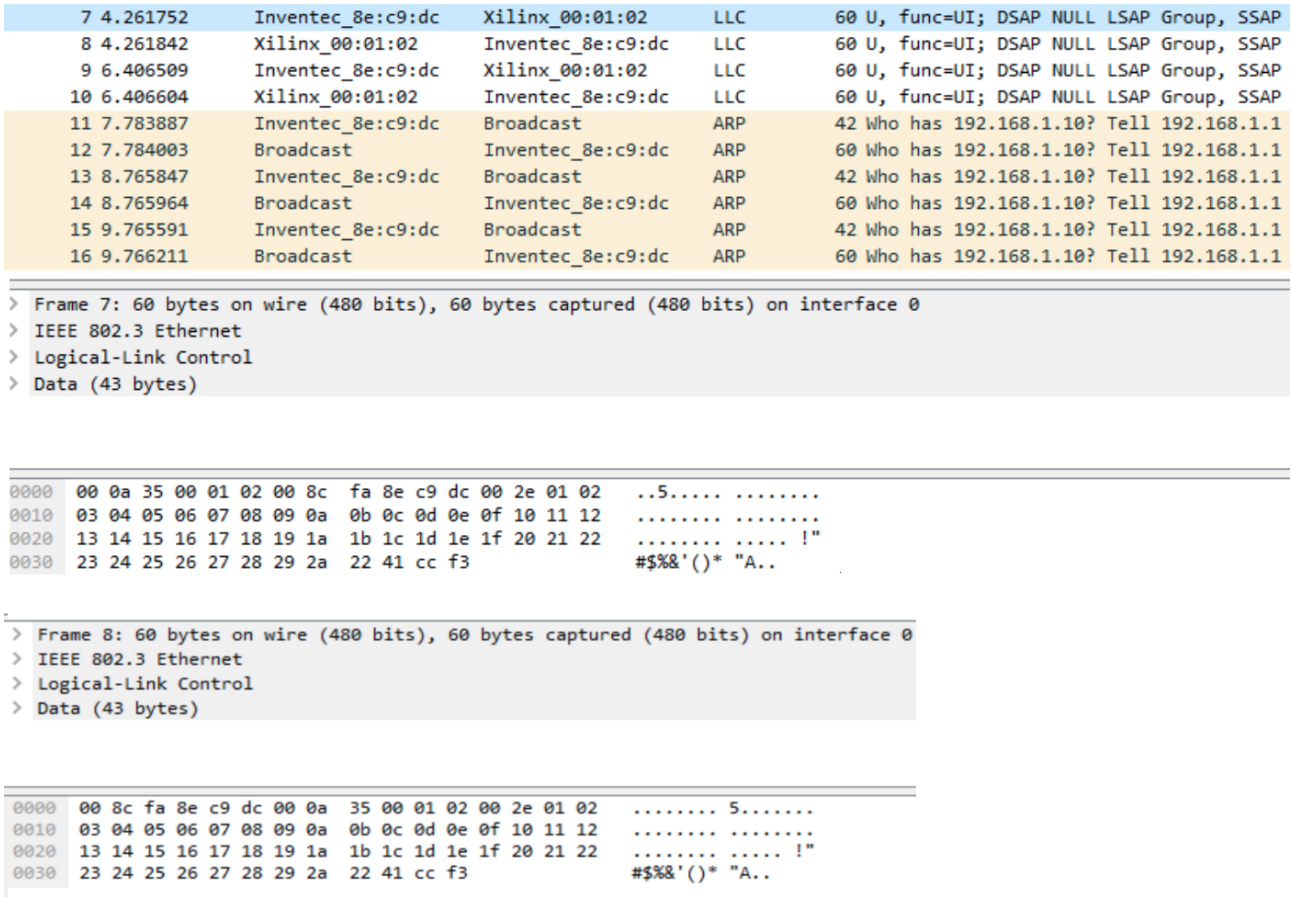


Figure 3.5: Functionality of the design implementing an Ethernet capable device.

The second was used to generate the clock signals used by the AXI buses and the reference clock required by the IP.

The circuit presented in this Chapter is able to set up the AXI Ethernet Submodule as well as the PHY, receive Ethernet packets and send them back with swaped source and destination address. The code version able to process EtherCAT packets is presented in the following Chapter.

Resource	Utilization	Available	Utilization %
LUT	3333	53200	6.27
LUTRAM	215	17400	1.24
FF	5118	106400	4.81
BRAM	4	140	2.86
IO	22	200	11.00
MMCM	3	4	75.00

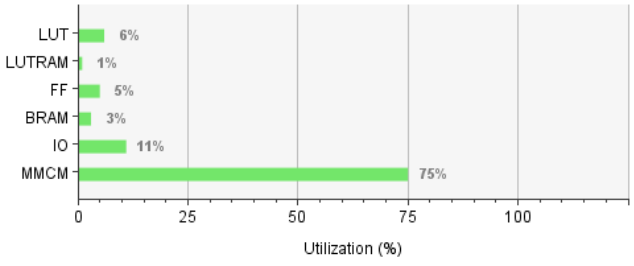


Figure 3.6: Resource utilization of the design driving one Ethernet port.

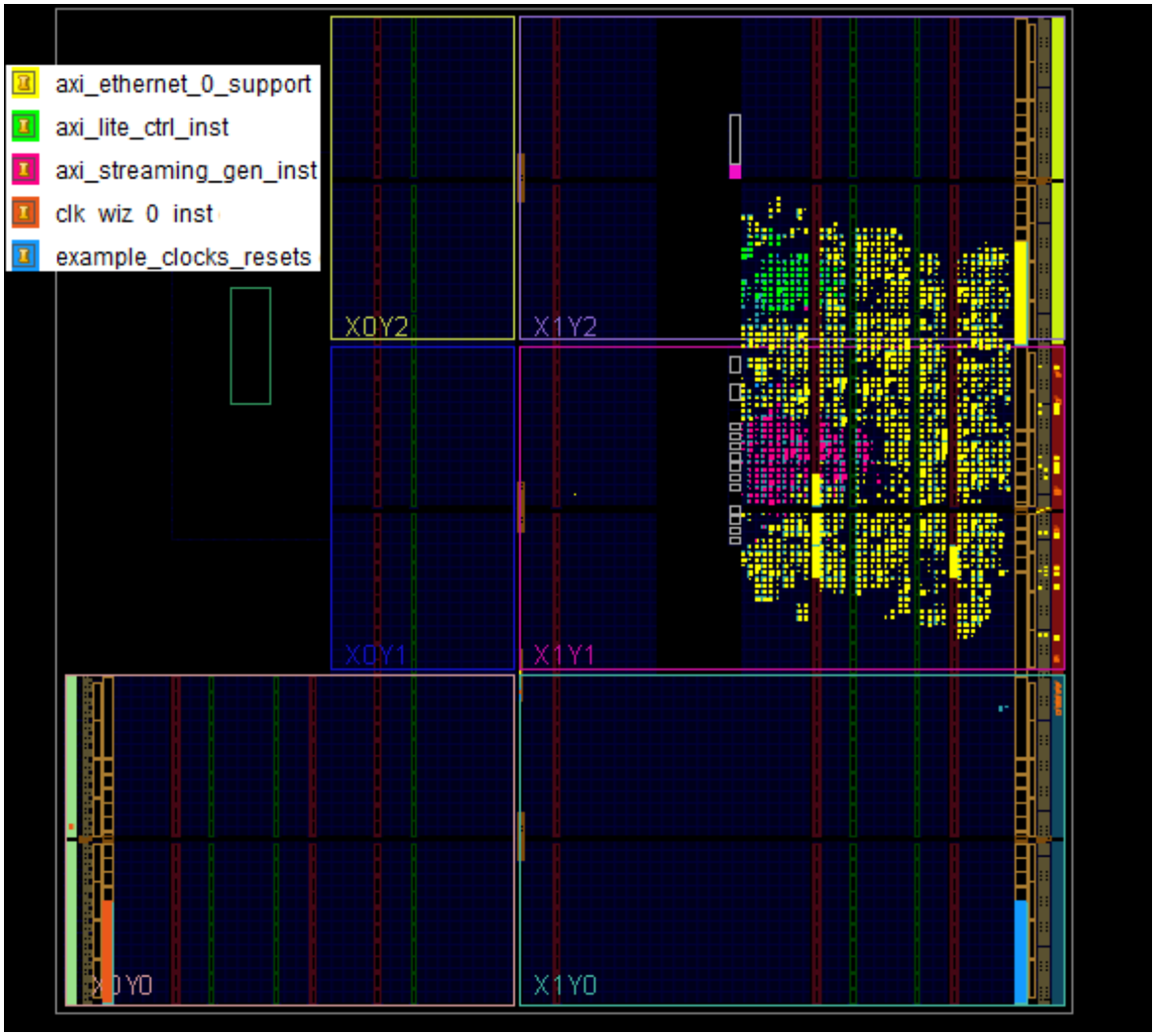


Figure 3.7: Placement of the design.

Chapter 4
Designing an EtherCAT Slave based on the AXI Ethernet Subsystem

The design presented in Chapter 3 was able to receive Ethernet packets, switch their destination and source address and loop them back to the computer. The project must be further developed in order to be able to recognize and process EtherCAT type frames.

First, the implementation of the design will be described in the next Section, which is followed by an analysis of the functionality and performance.

4.1 Implementation

The Verilog module responsible for writing the received data in a FIFO and sending it back is called: "axi_ethernet_0_slave_loopback". This module will be extended to process EtherCAT packages instead of passively looping them back.

The data coming from the computer goes through the PHY, is forwarded via RGMII interface to the AXI Ethernet Subsystem, which forwards it to the "axi_ethernet_0_slave_loopback" module through an AXI Stream interface. This interface sends 32 bit wide packages. This is an important aspect to be taken into consideration when processing incoming data. It poses some difficulties which are described in the following paragraphs.

When incoming packets were analyzed, an important aspect was observed. The user must take into consideration that the byte order of the AXI Stream is little endian. The same endianness issue is observed when analyzing packets in Wireshark. The word "0301_{16}" corresponds to the address 0103_{16}, which is shown in Figure 4.1. This must also be taken into consideration when interpreting the incoming packets.

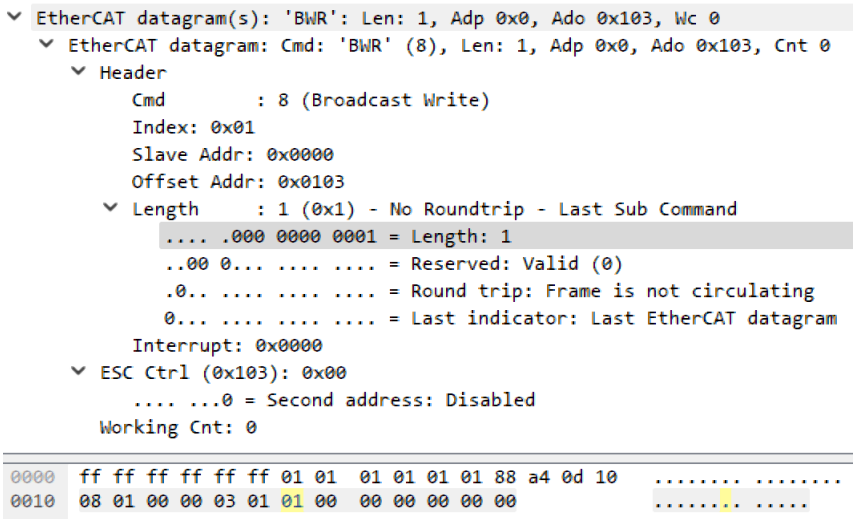


Figure 4.1: Interpretation of the EtherCAT Datagram in Wireshark.

In order to recognize different fields of the frame, some counters were used. One of them counts the incoming words of the Ethernet header, while the other is used for recording the

received datagrams.

The first step in processing data was recognizing an EtherCAT type package and ignoring all other frames. This was done by checking whether the type of the incoming package was $a488_{16}$.

Another step involved recording the length of the EtherCAT message. This was not necessarily required, as the end of an AXI Stream transaction is always signaled by the "axis_slvlb_d_tlast" flag.

In order to analyze the header of the datagrams, some aspects need to be taken into consideration. According to Figure 2.5, the Ethernet header and the EtherCAT header consist of 16 bytes, which corresponds to 4 AXI Stream transactions. Thus, the command type of the first datagram shall be the first byte in a 32 bit AXI transfer. This however is not necessarily valid for the next datagram. Depending on the length of the message in datagrams, the position of the datagram header in a 32 bit AXI transfer varies and it is also possible that a control field will be sent over two transactions. This misalignment problem must be taken into consideration when interpreting the incoming bytes. One common solution would be buffering the datagram and shifting it correctly. However, this shall not be implemented for an EtherCAT slave that must process data on the fly.

