

# CHALMERS



## Real-Time Signal Processing Implementation for 100 Gb/s Fibre Communication

*Master's Thesis in Integrated Electronic Systems Design*

Fredrik Toft  
Niclas Rousk

CHALMERS UNIVERSITY OF TECHNOLOGY  
Department of Computer Science and Engineering  
Gothenburg, Sweden, 2011

The Author grants to Chalmers University of Technology the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology store the Work electronically and make it accessible on the Internet.

Real-Time Signal Processing Implementation for 100 Gb/s Fibre Communication

Fredrik Toft  
Niclas Rousk

© Fredrik Toft, July 2011  
© Niclas Rousk, July 2011

Examiner: Per Larsson-Edefors

Chalmers University of Technology  
Department of Computer Science and Engineering  
SE-412 96 Gothenburg Sweden  
Telephone + 46 (0)31-772 1000

## **Abstract**

With ever increasing demands on transmission rates, new ways of transmitting data through fiber are being researched. Up until now on-off keying has sufficed, but this modulation technique has limitations in terms of transmission rates. Complex modulation techniques like 16-QAM allow for much higher transmission rates, however, the computational burden will increase drastically at the receiver's end.

The main focus of this thesis is a feasibility study of implementing a 16-QAM 112-Gb/s DD-equalizer on an FPGA, considering throughput, area and power dissipation.

The approach used was to gradually change the algorithm in MATLAB to account for different hardware limitations of current FPGA technology. System functionality was proven in terms of BER. In this process, a method for compensating the limitation of phase feedback delay was devised.

Simulations of the equalizer in MATLAB show that the algorithm is compatible with current FPGA technology. FPGA mapping of the equalizer parts indicate that speed and area constraints can be met. Power dissipation should be further explored and optimized.

## **Acknowledgements**

First and foremost we would like to thank our supervisors for their help and guidance throughout the thesis:

Per Larsson-Edefors, Chalmers

Bengt-Erik Olsson, Ericsson

Further we would like to thank everyone at Acreo and Ericsson who has contributed to the algorithm and helped us with insightful feedback, especially: Jonas Mårtensson, Marco Forzati and Christina Larsson.

Our thanks also goes to Lars Svensson, Ioannis Sourdis and the VLSI workgroup at Chalmers for their interest and feedback.

Special thanks to friends and family for their support.

# Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introduction</b>                        | <b>1</b> |
| 1.1      | 100GET Project . . . . .                   | 1        |
| 1.2      | Problem Description . . . . .              | 1        |
| <b>2</b> | <b>Background</b>                          | <b>2</b> |
| 2.1      | System Overview . . . . .                  | 2        |
| 2.1.1    | Modulation . . . . .                       | 2        |
| 2.1.2    | Analog System Setup . . . . .              | 2        |
| 2.1.3    | Digital Signal Processing . . . . .        | 4        |
| 2.1.4    | Error Correction . . . . .                 | 5        |
| 2.2      | Hardware Overview . . . . .                | 5        |
| 2.2.1    | Hardware Platforms . . . . .               | 5        |
| 2.2.2    | FPGA:s and VHDL . . . . .                  | 5        |
| 2.2.3    | Hardware Considerations . . . . .          | 6        |
| <b>3</b> | <b>Method</b>                              | <b>7</b> |
| 3.1      | Prestudy . . . . .                         | 7        |
| 3.2      | MATLAB . . . . .                           | 7        |
| 3.3      | Hardware Implementation . . . . .          | 7        |
| 3.4      | Test Setup . . . . .                       | 8        |
| <b>4</b> | <b>Signal Processing</b>                   | <b>9</b> |
| 4.1      | Introduction . . . . .                     | 9        |
| 4.2      | Equalizer Overview . . . . .               | 9        |
| 4.2.1    | FIR Filter . . . . .                       | 9        |
| 4.2.2    | Phase Recovery Loop . . . . .              | 10       |
| 4.2.3    | Decision . . . . .                         | 10       |
| 4.3      | Hardware Impact on DSP Algorithm . . . . . | 10       |
| 4.3.1    | Parallelization . . . . .                  | 10       |
| 4.3.2    | Filter Taps . . . . .                      | 11       |
| 4.4      | Simulations . . . . .                      | 12       |
| 4.4.1    | Filter Taps . . . . .                      | 12       |
| 4.4.2    | Coefficient Delay . . . . .                | 13       |
| 4.4.3    | Phase Feedback Delay . . . . .             | 13       |
| 4.4.4    | Fixed Point Representation . . . . .       | 14       |
| 4.5      | Algorithm Improvements . . . . .           | 14       |
| 4.5.1    | Effect of Phase Delay . . . . .            | 15       |
| 4.5.2    | Rotation Compensation . . . . .            | 15       |
| 4.5.3    | Simulation Results Using PREST . . . . .   | 16       |
| 4.5.4    | Final Simulations . . . . .                | 16       |