The offset of the datagrams is stored in the register "datagr_start". It might happen that one transaction contains information of two datagrams: n and n+1. For computing the byte position of datagram n, the value of the register "datagr_start" is used. In contrast, for the n+1 datagram, the combinational signal "new_datagr_start" is used. In the example processing the address, the second byte of the address is computed using the combinational signal when the start offset is zero or the value stored in the register "datagr_start" otherwise. Figure 4.2 shows how this misalignment problem is solved when computing the address of the datagram.

```
// check address
always @(posedge axis_clk) begin
    if(!axis_resetn)
        datagr_adr  <=11'd0;
    else begin
        if(wre && (((datagr_count==11'd4) && (datagr_start!=3)) || ((datagr_count==11'd8) && (datagr_start==3))))
            datagr_adr[7:0] <= s_axis_slvlb_d_tdata[(15+datagr_start*8)%32-:8];

        if(wre && (datagr_count==11'd4))
            datagr_adr[15:8] <= s_axis_slvlb_d_tdata[(7+datagr_start*8)%32-:8];

        if(wre && ((datagr_end) && (new_datagr_start==0)))
            datagr_adr[23:16] <= s_axis_slvlb_d_tdata[(31+new_datagr_start*8)%32-:8];
        else begin
            if(wre && ((datagr_count==11'd4) && (datagr_start!=0)))
                datagr_adr[23:16] <= s_axis_slvlb_d_tdata[(31+datagr_start*8)%32-:8];
        end

        if(wre && ((datagr_end) && (new_datagr_start<2)))
            datagr_adr[31:24] <= s_axis_slvlb_d_tdata[(23+new_datagr_start*8)%32-:8];
        else begin
            if(wre && ((datagr_count==11'd4) && (datagr_start>1)))
                datagr_adr[31:24] <= s_axis_slvlb_d_tdata[(23+datagr_start*8)%32-:8];
        end
    end
end
```

Figure 4.2: Solving misalignment problems when interpreting the datagram's address.

According to [48], an EtherCAT slave has a 64 KB address space, out of which 4 KB are used for configuration information and the data RAM begins at 0x1000. In this project, the entire address space was not implemented. A register file with 32 entries was described instead, which was sufficient for test purposes. Each entry is 8 bits wide, as the registers of a common EtherCAT Slave, according to [48]. When a message longer than 8 bits is received, the corresponding address is simply incremented. Depending on the transaction, the 32 bits received thorugh the AXI transfer could yield interrupt flag and data, only data bits or data

and the working counter field. Due to this, the position of the data bits varies and this must be taken into consideration when reading the incoming bits and writing them into the register. In order to do this correctly, another counter was used, which stores the number of already received bytes: "datagr_nr_B_wrttn".

An EtherCAT transaction can have different outcomes. This depends on the command type. [49] enumerates them. The design implemented in this project distinguishes between a read, a write or a read-write command. It takes into consideration the command types which have the value between 5 and 9. Depending on the type of command, the working counter is incremented differently. A valid read increments it with one. A successful write has the same effect. In case of a read/write command, a successful:

- Read increments the working counter with 1
- Write increments the working counter with 2
- Read and write increments the working counter with 3

4.2 Design Evaluation

The functionality of the design as well as its timing performance and resource usage are analyzed.

In order to test the EtherCAT slave on hardware, the Wireshark tool is used for analyzing network packets. The frame shown in Table 4.2 was built and sent from the computer to the FPGA.

Figure 4.3 shows the outgoing and incoming frames. Transaction number 15 corresponds to the sent data, while number 16 to the received one.

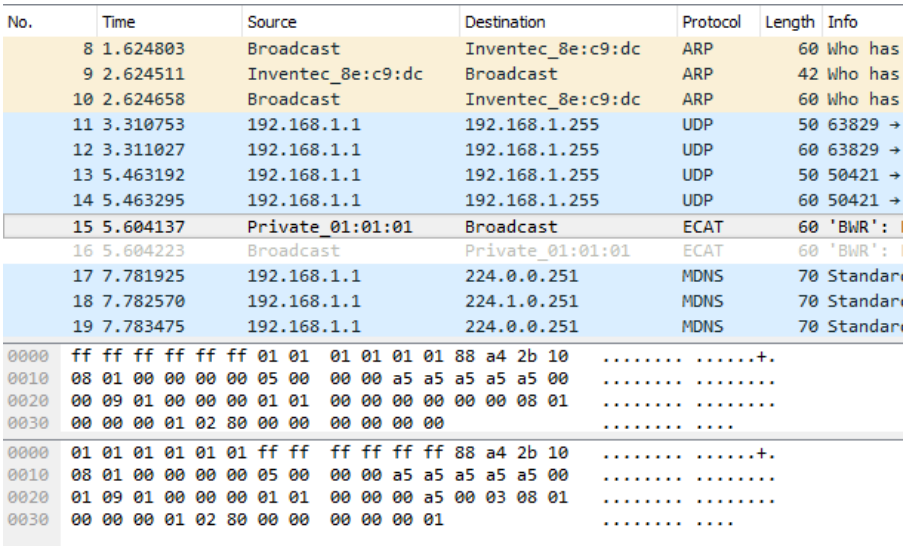


Figure 4.3: Analyzing the sent and received EtherCAT frame using Wireshark.

The answer from the slave was received after 86 µs. It must be noted however that the PC is not a real time system, thus this delay value might vary for different runs. Like in the previous design presented in Section 3, the module was capable of switching correctly the source and destination addresses. It recognized the EtherCAT frame and processed its headers and data. The slave was able to correctly update the value of the working counters, depending on the received command. For example, the last received byte was 1_16, which corresponds to a successful write command. Updating the working counter on the correct position of the

frame also implies processing correctly the length of the datagrams. This is valid also when the received datagram's headers have different offsets in the AXI packages, because the datagrams shown in Table 4.2 generate such offsets and the received frame is still standard compliant.

Additionally, the data received by the slave during the first datagram was written correctly in the memory. The proof is the value inserted in the second datagram, which is a read-write command at the same address.

Nr. of the last sent byte	Sent data	Dscription	Datagram number
0	0xffffffffffff	Destination Address	- Ethernet Header
6	0x010101010101	Source Address	- Ethernet Header
12	0x88a4	Signals an EtherCAT frame	- Ethernet Header
14	0x2b10	Length = 0x02b Reserved: Valid = 0x0 Type: EtherCAT Command = 0x1	- EtherCAT Header
16	0x08010000000005000000	Command = 8. Broadcast write Length = 0x5 Slave Address = 0x0000 Offset Adress = 0x0000	1
26	0xa5a5a5a5a5	Data sent to the EtherCAT Slaves	1
31	0x0000	Working Counter = 0x0000	1
33	0x09010000000101000000	Command = 9. Broadcast read-write Length = 0x1 Slave Address = 0x0000 Offset Adress = 0x0001	2
43	0x00	Data sent to the EtherCAT Slaves	2
44	0x0000	Working Counter = 0x0000	2
46	0x080100000001028000	Command = 8. Broadcast write Length = 0x2 Slave Address = 0x0000 Offset Adress = 0x0001 Last Datagram	3
56	0x00000	Data sent to the EtherCAT Slaves	3
58	0x0000	Working Counter = 0x0000	3

Table 4.1: Frame sent from the computer to the designed EtherCAT slave.

Further on, the resource usage of the design is analyzed. The results are shown in Figures 4.4. A visual result of the routing and placement process is shown in Figure 4.5. The module that was simply looping back the data in Chapter 3 is now able to process EtherCAT frames. The added logic increased the resources used by this module with 1531 LUTs. However, as in the previous design, most of the resources are required by the Ethernet Subsystem.

Resource	Utilization	Available	Utilization %
LUT	4864	53200	9.14
LUTRAM	215	17400	1.24
FF	4862	106400	4.57
BRAM	4	140	2.86
IO	23	200	11.50
MMCM	2	4	50.00

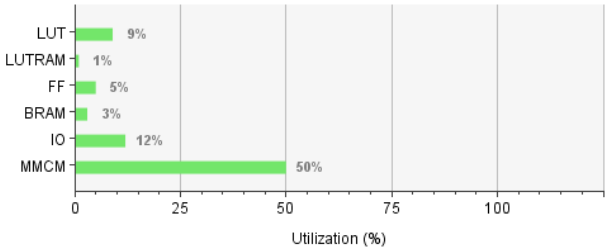


Figure 4.4: Summary of the used resources.

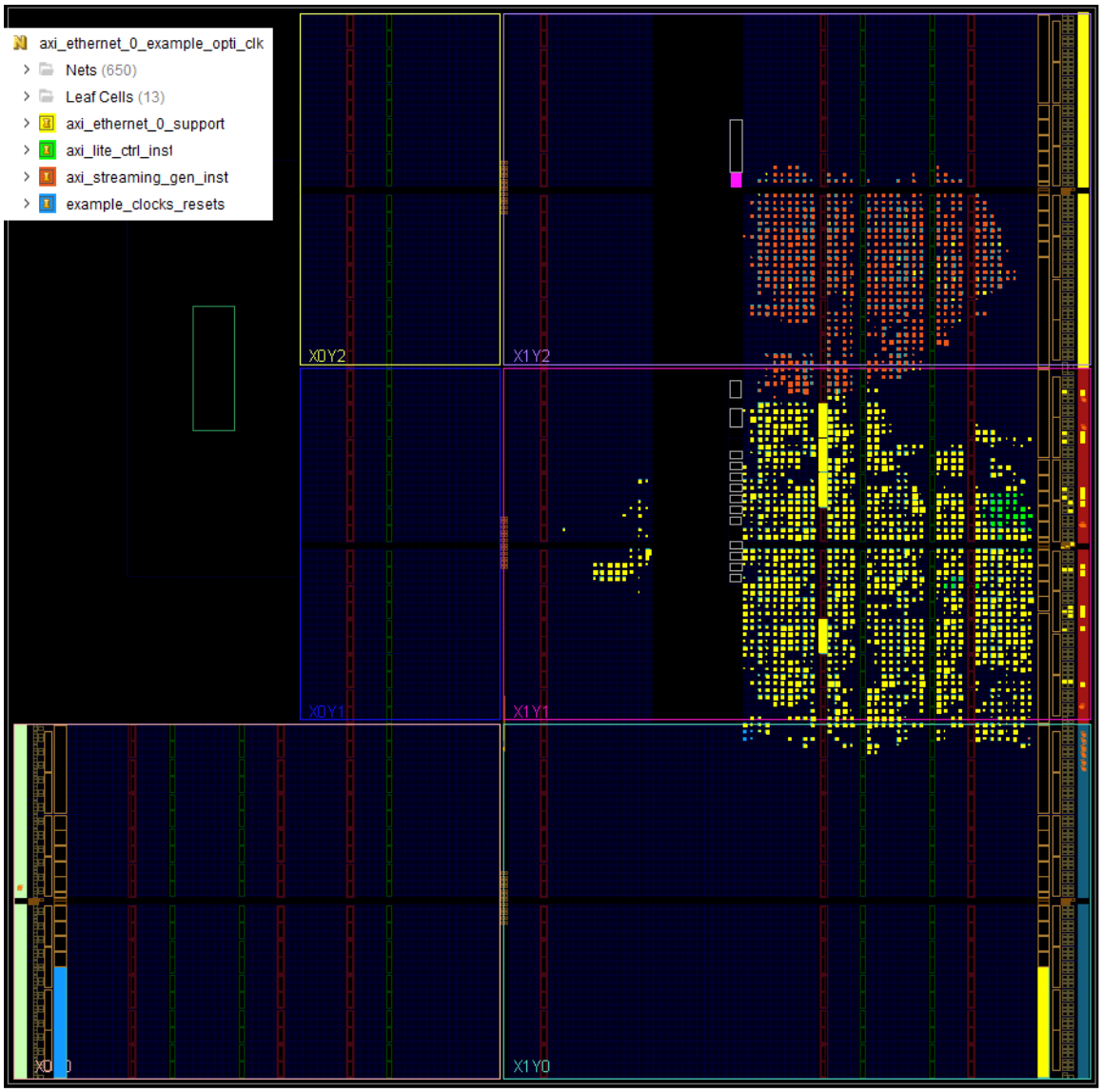


Figure 4.5: Design placement of the EtherCAT slave.

Chapter 5
Designing a Linear Topology

5.1 Design Architecture

As stated in Chapter 2, the EtherCAT technology is used in distributed systems. Different network topologies can be used, like line, tree, star or daisy-chain configurations. As the designed EtherCAT slave shall be able to be attached to such a network, this Chapter focuses on analyzing this capability of the already developed design.

For test purposes, a linear topology was chosen. The design used as EtherCAT slave was the one developed in Chapter 4. However, its capabilities must be extended in order to be able to attach it to a network. These changes and the system architecture are described in more detail in Section **??**. The design is than evaluated in Section **??**.

5.2 Implementation

The tested network configuration is a linear one, having two EtherCAT slaves. It fulfills the architecture shown in Figure **??**. The data is sent from the computer to the first slave, which processes the information and forwards the message to the next slave. After processing it, the second slave must send back the data to the first one. Further on, the first slave forwards directly the data to the master.

The current design version can not be inserted directly into the topology. Some extensions must be implemented.

First of all, the current design is not able to detect whether it is the last one in the chain. This is necessary so that it would be able to loop back the EtherCAT frame in case no other slaves follow him in the chain. This feature can be implemented by using the registers of the PHY. As stated in Chapter 2, the PHY has its registers organized in pages. The status information provided by register number 17, of page 0, called "Copper Specific Status Register 1", can be useful in solving this task. Bit number 10 gives information in real time regarding the status of the link. A high state of this bit means the communication link is up. If the current slave is the last one in the chain, one of its PHYs will not be connected to any other decive. Thus, its link will remain down and this can be read from the status register number 17.

In order to test whether the current slave is the last one in the chain or not, two new states are added to the control module, "RDLINK" and "CHECKLINK". The first one requests a MDIO access to register 17, page number 0. The second one reads the received value from register $50c_16$ of the Ethernet Subsystem.

If the link is down, the data processed by the streaming generator is looped back to the first AXI Ethernet core. If the link is up, it forwards the information to the second Ethernet subsystem.

The original slave had the task to switch the source and destination addresses of the Ethernet frame before sending it back to the master. However, if this would be conducted by each slave in the topology, depending on the number of connected devices, the data received by the master

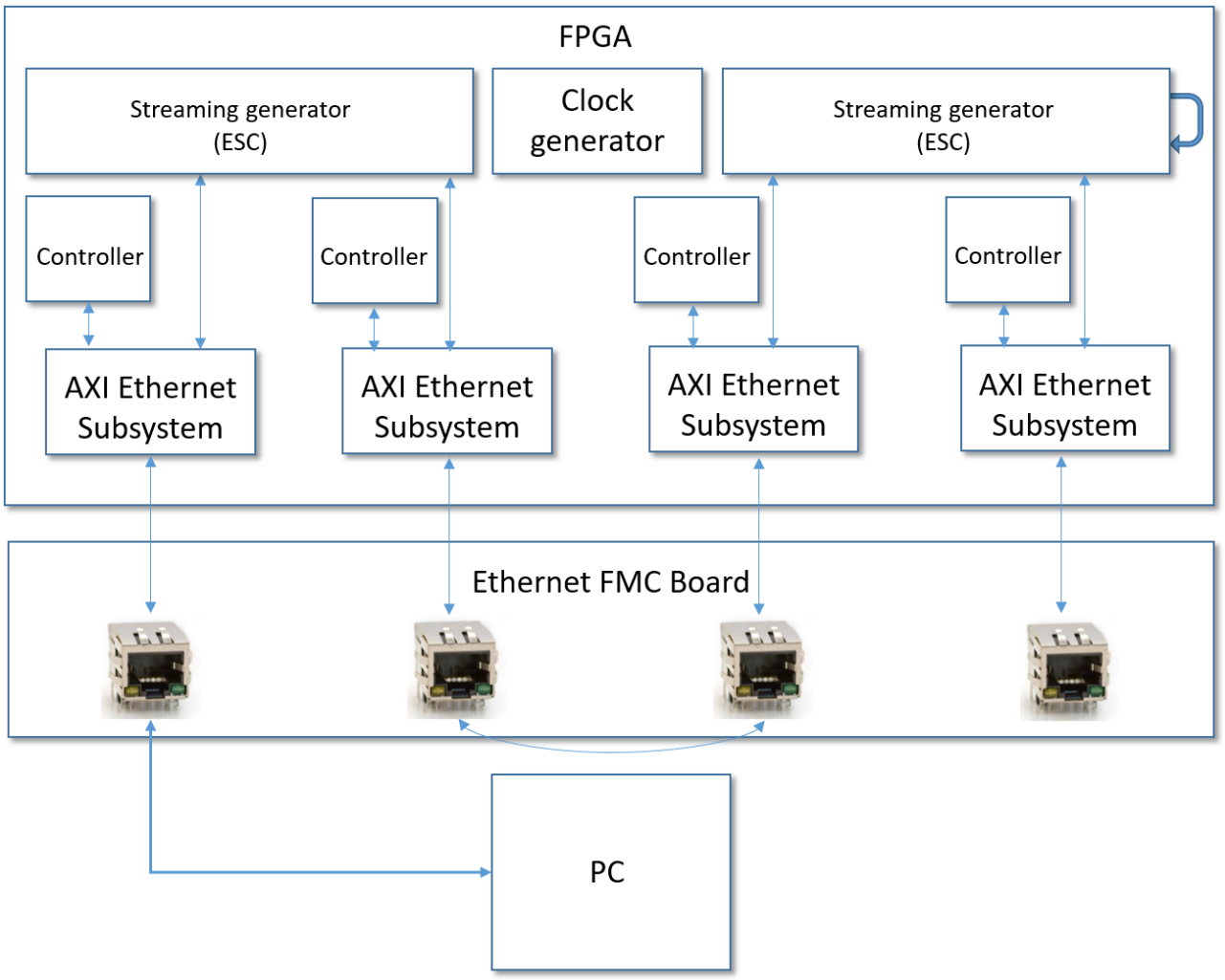


Figure 5.1: Linear topology containing two EtherCAT slaves.

might not have the correct addresses. This must be updated so that only one slave would swap the addresses. As it can already be detected which slave is the last one in the chain, he could be the one conducting this task. Thus, the streaming generator module was updated so that it receives information from the control module regarding its position in the chain and depending on that it interchanges the addresses or not.

The master can conduct a topology scan by sending a broadcast package and reading the received value of the working counter.

After implementing the design updates, the design needs to be synthesized so that later on it could be tested on actual hardware. The real distributed system would have one FPGA for each field device. One board would contain:

- two PHY ports
- two AXI Ethernet Subsystems
- one clock management system
- two control modules, one for each Ethernet subsystem

However, for test purposes these modules will be instantiated on the same FPGA, as the Ethernet FMC supports the design principle by having four ports. In order to save resources

and speed up the synthesis and implementation processes, only one clock management unit is used for both EtherCAT slaves.

An important aspect observed when instantiating more AXI Ethernet Subsystems was the fact that out of the four AXI Ethernet instances, only one is allowed to include shared logic in the core option. This behavior is described in subchapter 3.2.

5.3 Design Evaluation

Before testing the implemented design on hardware, the right cable connections must be done. Data from the second port, which corresponds to the first slave, must be forwarded to the third one, which corresponds to the second slave. The correct type of cable must be chosen in order to transmit the correct data. Some of the PHYs support only crossover cables, while others are able to swap internally the signals in case straight through cable is detected.

Accroding to [29], the module is capable of swapping the receive and transmit signals if this is needed, so that no cross over cable is required. According to the control register "Copper Specific Control Register 1" bits five and six are initialized with value 3_16,so the automatic crossover is enabled after hardware reset.

The first conducted test was checking whether the EtherCAT slave is able to detect if there is any device connected to it. Reading the "Copper Specific Status Register 1" once, after reset does not return the correct result, as the link is down in the beginning. Thus, a loop must be implemented to request repeatedly the link status in case the corresponding bit is low. If the device is not the last one in the chain, bit ten of the above mentioned register becomes high after a certain time. The corresponding value can be seen in the data captured using ILA (Figure **??**). Bit ten of the "s_axi_rdata" signal is one.

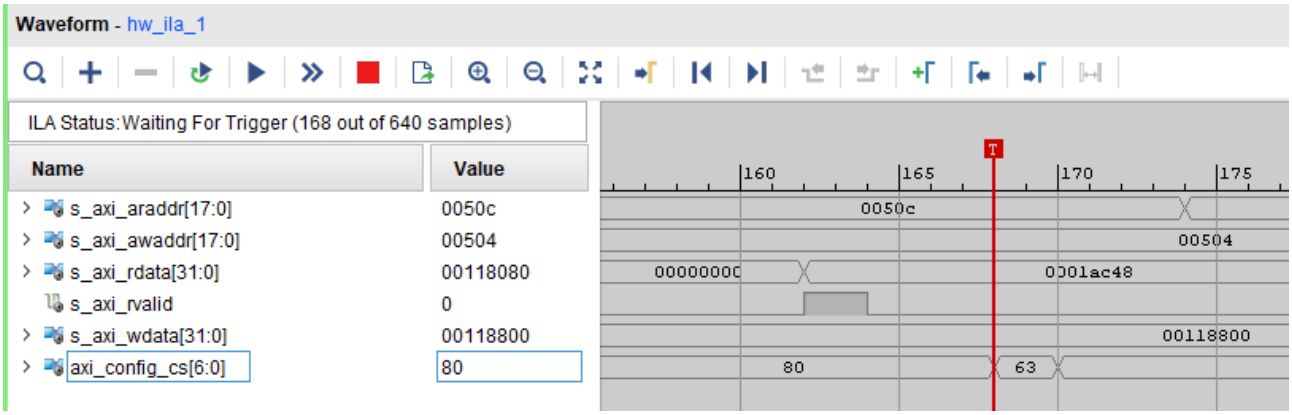
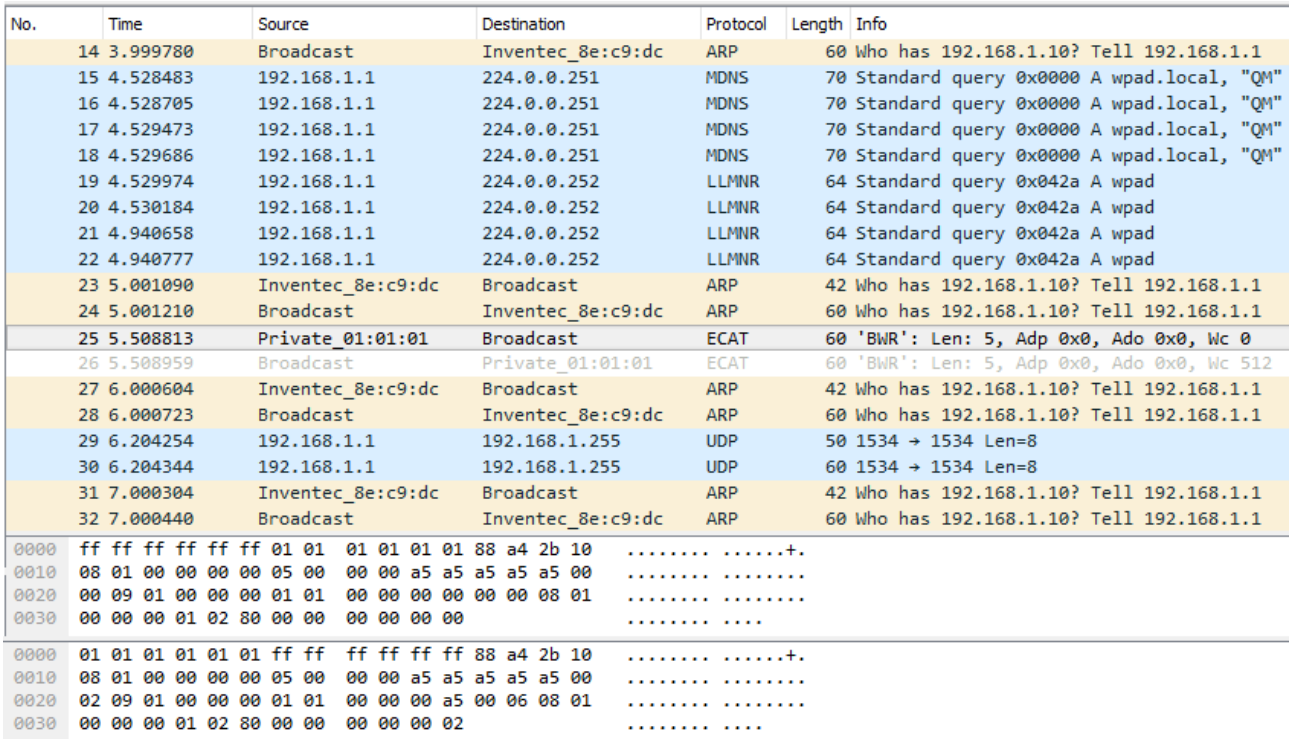


Figure 5.2: Reading the link status of an Ethernet port connected to another device.