|          |                                      |           |
|----------|--------------------------------------|-----------|
| <b>5</b> | <b>Hardware</b>                      | <b>18</b> |
| 5.1      | FIR with LMS Updating . . . . .      | 18        |
| 5.2      | Phase Recovery . . . . .             | 18        |
| 5.3      | Area and Power . . . . .             | 19        |
| <b>6</b> | <b>Discussion</b>                    | <b>21</b> |
| <b>7</b> | <b>Conclusion</b>                    | <b>23</b> |
| <b>8</b> | <b>Future Work</b>                   | <b>24</b> |
| 8.1      | Optimizations . . . . .              | 24        |
| 8.1.1    | FIR Filter . . . . .                 | 24        |
| 8.1.2    | Feedback Loops . . . . .             | 24        |
| 8.2      | Compatibility with 400G-1T . . . . . | 25        |
|          | <b>Bibliography</b>                  | <b>26</b> |

# Nomenclature

|        |  |
|--------|--|
| 100GET | 100 Gbit/s Carrier-Grade Ethernet Transport Technologies |
| ASIC   | Application Specific Integrated Circuit                  |
| BER    | Bit Error Rate   |
| BPD    | Balanced PhotoDetectors                                  |
| BPSK   | Binary Phase Shift Keying                                |
| CD     | Chromatic Dispersion                                     |
| CMA    | Constant Modulus Algorithm                               |
| CORDIC | COordinate Rotation DIgital Computer                     |
| DD     | Decision Directed  |
| DSP    | Digital Signal Processing                                |
| ECC    | Error Correcting Code                                    |
| FEC    | Forward Error Correction                                 |
| FFT    | Fast Fourier Transform                                   |
| FIR    | Finite Impulse Response                                  |
| FPGA   | Field Programmable Gate Array                            |
| I/Q    | In phase/ Quadrature                                     |
| LMS    | Least Mean Square  |
| LUT    | LookUp Table   |
| OOK    | On-Off Keying  |
| OSNR   | Optical Signal to Noise Ratio                            |
| PLL    | Phase Lock Loop  |
| PMD    | Polarization Mode Dispersion                             |
| PRBS   | Pseudo Random Binary Sequence                            |
| PREST  | Phase Rotation ESTimation                                |
| PSK    | Phase Shift Keying                                       |
| QAM    | Quadrature Amplitude Modulation                          |

|       |                                       |
|-------|---------------------------------------|
| QPSK  | Quadrature Phase Shift Keying         |
| RF    | Radio Frequency                       |
| SAIF  | Switching Activity Interchange Format |
| SNR   | Signal to Noise Ratio                 |
| VHDL  | VHSIC Hardware Descriptive Language   |
| VHSIC | Very High Speed Integrated Circuit    |

# List of Figures

|  |    |
|--|----|
| 2.1.1 Complex modulation constellations . . . . .  | 2  |
| 2.1.2 Analog system overview . . . . .   | 3  |
| 2.1.3 Digital system overview . . . . .  | 4  |
| 4.2.1 Overview of the equalizer. . . . .   | 10 |
| 4.2.2 Butterfly structure FIR filters. X1 and X2 corresponds to different polarizations. . . . . | 11 |
| 4.2.3 16 QAM constellation map with gray mapped symbols. . . . .                                 | 12 |
| 4.4.1 Effect of varying the number of filter taps. . . . .                                       | 13 |
| 4.4.2 BER versus number of symbols used for smoothing. . . . .                                   | 13 |
| 4.4.3 Effect of delaying the update of filter coefficients. . . . .                              | 13 |
| 4.4.4 Effect of delaying the phase feedback loop. . . . .  | 14 |
| 4.5.1 Effect of delaying the phase feedback loop. Left to right: 1, 2, 3 cycles delay.           | 15 |
| 4.5.2 PREST1 and PREST2 mapping . . . . .  | 16 |
| 4.5.3 Comparison of unmodified and PREST versus phase delay. . . . .                             | 16 |
| 5.2.1 Hardware view of phase feedback . . . . .  | 19 |

# List of Tables

- 2.1.1 Repetition code decision (3,1) . . . . . 5
  
- 4.3.1 Hardware resources for different FPGA models. . . . . 11
- 4.4.1 Difference in BER using fixed and floating point. . . . . 14
- 4.4.2 Number of bits used in different parts of the system. . . . . 14
- 4.5.1 Simulation using fixed point and counter. . . . . 17
  
- 5.3.1 Area estimation of the equalizer . . . . . 20
- 5.3.2 Active power estimation of the equalizer . . . . . 20

# Chapter 1

## Introduction

### 1.1 100GET Project

The demand for internet capacity close to doubles each year, much due to video streaming sites. To cope with this increase a large European collaboration, 100 Gbit/s Carrier-Grade Ethernet Transport Technologies (100GET) [1], was formed to develop 100 Gb/s Ethernet communication. This collaboration is split into several subgroups which focus on different problem areas including metro and long distance communication. Ericsson is the main partner of one of these groups, 100GET-ER, and aims to develop a cost effective system with the help of their leading expertise in Radio Frequency (RF)-communication.

In 10 Gb/s communication On-Off Keying (OOK), light on/off, can be used but this becomes increasingly difficult with higher transmission rates [2]. This is where Ericsson's knowledge of RF-communication comes in, as it is possible to use similar complex modulations in optics as the ones used in RF to enhance throughput to over 100 Gb/s [3].

One problem of moving from a simple modulation like OOK to a more complex, e.g. 16-Quadrature Amplitude Modulation (QAM), is the large increase in computational demand[4].

The research team at Ericsson has completed a successful field trial over a 824 km fiber cable [5]. The Digital Signal Processing (DSP) part of the system is however still done as post-processing in MATLAB. But a MATLAB implementation of the DSP can not be used in the final product. Normally an Application Specific Integrated Circuit (ASIC) will be designed to solve this as it is the most powerful platform available. ASICs are however extremely costly both in time and resources to develop and can not be changed once manufactured. Over the past years the computational capacity of a Field Programmable Gate Array (FPGA) has begun to catch up. Although it will never be as fast or efficient as an ASIC the FPGA is gradually becoming an option in high end systems.

### 1.2 Problem Description

The aim of this thesis is to research the feasibility of implementing the DSP part with FPGA technology. The entire DSP system is too large to be covered in any depth with the time available. This leads to the limitation of only considering the equalizer as it is the most computationally heavy part of the system. The main focus is to test the possibility of implementing the equalizer algorithm and still have a stable system. Another aspect of the thesis is to find a methodology for developing future systems in MATLAB with hardware taken into consideration.

# Chapter 2

## Background

### 2.1 System Overview

#### 2.1.1 Modulation

OOK is not enough to reach 100 Gb/s, thus, a higher modulation format is needed. In phase/ Quadrature (I/Q) data is a modulation format which represents amplitude and phase but is stored as a complex number [6]. This is commonly used to code data with both amplitude and phase modulation. To generate data you send the In-phase (real) data on the carrier and Quadrature (imaginary) on the carrier delayed by  $90^\circ$ .

By using a higher modulation format more bits/sample can be sent, for example each sample of Quadrature Phase Shift Keying (QPSK) represents 2 bits and 16-QAM represents 4 bits. This has the obvious advantage of having higher transmission rates using lower baud rates (symbols per second), but the distance between each symbol in the constellation is reduced and thus requires a higher Signal to Noise Ratio (SNR).

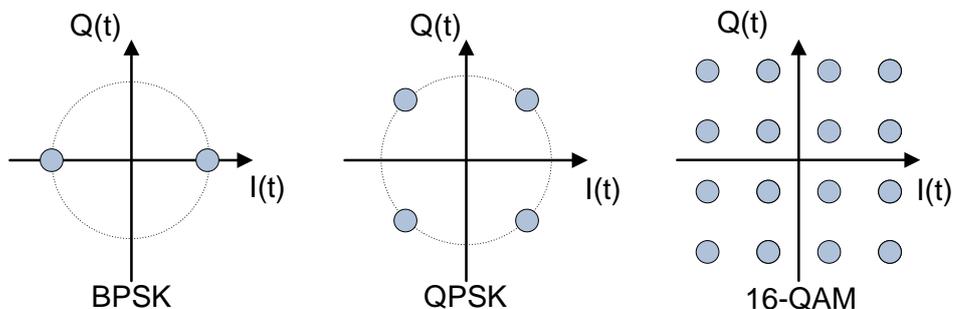


Figure 2.1.1: Complex modulation constellations

#### 2.1.2 Analog System Setup

All test data used in the system is generated by a 7 Gb/s Pseudo Random Binary Sequence (PRBS)-generator which gives a 1023 bit string of both data and the inverse of the same data to transmit on two different channels [7]. A Binary Phase Shift Keying (BPSK) signal is generated by using XOR gates with data and clocks as inputs. The clocks used are 14 GHz and 24 GHz, the centre of each band. The BPSK is converted to a QPSK signal by splitting it into two and giving one signal a 90 degree phase shift and delaying it about half the PRBS word for de-correlation. It is further split into two and delayed