The data is looped back correctly by the last slave, as the master receives the sent frame. This can be seen in Figure 5.3. If the logic implementing the swapping of the addresses was defective, both slaves would have interchanged the source and destination addresses so that the master would have received the same Ethernet header. This however does not happen according to Figure 5.3, so the two slaves can detect correctly whether they are the last ones in the network chain or not.

Another important aspect is observing the values of the EtherCAT datagrams. Although in real-case designs two slaves would have different addresses, for test purposes, the slaves instantiated in this project had the default address 0. Therefore, when sending a packet to this address, both slaves should react and update the working counter. This behaviour is confirmed when using Wireshark. If we compare the last byte received in Figure 4.3 with the one in Figure 5.3, we can see that the working counter was incremented twice in the second case.



No.	Time	Source	Destination	Protocol	Length	Info
14	3.999780	Broadcast	Inventec_8e:c9:dc	ARP	60	Who has 192.168.1.10? Tell 192.168.1.1
15	4.528483	192.168.1.1	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM"
16	4.528705	192.168.1.1	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM"
17	4.529473	192.168.1.1	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM"
18	4.529686	192.168.1.1	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM"
19	4.529974	192.168.1.1	224.0.0.252	LLMNR	64	Standard query 0x042a A wpad
20	4.530184	192.168.1.1	224.0.0.252	LLMNR	64	Standard query 0x042a A wpad
21	4.940658	192.168.1.1	224.0.0.252	LLMNR	64	Standard query 0x042a A wpad
22	4.940777	192.168.1.1	224.0.0.252	LLMNR	64	Standard query 0x042a A wpad
23	5.001090	Inventec_8e:c9:dc	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.1
24	5.001210	Broadcast	Inventec_8e:c9:dc	ARP	60	Who has 192.168.1.10? Tell 192.168.1.1
25	5.508813	Private_01:01:01	Broadcast	ECAT	60	'BWR': Len: 5, Adp 0x0, Ado 0x0, Wc 0
26	5.508959	Broadcast	Private_01:01:01	ECAT	60	'BWR': Len: 5, Adp 0x0, Ado 0x0, Wc 512
27	6.000604	Inventec_8e:c9:dc	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.1
28	6.000723	Broadcast	Inventec_8e:c9:dc	ARP	60	Who has 192.168.1.10? Tell 192.168.1.1
29	6.204254	192.168.1.1	192.168.1.255	UDP	50	1534 → 1534 Len=8
30	6.204344	192.168.1.1	192.168.1.255	UDP	60	1534 → 1534 Len=8
31	7.000304	Inventec_8e:c9:dc	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.1
32	7.000440	Broadcast	Inventec_8e:c9:dc	ARP	60	Who has 192.168.1.10? Tell 192.168.1.1

```
0000  ff ff ff ff ff ff 01 01  01 01 01 01 88 a4 2b 10   ........ ......+.
0010  08 01 00 00 00 00 05 00  00 00 a5 a5 a5 a5 a5 00   ........ ........
0020  00 09 01 00 00 00 01 01  00 00 00 00 00 00 08 01   ........ ........
0030  00 00 00 01 02 80 00 00  00 00 00 00               ........ ....
```

```
0000  01 01 01 01 01 01 ff ff  ff ff ff ff 88 a4 2b 10   ........ ......+.
0010  08 01 00 00 00 00 05 00  00 00 a5 a5 a5 a5 a5 00   ........ ........
0020  02 09 01 00 00 00 01 01  00 00 00 a5 00 06 08 01   ........ ........
0030  00 00 00 01 02 80 00 00  00 00 00 02               ........ ....
```

Figure 5.3: Sent and received EtherCAT frame in case of a linear topology containing two slaves.

Moreover, the network delay in case of having only one slave can be compared to the delay observed now. Table 5.3 illustrates a comparison between the design implemented in Chapter 4 and the linear topology implemented now. As stated also in the previous chapter, because the PC is not a real time device, different runs might result in different delays.

Type of design	Design containing one EtherCAT slave		Linear topology containing two EtherCAT slaves	
Type of data	Transmitted frame	Received frame	Transmitted frame	Received frame
Timestamp (s)	5.604137	5.604223	5.508813	5.508959
Delay (s)	0.000086		0.000146	

Table 5.1: Timing characteristics of the sent and received EtherCAT frames in case of one or two slaves.

Figures 5.4 and 5.5 illustrate the resources used for implementing the linear topology. However we must take into consideration that one module was used for generating the clocks for both slaves, which is not valid in real-case applications, where the slaves are implemented on different boards.

Resource	Utilization	Available	Utilization %
LUT	15648	53200	29.41
LUTRAM	1247	17400	7.17
FF	18926	106400	17.79
BRAM	16	140	11.43
IO	68	200	34.00
MMCM	2	4	50.00

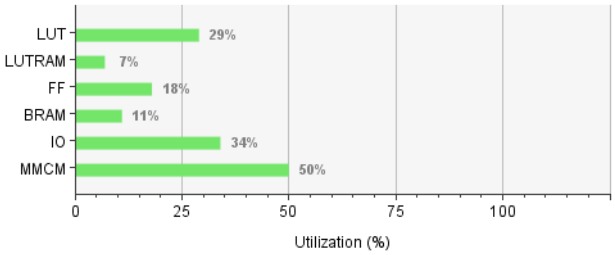


Figure 5.4: Summary of the resources used for implementing two EtherCAT slaves connected in linear topology.

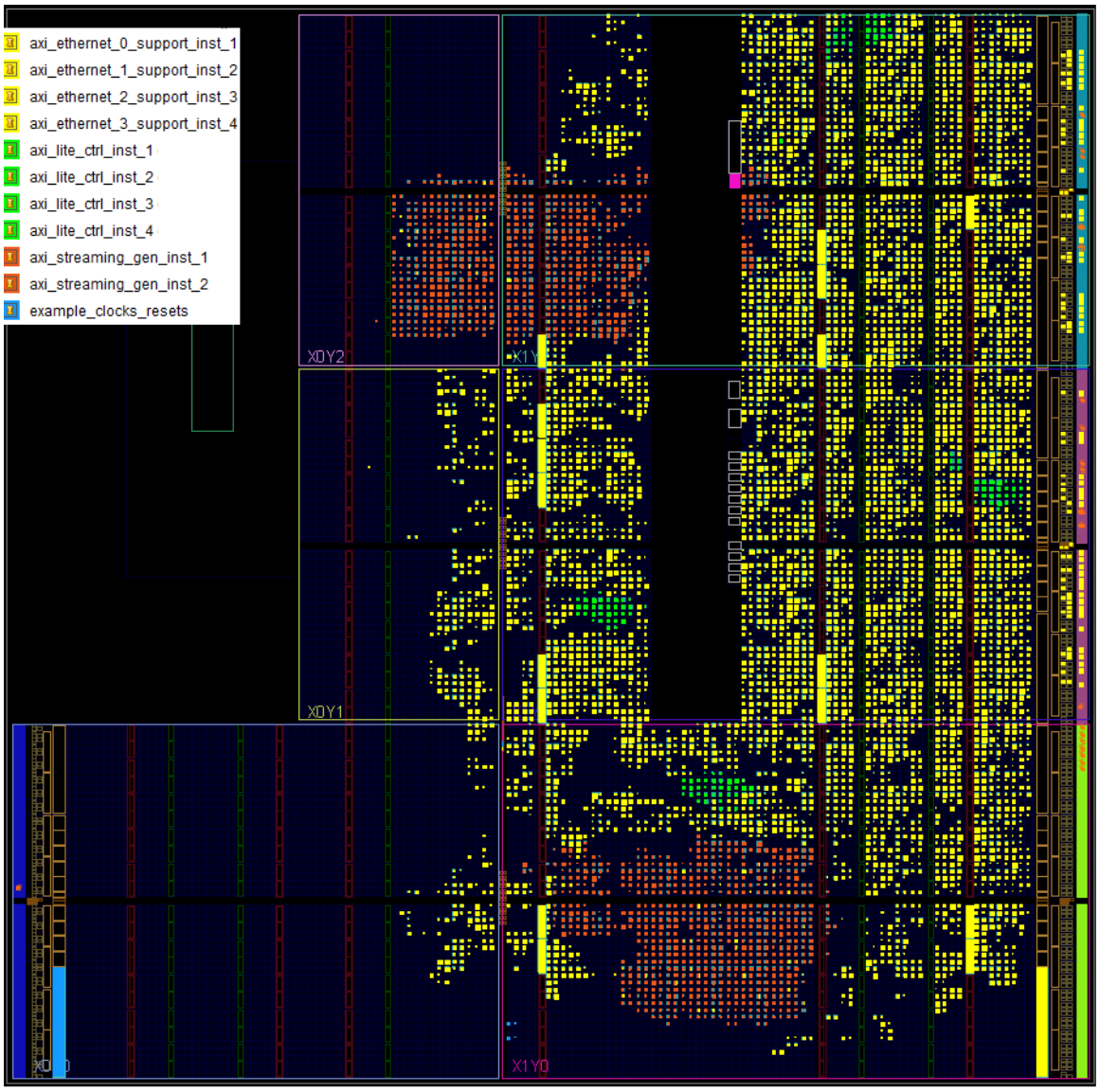


Figure 5.5: Design placement of two EtherCAT slaves connected in linear topology.

Chapter 6
Removing the AXI Ethernet Subsystem

In Chapter 4, an EtherCAT slave was designed using the AXI Ethernet Subsystem. However, using this core poses some decisive drawbacks. Firstly, using this IP brings some limitations regarding design speed and reduces performance when it comes to resource utilization. Secondly, an essential limitation when using this core is not having access to the HDL files.

6.1 Motivation for replacing the Xilinx IP

Despite being able to set some parameters of the IP before generating the netlist, the user can not modify particular signals of the source code. Thus, he can not adjust the design to his own needs.

Additionally, the design speed is limited by the AXI interfaces, which can be observed in Figure 6.1. The IP first buffers the data received through the RGMII interface, which is pointed by the first cursor. After receiving the last nibble, it initiates an AXI transfer, signaled by the second cursor. The control logic processes the data and sends it back to the Ethernet Subsystem. The IP buffers again the received data before sending it to the PHY. The fourth cursor signals the transmission through the RGMII interface. This highlights the delay of the loopbacked data.

The AXI stream data bus is 32 bit wide, thus the received frame is sent within several bus accesses. According to the datasheet [31], the AXI Stream interfaces "typically operate with a clock between 100 MHz and 125 MHz". The RGMII works however at 125 MHz, sending one data nibble on each clock edge. Thus, the data needs to be buffered before sending it through the AXI interface. In case of real time devices, the optimal solution would be receving data, processing it and forwarding it without any buffering delay. Thus, removing the subsystem and implementing a design to improve timing performance would be of interest.

Another important issue when instantiating the Ethernet Subsystem is the size of the circuit. The area used by the core is around 45% of the total design area required by the design in Chapter 4.2. As the core can change at run-time some parameters of the communication, depending on the control registers, its logic is more complex than required by this project. Complexity is reflected also in the required hardware and the unused functionalities shall be removed in order to save resources. A simple example is supporting Jumbo Frames, which have a length bigger than the one specified in IEEE Std 802.3-2002. This feature will not be used in case of the EtherCAT slave developed in this project, so it can be removed.

Last but not least, the AXI Ethernet Subsystem is not an open source project and the required license raises the design costs.

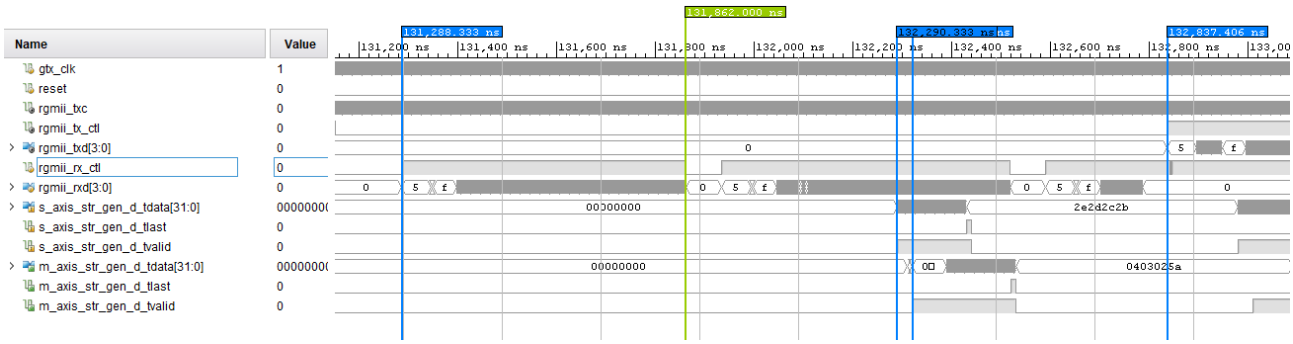


Figure 6.1: Timing performance of the AXI Ethernet Subsystem.

6.2 Implementation of the Cyclic Redundancy Check

When transmitting data to another Ethernet capable device, a 32 bit CRC value must be padded to the message. This will be the FCS field of the Ethernet frame, according to the protocol requirements described in Chapter 2.1. According to [50], the CRC "is a non-secure hash function designed to detect accidental changes to raw computer data". Because the frame would be dropped at OSI layer two if the received checksum would not correspond to the actual computed one, implementing a CRC generator is essential. In previous designs, this task was conducted by the AXI Ethernet Subsystem.

The design has the following input signals:

- The generator polynomial
- The input data
- The last result

As output signal, the circuit returns the computed checksum. The CRC computation is based on the division operation. The input data is the divident. Depending on the degree of the polynomial, zeros must be padded to it. The input data is divided by the generator polynomial and the rest of the computation is the required CRC.

When using this module for Ethernet capable devices, the generator polynomial can also be defined as a parameter, because it has a constant value. It shall have a high bit on each position where the polynomial has a coefficient different from zero. According to [51], the generator polynomial used for Ethernet capable devices should comply to the following formula:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0 \tag{6.1}$$

However, this signal was left as input so that the module could be used in future projects also for other purposes.

In order to design the CRC generator in hardware, a shift register was used. The most shifted bit is xored with the input data, which is eight bit wide. The obtained value is "AND"combined with the generator polynomial and the result is xored with the shifted previous result. This behavior is illustrated in Figure 6.2.

The design works asynchronously, computing the data in one clock cycle. So the above decribed circuit has an actual parallel structure. It requires more resources as a synchronous shift register, but it is able to process on the fly each byte received through the RGMII interface.

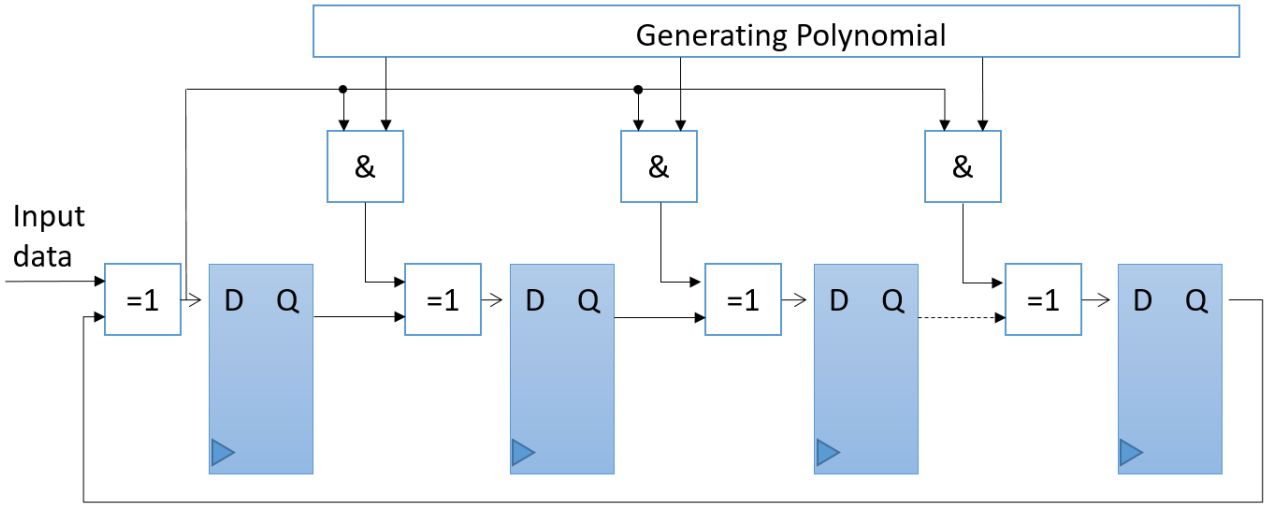


Figure 6.2: Design of the CRC module.

6.3 Implementation of the MDIO communication

The MII managment interface is used in order to write control data in the registers of the PHY. Its functionality was described in Section **??**.The "axi_ethernet_0_axi_lite_ctrl" control module, that has the task of bringing up correctly the PHY and the AXI Ethernet Subsystem has three implemented finite state machines. One of them controls AXI Lite accesses in case data must be written through the MDIO wire in the PHYs registers. Thus, all the control data for the PHY first goes through the Ethernet IP and requires two AXI bus accesses, as described in Section 3.2. This steps can be removed by implementing a design capable of conducting the MDIO accesses.

An additional important advantage in creating a MDIO capable module is removing some states of the finite state machine that were writing data into the IPs control registers in order to set up correctly the MII interface. The parameters of this communication will be hard-wired in the design.

The new module receives from the "axi_ethernet_0_axi_lite_ctrl" the following signals:

- Enable signal, telling the design when the data line has valid entries
- Address of the PHY
- Address of the accessed register
- Data to be written into the PHYs registers
- Read/Write command

After a transfer was issued, a counter stores the number of the already sent bits and generates on the MDIO line the pattern described in Section **??**. The module releases the line (through a high impedance signal), when it expects data from the PHY or does not conduct any access. In the top module, an Input/Output Buffer (IOBUF) cell makes possible that both this module as well as the PHY can access the MDIO wire.

The module receives a 2.5 MHz clock signal.

The output data consists of the information sent to the control module, as well as the data sent to the PHY.

6.4 Implementation of the RGMII communication

The PHY communicates with the EtherCAT data link layer thorugh the MDIO and RGMII interfaces. The first one was already implemented in the previous Section. This Section introduces an implementation of the RGMII interface.

The designed module to conduct this task is called “rgmii”. It must conduct receive as well as transmit operations. It complies to the protocol pattern introduced in Chapter **??**.

In order to detect the start of a frame sent by the PHY, the receiver logic stores the last received nibble. When the sequence “$5d_{16}$” is detected, a start of a new frame is signaled. The data received afterwards is stored in a buffer of 45 bits. The reason is the FCS field. As the “Streaming generator” module, processing the incoming data, does not need the FCSfield, removing it is desired. The “buffer_remove_fcs ” FIFO stores each received nibble with an attached valid bit. This bit is set high as the data is received. If the end of the frame was reached, by detecting a negative edge of the “rgmii_rx_ctl” signal, the valid bits of the last eight stored nibbels are reset. Thus, they are not forwarded to the processing logic.

The transmission logic forwards the data to the PHY when a write request is received from the “Streaming generator” design. Apart from the preamble data, each sent nibble is also forwarded to the module computing the CRC. After sending the data blocks, the CRC result is also attached to the signal sequence.

Because data must be sent at double data rate, both receiver and sender logic work with a 250 MHz clock. The transmission clock forwarded to the PHY is defined at 125 MHz. The “rgmii_rxc” clock signal is received from the PHY and shall have the same frequency.

6.5 Updating the architecture design

After replacing the AXI Ethernet Subsystem with own implemented modules, the architecture of the design also changes. Figure 6.3 shows the updated structure of the project.

The Ethernet Subsystem module was replaced by two designs, one implementing the MDIO communication, while the other conducting RGMII transfers.

The “Controller”module has the task of sending control information to the PHY via the “Management Data Input/Output”. By writing correctly the registers of the PHY, it must bring it up correctly.

The “Streaming generator”module processes data received from the “Reduced Gigabit Media Independent Interface” module that communicates with the PHY through the RGMII interface. Both modules work at the same frequencies.

In order to generate the correctCRC value of the frame, so that the receiver would not drop the message, the “Reduced Gigabit Media Independent Interface” also instantiates the “CRC” module.

As the IP provided by Xilinx was removed, the AXI buses were also no longer needed. Thus, the new design is more simplified. Additionally, because AXI buses worked at different frequencies than the MDIO and RGMII interfaces, removing them also meant having less clock domain crossings.

6.6 Design Evaluation

First, the functionality of the design will be tested. In order to ease debugging, the “Streaming generator”module is in this case a simple loop back, without implementing EtherCAT capabilities.

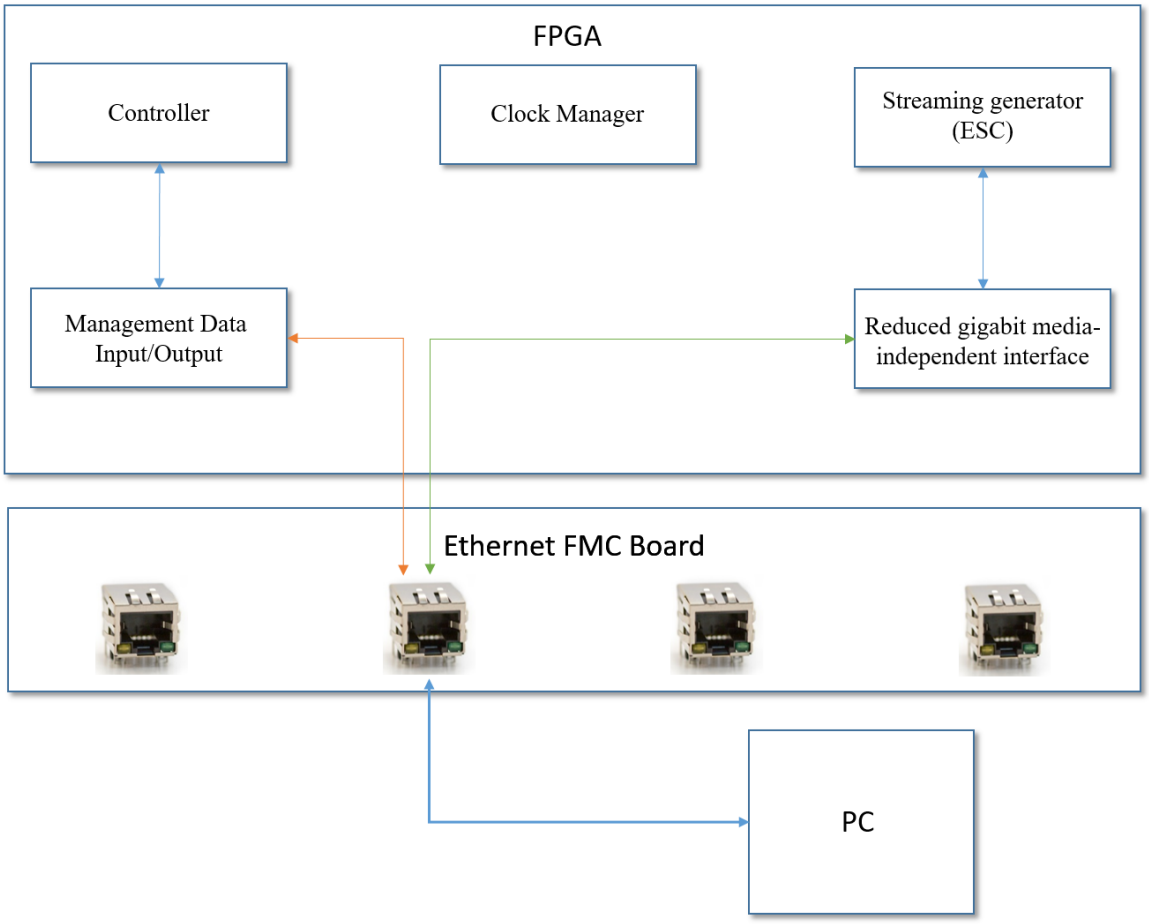


Figure 6.3: Design of the CRC module.

The design works in case of certain frames, while in other cases it introduces some bit errors. Figure 6.4 shows a case when the design passes the test, while Figure 6.5 shows an example when it fails. The data received in the second case was not completly wrong. However, a few bits had the wrong value that flag some timing issues in the design. As the RGMII interface requires strict conditions regarding signal delay, it might be possible that the defined timing constraints do not match entirly the specification.

In order to debug this problem, ILA was used. THe results are shown in Figure 6.6. The frame presented in Chapter 4.2 was used again as test data. At first sight, the signals captured in ILA on the RGMII interface were according to the requirements. We can recognize some bytes of the frame not only on the receiving bus, but also on the transmission line. We can also recognize in Figure 6.6 the preamble before sending the data. Additionally, the addresses are swaped correctly. Taking this into consideration, an assumption would be an error in defining the timing of the transmission wires. This must be researched in further work.

As expected, the design requires less logic. Figure 6.8 shows in yellow the logic replacing the AXI Ethernet Subsystem. As it does not provide the flexibility and complexity of the Xilinx core, it uses much less logic. Additionally, it does not buffer the entire data received through the RGMII interface, which also saves a great amount of logic.

The subsystem required one MMCME2 cell and the rest of the logic used a second clock manager module. Removing the core also means removing one of the MMCME2 cells. As the FPGA does only have 4 such elements, this is might be useful if additional clocking resources are needed in later complex projects.

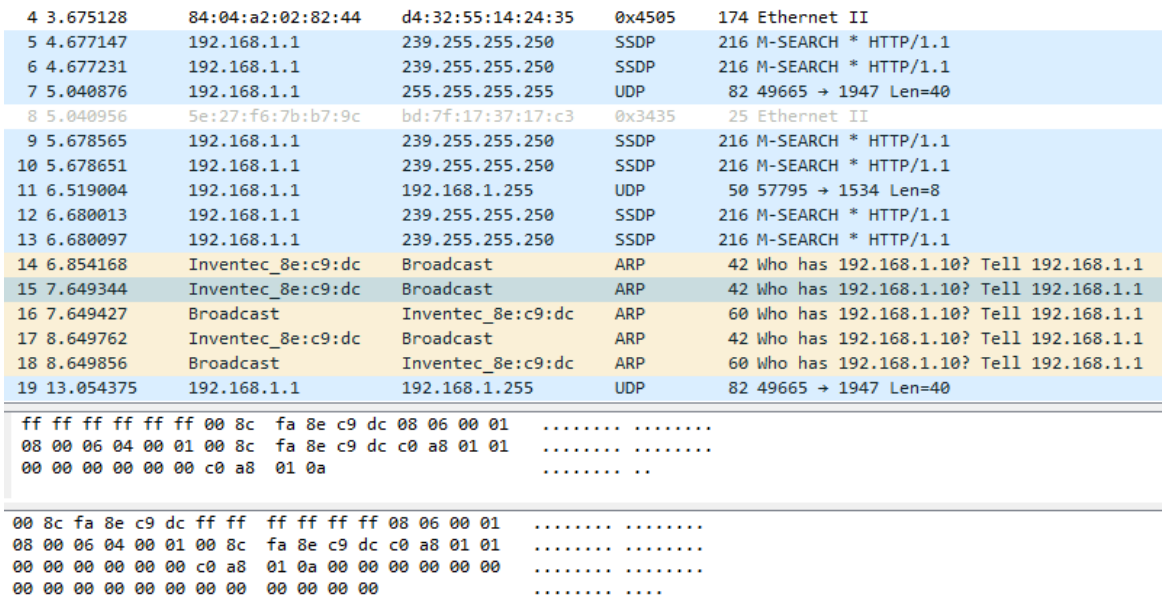


Figure 6.4: Example case when the design fulfills the requirements. Top frame is the sent data, bottom frame is the received data.

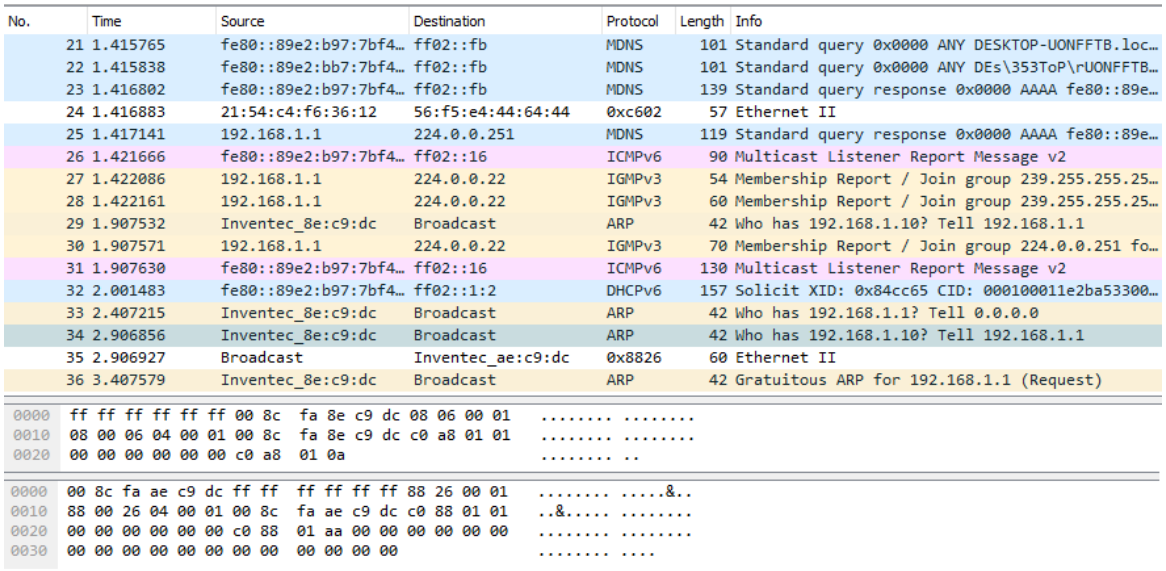


Figure 6.5: Example case when the transmission includes a few wrong bits. Top frame is the sent data, bottom frame is the received data.

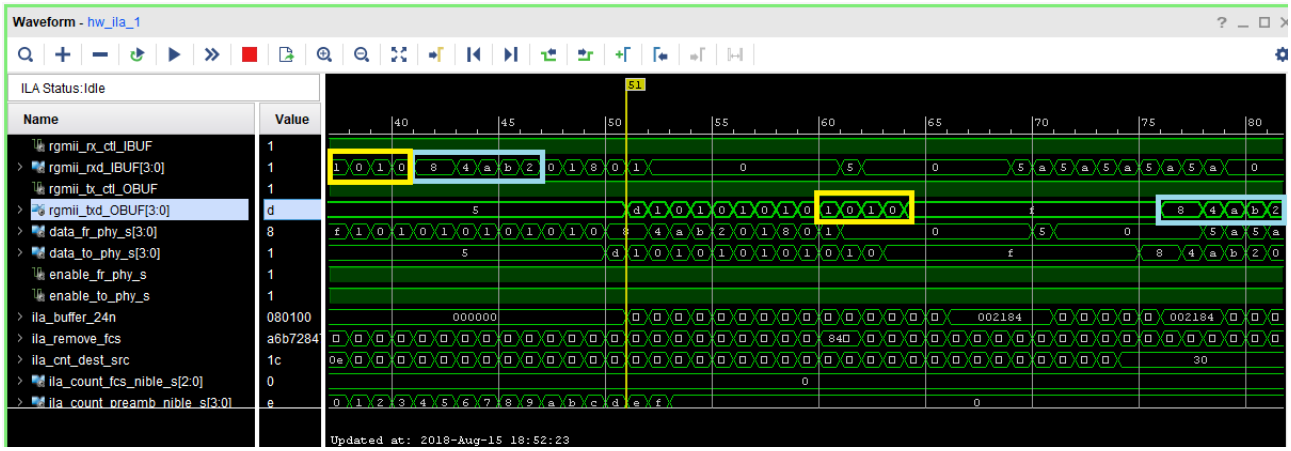


Figure 6.6: Debug information captured by ILA.

Resource	Utilization	Available	Utilization %
LUT	393	53200	0.74
LUTRAM	10	17400	0.06
FF	356	106400	0.33
IO	22	200	11.00
MMCM	1	4	25.00

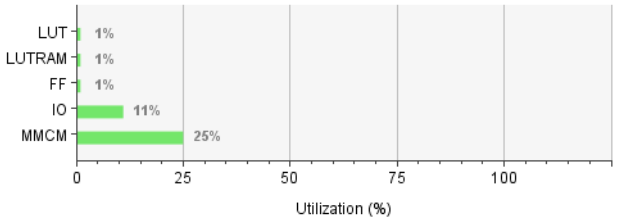


Figure 6.7: Summary of the required resources.

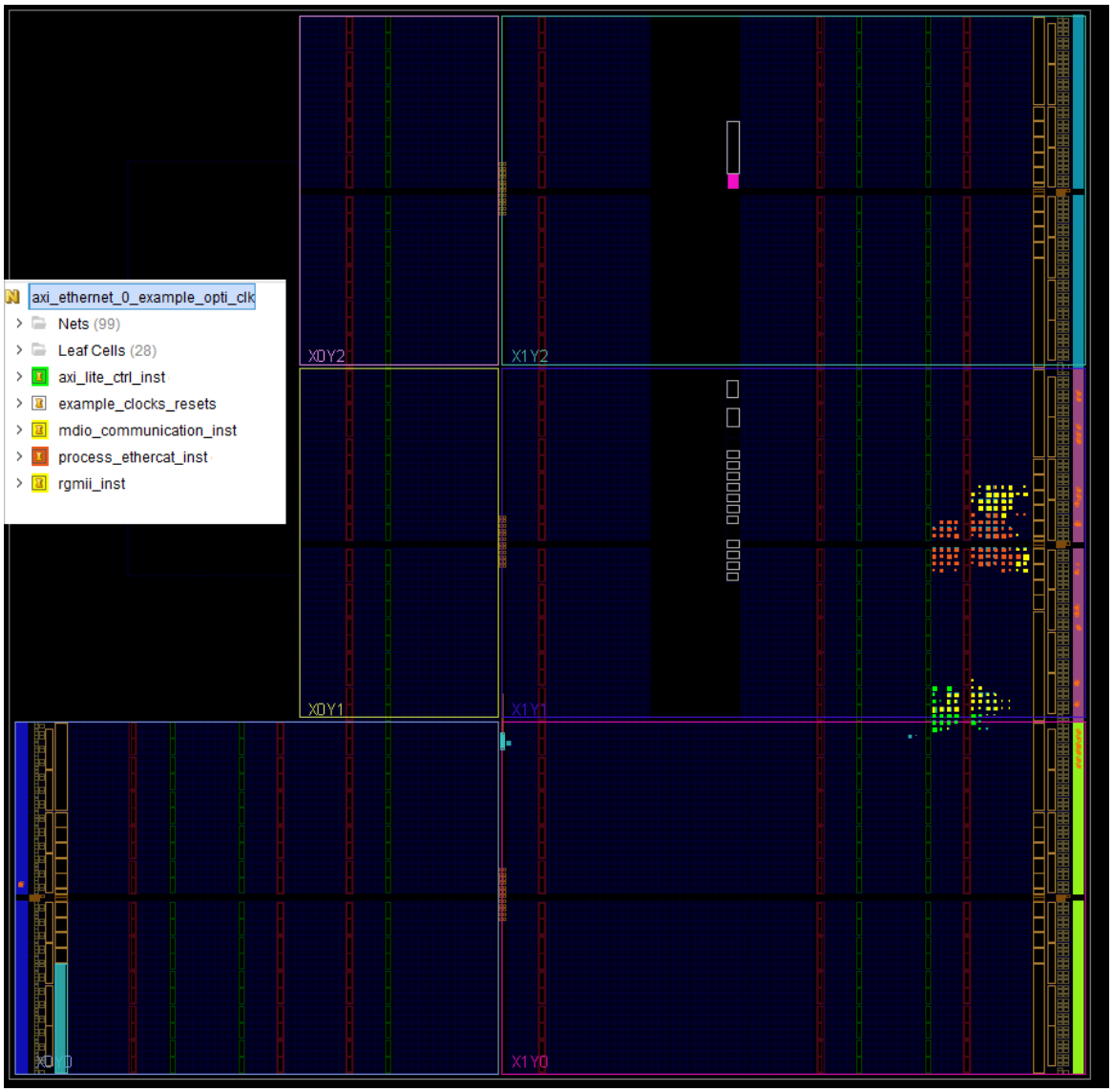


Figure 6.8: Design placement.

Chapter 7
Interface to CGRA

Compared to FPGAs, CGRAs have the advantage of processing data at word level, thus implying smaller configuration files and less configuration time. The architecture is able to load a new design in each clock cycle. This strong capability might increase the usage of CGRAs in the detriment of FPGAs. As EtherCAT receives a growing enthusiasm in industrial projects and some controllers used in this field require high computation power, providing an interface to CGRAs might also increase the interest in this reconfigurable technology. Each field device would process the measured data and compute control signals using a CGRA, while communicating with other field devices through EtherCAT protocol. Thus, defining an interface between the EtherCAT controller and such a reconfigurable architecture would be of benefit. The following Section introduces the design requirements, while the second Section describes its implementation. In the end, an evaluation of design performance is conducted.

7.1 Architecture requirements

The current project implements a peripherial for the CGRA architecture used in the AMIDAR project developed by the "Computer Systems" group of the Faculty of Electrical Engeneering, within the Technical University in Darmstadt [46]. For the above mentioned CGRA type, the peripherals cannot directly write data into the register file of the PEs. The current architecture implies that in order to process data from peripherals, the context must be first written accordingly. As the architecture is not yet well defined and it will undergo several modifications according to ongoing research studies, a simple, parameterizable interface is preferred for the current project.

The EtherCAT master as well as the CGRA must both have access to the data memory. A possible access conflict between the two must be supported. In contrast, the configuration memory of the EtherCAT slave must be accessed only by the network master, as it will be the only capable of controlling the communication's parameters.

As different software codes need a different number of PEs, in order to have an optimal runtime, the EtherCAT controller must support a variable number of PEs. Thus, the memory address must be structured in such a way that each PE would receive its own storage space and the various address spaces would be transparent to the PEs. The EtherCAT slave does not need to flag any new incoming data to the CGRA, as this information will be generated by the scheduler. Thus, the PEs will initiate on their own a memory access to the EtherCAT slave, according to their context. All of them must be able to request access to their own storage space in parallel.

7.2 Implementation

The design architecture fulfilling the requirements described in the above Section can be visualized in Figure 7.1.

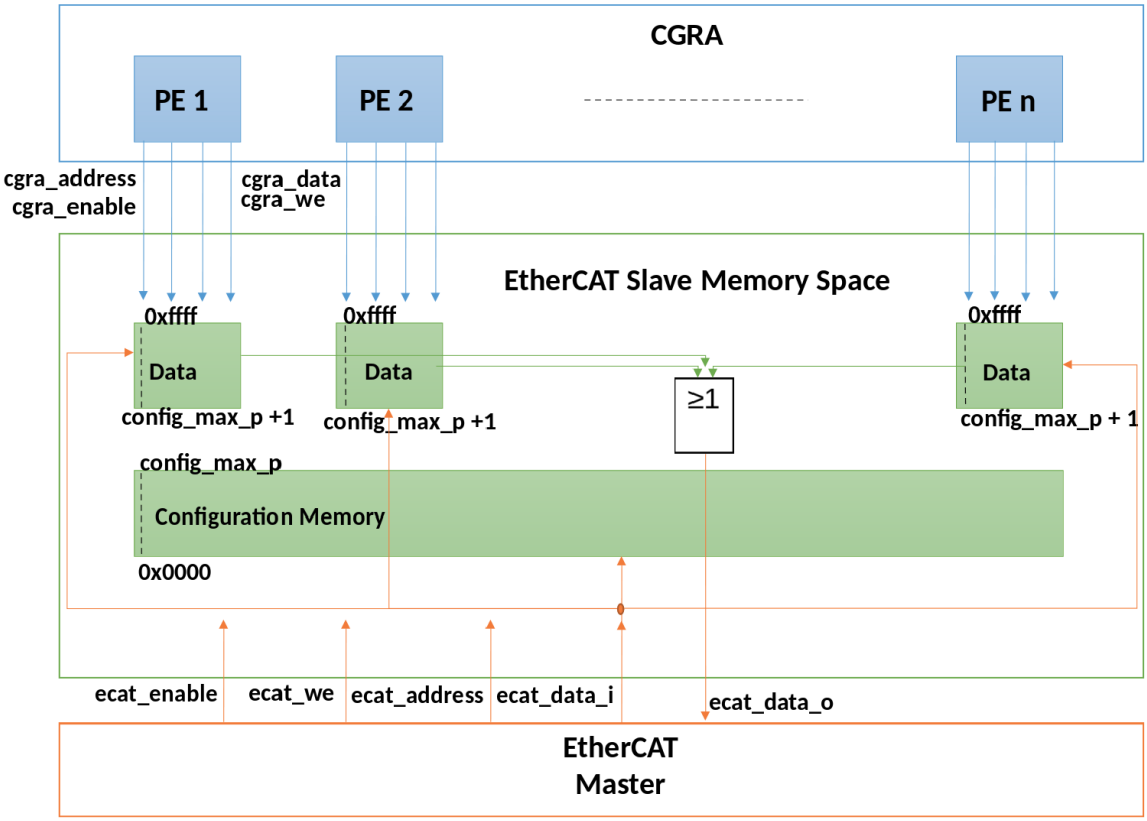


Figure 7.1: Implementation of the interface between CGRA and EtherCAT Slave.

The data coming from the CGRA is defined in Verilog as a big bus containing information bits from all the PEs. The same applies for the address, enable and write bits.

There are two types of memories defined in the design. One of them, called "config_reg_file_reg", stores the configuration information and is single port. The second one, storing user's data, is instantiated in the "data_memory" module and is a dual port. The later one must provide access to both the EtherCAT master as well as to the CGRA. Using the generate construct, the Verilog module is able to generate a variable number of "data_memory" modules. The PEs can access only the data stored by this modules and they do not see the address offset introduced by the configuration memory.

If a collision occurs, meaning a simultaneously write access from the EtherCAT master and the CGRA, the "data_memory" module is able to recognize it and gives priority to the PE. Additionally, it stores the data from the master in a FIFO. If no write accesses occur and there is still data in the buffer, the module will write this data into the memory. No collision occures if both the CGRA and the EtherCAT master try to read data from the memory or if they request a read and a write access simultaneously. Additionally, no conflict is detected if all the PEs write data at the same time, as their storage space is physically separated.

When sending data to the EtherCAT master, the module needs to choose between the output data of the "data_memory" modules. An "or" gate is used for this purpose. Thus, if a module is selected, the output data will correspond to the content of the memory. In contrast, if it is not selected, it will output zeroes, so that the result of the "or" function will correspond to the enabled module.

The EtherCAT master selects between the PEs through the incoming address. According to the EtherCAT protocol, the address bus has 32 bits. Out of these, the first eight will be used to recognize the EtherCAT slave, the following eight will differentiate between the PEs while the last sixteen bits will access the configuration or data memory space.

7.3 Design Evaluation

In this Section, the functionality of the design, the resource utilization and the maximum path delay are analyzed. The design under test can be connected to three PEs, but as stated in the previous Section, this number is variable.

Before running the synthesis, the design is tested to comply the requirements. A testbench conducting the following operations was established (data is given in decimal format):

- EtherCAT read request. Address = 257_{10}. Data = 24_{10}
- EtherCAT write request. Address = 257_{10}. Data = 24_{10}
- Generate Conflict. EtherCAT write request. Address = 258_{10}. Data = 24_{10}. CGRA write request. Address = 258_{10}. Data = 11_{10}
- Concurrent read requests
- EtherCAT write request. Address = 65796_{10}. Data = 24_{10}.

The results are shown in Figure 7.2. Should I make the simulation photo landscape? In this case maybe they will ask me why I didn.t attach it as Annex (Answer: because 60 pgs =)))

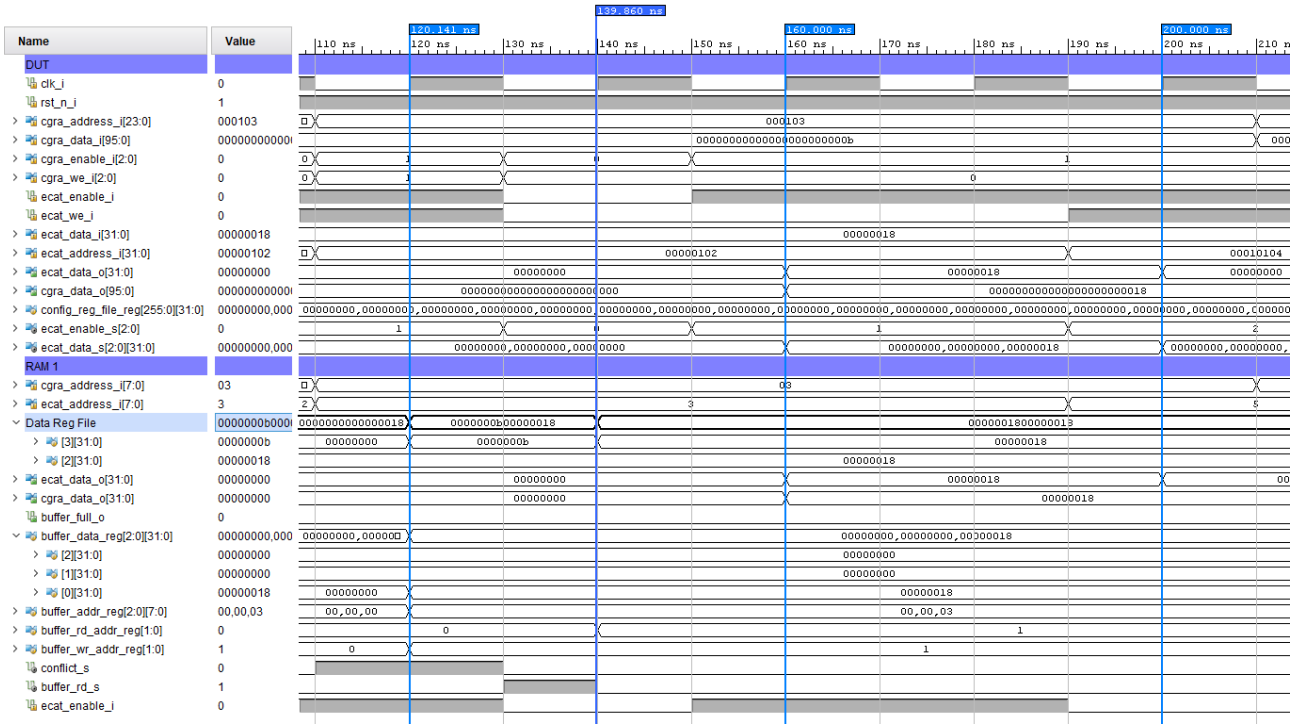


Figure 7.2: Simulation results of the design implementing the CGRA interface.

The first access is requested by the EtherCAT master. Its address is 257_{10}. Because it is higher than 255_{10}, an access to the data memory space at address 2_{10} is initiated. Moreover, due to the second highest byte, which is zero, the first PE is addressed.

The next access generates a conflict, as both the CGRA and the ETherCAT master try to write data to the same location. The least significant byte of the CGRA address bus is 3_{16}. This is the address information sent by the first PE and is equal to the address requested by the EtherCAT master (258_{10}-255_{10}). The access is pointed out in the simulation by the second cursor. The conflict signal is asserted and the design stores the data from the EtherCAT master into the buffer.

When no access to the memory is requested, the data stored in the buffer is written back to the memory. The event is pointed out by the yellow cursor in Figure 7.2, when the "Data Reg File" updates its entry at address 3_{10}.

The concurrent read requests pointed out by the third blue cursor do not generate a conflict. The memory returns both data, on the "cgra_data_o" bus as well as on the "ecat_data_o" bus.

The last EtherCAT write request accesses address $65796_{10} = 010104_{16}$, which enables the second PE, because the second highest byte is 01_{16}. Thus, this does not affect the memory address space of the first PE and no effect is seen in the corresponding storage elements.

After testing the functionality of the code, its performance regarding timing and resource usage must be analyzed. In FPGAs, storage elements are either distributed Random-Access Memorys (RAMs) or Block RAMs (BRAMs). The first type of storage strategy is useful in case of small designs and it usually gives better performance regarding speed. In case of large memories however, using LUTs as storage elements would require much hardware resources and would generate a slow design, due to routing. However, if asynchronous read is required, then distributed RAM shall be used [52].

The Vivado tool is able to conduct an analysis and decide which solution is optimal when choosing between distributed RAM and BRAMs. However, in order to be able to do that, the user must write the HDL code in such a manner that the tool would recognize the structure.

In the beginning, the interface was written without taking into consideration the BRAMs, but only the functionality. All the registers were initialized through synchronous reset. The corresponding resource usage is shown in Table 7.1. This design was used when runing the simulation shown in Figure 7.2.

After removing the reset signals from most of the storage elements, the resource usage reduced significantly. This paragraph is really short. I wanted to show that this was a separate test-case though. Should I add it to the previous one?

In both cases described above, distributed RAM was used as a storage element for the memories. In order to be able to use BRAMs, the format of the code must be updated. The EtherCAT master and the CGRA can access the memory at the same time. Thus, the corresponding hardware element must be dual port. The HDL format required for instantiating such BRAMs can be vizualized in Figure 7.3.

The resource usage in this case is strongly improved when it comes to number of LUTs.

After implementing the solution using dual port BRAMs, the functionality must be checked again. Accroding to Xilinx, access collisions might occur in case of dual port RAMs if three conditions are met simultaneously. One of them specifies that both access ports have different clock signals [53]. Because this is not the case for this design, reading data from an address written simultaneously by the other port does not flag a collision.

When it comes to timing performance, the following time constraint was defined in order to obtain data regarding the maximum allowed frequency: "create_clock -period 10.000 -name clk -waveform 0.000 5.000 [get_ports clk_i]". The results are shown in Figure 7.3.

The resource usage and timing performance given in this Chapter are in favour of the solution implying BRAMs. However, in case small memory area is required, the approach using distributed RAMs might give better results. An important thing to be underlined is the effect of the HDL format on the synthesized design. In this case, the less complex code leads to a higher resource usage. Additionally, initializing the memory content using a synchronous reset also has a big impact on the area.

Design Approach	Total Delay	LUTs	FF	BRAMs
Xilinx format code, BRAMs	3.857	177	108	3.5
Shortest Code, without reset	6.394	2422	204	
Shortest Code, with reset	4.849	19134	33140	

Table 7.1: Resource usage of the implemented interface to CGRA, using different programming approaches.

```
// Write data into the memory
always @(posedge clk_i) begin
    if(cgra_enable_i & cgra_we_i)
        data_reg_file_reg[cgra_address_i] <= cgra_data_i;
    else if(ecat_enable_i & ecat_we_i)
        data_reg_file_reg[ecat_address_i] <= ecat_data_i;
    else if(buffer_rd_s)
        data_reg_file_reg[buffer_addr[buffer_rd_addr_reg]]<= buffer_data[buffer_rd_addr_reg];
end

// Compute output data sent to the ECAT
always @(posedge clk_i) begin
    if(ecat_enable_i)
        ecat_data_o <= data_reg_file_reg[ecat_address_i];
    else
        ecat_data_o <= {data_width_p{1'b0}};
end

// Compute output data sent to the CGRA
always @(posedge clk_i) begin
    if(cgra_enable_i)
        cgra_data_o <= data_reg_file_reg[cgra_address_i];
    else
        cgra_data_o <=  {data_width_p{1'b0}};
end
```

```
173 // Compute data to write into memory
174 always @(*) begin
175     if(ecat_enable_i & ecat_we_i)
176         data_to_ram_B = ecat_data_i;
177     else if(buffer_rd_s)
178         data_to_ram_B =  buffer_data[buffer_rd_addr_reg];
179 end
180
181 assign enable_wr_ram_B = (cgra_enable_i & cgra_we_i) ? 0 :
182                          ((ecat_enable_i & ecat_we_i) || buffer_rd_s);
183 assign address_ram_B   = (ecat_enable_i) ? ecat_address_i :
184                           buffer_addr[buffer_rd_addr_reg];
185 assign enable_ram_B    =  ecat_enable_i || buffer_rd_s;
186
187 // Read/Write data into the memory. CGRA Access
188 always @(posedge clk_i) begin
189     if(cgra_enable_i) begin
190         if(cgra_we_i) begin
191             data_reg_file_reg[cgra_address_i]  <= cgra_data_i;
192             cgra_data_o                        <= cgra_data_i;
193         end else
194         cgra_data_o               <= data_reg_file_reg[cgra_address_i];
195     end
196 end
197
198 // Read/Write data into the memory. ECAT Access
199 always @(posedge clk_i) begin
200     if(enable_ram_B) begin
201         if(enable_wr_ram_B) begin
202             data_reg_file_reg[address_ram_B]   <= ecat_data_i;
203             ecat_data_o                        <= ecat_data_i;
204         end else
205         ecat_data_reg             <= data_reg_file_reg[address_ram_B];
206     end
207 end
208
209 always @(*) begin
210     if(ecat_enable_i)
211         ecat_data_o <= ecat_data_reg;
212     else
213         ecat_data_o <= {data_width_p{1'b0}};
214 end
```

Figure 7.3: Different code techniques. Right, taking into account only functionality and faster programming. Left, format sugested by Xilinx for instantiating BRAMs.

Chapter 8
Conclusion

8.1 Comparison to available solutions

8.2 Further improvements

8.3 Project Summary

References

[1] Ethernet FMC, "Ecat-2052," [Online; accessed September 1,2018]. [Online]. Available: https://www.icpdas-usa.com/ecat_2052.html

[2] Digilent, "The zedboard," [Online; accessed April 15, 2018]. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/zedboard/start

[3] Ethernet FMC, "The ethernet fmc board," [Online; accessed April 1, 2018]. [Online]. Available: https://opsero.com/product/ethernet-fmc/

[4] Toshiba, "Qosmio laptop," [Online; accessed April 1, 2018]. [Online]. Available: http://www.toshiba.com.ro/discontinued-products/qosmio-x70-a-11k/

[5] E. FMC, "Technical specifications of the ethernet fmc board." [Online]. Available: https://ethernetfmc.com/

[6] D. Chowdhury, *High Speed LAN Technology Handbook*. Springer Berlin Heidelberg, 2013. [Online]. Available: https://books.google.de/books?id=DdaoCAAAQBAJ

[7] EtherCAT Technology Group, "Ethercat slave implementation guide," [Online; accessed April 5, 2018]. [Online]. Available: http://www.ethercat.org/pdf/english/ETG2200_V2i0i0_SlaveImplementationGuide.pdf

[8] Beckhoff Automation GmbH & Co. KG, "Structure of the ethercat frames," [Online; accessed April 7, 2018]. [Online]. Available: https://download.beckhoff.com/download/document/io/ethercat-development-products/ethercat_esc_datasheet_sec1_technology_2i3.pdf

[9] Ethernet FMC, "Architecture of the ps-based solution," [Online; accessed April 7, 2018]. [Online]. Available: http://ethernetfmc.com/exampledesigns/

[10] H. Wang, Y. Rong, S. Liu, and J. Cui, "Fieldbus technology and rolling process automation," in *2010 International Conference On Computer Design and Applications*, vol. 4, June 2010, pp. V4–73–V4–76.

[11] Moore Industries-International, Inc., "Introduction to fieldbus." [Online]. Available: http://www.miinet.com/Portals/0/PDFs/Introduction_to_Fieldbus_White_Paper_Moore_Industries.pdf

[12] B. Žeželj and H. Hajdo, "Fieldbus diagnostic online solution program establishment at rijeka oil refinery," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2016, pp. 720–724.

[13] EtherCAT Technology Group, "Ethercat – the ethernet fieldbus." [Online]. Available: http://www.ethercat.org/pdf/ethercat_e.pdf

[14] A. Depari, P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, "Evaluation of timing characteristics of industrial ethernet networks synchronized by means of ieee 1588," in *2007 IEEE Instrumentation Measurement Technology Conference IMTC 2007*, May 2007, pp. 1–5.

[15] L. L. Bello, E. Bini, and G. Patti, "Priority-driven swapping-based scheduling of aperiodic real-time messages over ethercat networks," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 741–751, June 2015.

[16] EtherCAT Technology Group, "The ethernet fieldbus," [PG138 October 4, 2017]. [Online]. Available: https://www.ethercat.org/pdf/english/EtherCAT_Introduction_0905.pdf

[17] Beckhoff Automation GmbH & Co. KG, "Ethercat system documentation," [Version 5.2, May, 2017]. [Online]. Available: https://download.beckhoff.com/download/document/io/ethercat-terminals/ethercatsystem_en.pdf

[18] EtherCAT Technology Group, "Ethercat – the ethernet fieldbus." [Online]. Available: https://www.ethercat.org/pdf/english/ETG_Brochure_EN.pdf

[19] "Example design for using the quad gigabit ethernet fmc with the zynq ps hard gigabit ethernet macs (gem) and the gmii-to-rgmii ip." [Online]. Available: https://github.com/fpgadeveloper/ethernet-fmc-zynq-gem

[20] Beckhoff Automation GmbH & Co. [Online]. Available: https://www.beckhoff.com/EtherCAT/

[21] HMS Industrial Networks. [Online]. Available: https://www.anybus.com/

[22] "Rta's blog, the world of ethercat," accessed on 01.09.2018. [Online]. Available: https://www.rtaautomation.com/blog/the-world-of-ethercat/

[23] S. O. E. Master. [Online]. Available: https://openethercatsociety.github.io/doc/soem/index.html

[24] K. Reed, *Data Network Handbook: An Interactive Guide to Network Architecture and Operations*. Wiley, 1996. [Online]. Available: https://books.google.de/books?id=iyFNAAAAYAAJ

[25] A. Tanenbaum and D. Wetherall, *Computer Networks*. Pearson Education, 2012. [Online]. Available: https://books.google.de/books?id=IRUvAAAAQBAJ

[26] M. Arregoces and M. Portolani, *Data Center Fundamentals*, ser. Cisco Press Fundamentals Series. Cisco, 2003. [Online]. Available: https://books.google.de/books?id=DRIryrLoxKkC

[27] "Media independent interface." [Online]. Available: https://muller.academic.ru/dic.nsf/enwiki/646301#Media_Independent_Interface

[28] A. Mishra and D. Johnson, *White Space Communication: Advances, Developments and Engineering Challenges*, ser. Signals and Communication Technology. Springer International Publishing, 2014. [Online]. Available: https://books.google.de/books?id=G0vPBAAAQBAJ

[29] Marvell, "Alaska 88e1510/88e1518/88e1512/88e1514 datasheet - public," [Doc. No. MV-S107146-U0, Rev. A January 4, 2018].

[30] T. Instruments.

[31] Xilinx, Inc., "Axi 1g/2.5g ethernet subsystem v7.1, product guide," [PG138 October 4, 2017]. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ethernet/v7_1/pg138-axi-ethernet.pdf

[32] S. Sen, *Fieldbus and Networking in Process Automation*. CRC Press, 2017. [Online]. Available: https://books.google.de/books?id=rJLNBQAAQBAJ

[33] N. Mahalik, *Fieldbus Technology: Industrial Network Standards for Real-Time Distributed Control*. Springer Berlin Heidelberg, 2013. [Online]. Available: https://books.google.de/books?id=8Lj8CAAAQBAJ

[34] P. James Powell, "Why use profibus for process automation," accessed on 02.09.2018. [Online]. Available: https://www.automation.siemens.com/w1/efiles/automation-technology/pi/techn_publications/why_use_profibus_en.pdf

[35] "Rta's blog, profibus," accessed on 02.09.2018. [Online]. Available: https://www.rtaautomation.com/technologies/profibus/

[36] M. Felser, "The fieldbus standards: History and structures," 01 2002.

[37] E. A. Experts, "Major difference between foundation fieldbus and hart protocols," accessed on 02.09.2018. [Online]. Available: https://www.emersonautomationexperts.com/2013/digital-busses/foundation-fieldbus/major-difference-between-foundation-fieldbus-and-hart-protocols/

[38] C. Harris, "Ethernet as a leading machine automation protocol," 06.05.2016. [Online]. Available: https://www.controleng.com/single-article/ethernet-as-a-leading-machine-automation-protocol/f938757e9f0f154e1f94c2934e97b50d.html

[39] E. T. Group, "Ethercat – the ethernet fieldbus." [Online]. Available: https://www.ethercat.org/pdf/english/ETG_Brochure_EN.pdf

[40] "Simple open ethercat slave or soes."

[41] "Ethercat for factory networking. ethercat automation protocol (eap)," accessed on 01.09.2018. [Online]. Available: https://www.ethercat.org/download/documents/EtherCAT_EAP_EN.pdf

[42] W. Voss, *A Comprehensible Guide to Controller Area Network*. Copperhill Technologies Corporation, 2008. [Online]. Available: https://books.google.de/books?id=PU6ppO3XbUwC

[43] "Anybus ip," accessed on 01.09.2018. [Online]. Available: https://www.xilinx.com/products/intellectual-property/1-as47h2.html

[44] Christian Hochberger, "Lecture notes of course "high level synthesis" given at tu darmstadt in the winter semester 2017/2018."

[45] Dennis Leander Wolf, "Design and implementation of a generic cgra for hardware-synthesis on amidar," [Master Thesis at the Technical University Darmstadt, 28th September 2015].

[46] T. Darmstadt, "Amidar project." [Online]. Available: http://www.amidar.de/?node=6fe6&page=general&ln=de

[47] N. Sedcole, "Reconfigurable platform-based design in fpgas for video image processing." phD thesis (2006).

[48] Beckhoff Automation GmbH & Co. KG, "Ethercat slave controller. hardware data sheet section ii," [Version 2.7, July, 2013]. [Online]. Available: https://download.beckhoff.com/download/Document/io/ethercat-development-products/ethercat_esc_datasheet_sec2_registers_2i7.pdf

[49] ——, "Ethercat slave controller. hardware data sheet section i," [Version 2.3, February, 2017]. [Online]. Available: https://download.beckhoff.com/download/document/io/ethercat-development-products/ethercat_esc_datasheet_sec1_technology_2i2.pdf

[50] F. Miller, A. Vandome, and J. McBrewster, *Cyclic Redundancy Check*. VDM Publishing, 2009. [Online]. Available: https://books.google.de/books?id=oX_aQgAACAAJ

[51] "Ieee std. 802.3 media independent interface specification."

[52] Xilinx, "!!!!!!!!!!!! url still needs to be added !!!!"

[53] Xilinx, "Virtex-6 fpga block ram design advisory - address space overlap." [Online]. Available: https://www.xilinx.com/support/answers/34859.html