for de-correlation, one half is amplified by 3 dB and added together in phase [2]. Both channels are filtered through bandpass filters before they are added together into the final signal. In the optical transmission the signal is divided into two polarizations and the data is again delayed for de-correlation. These polarizations are added together and then transmitted through an 824 km fiber from Stockholm to Hudiksvall and back [5].

On the receiver side the signal is amplified and filtered through an optical band-pass filter. A local oscillator laser is used as coherent detection of both polarizations, and the optical signal is converted to an electric signal by two Balanced PhotoDetectors (BPD). This signal is sampled with a 50 GS/s oscilloscope with a maximal resolution of 8 bits and a bandwidth of 20 GHz.

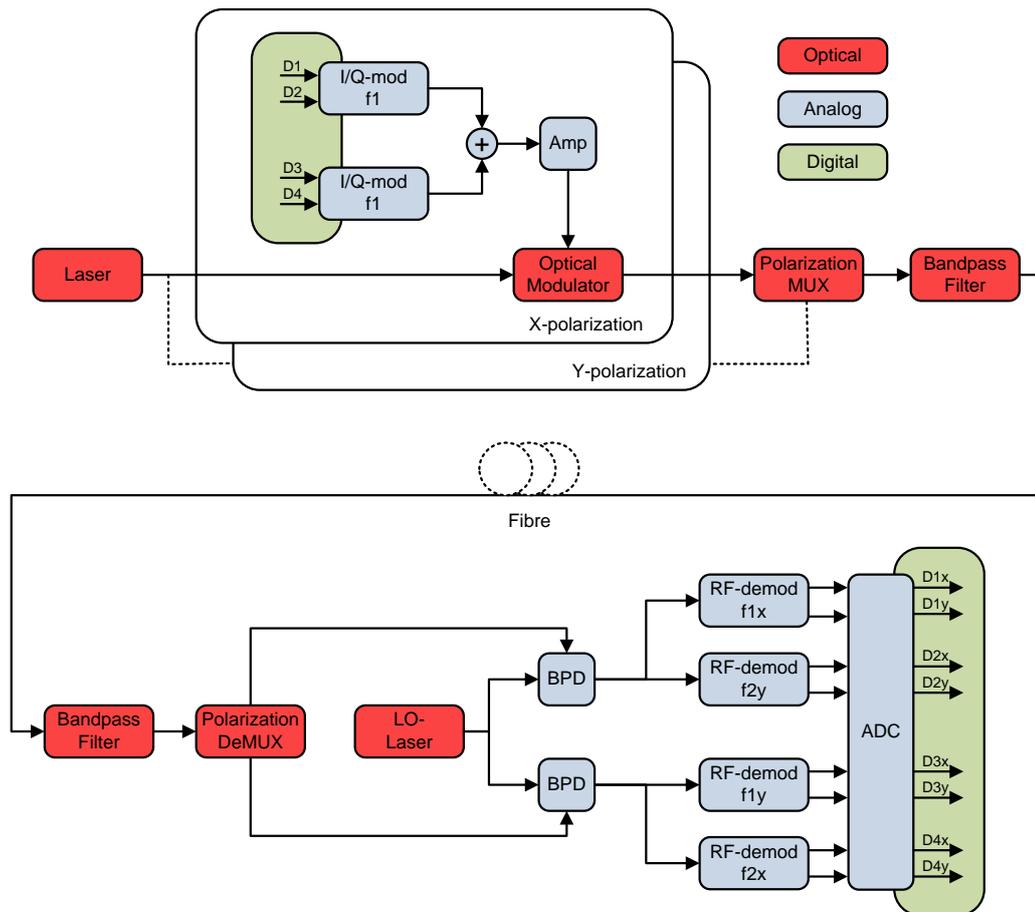


Figure 2.1.2: Analog system overview

There are two effects modifying the speed at which light travel through the fiber, and this introduces noise in the system. Chromatic Dispersion (CD) is the effect of different frequencies of light traveling at different speeds, and Polarization Mode Dispersion (PMD) is the effect of different polarizations travelling at different speeds. There are non-linear effects as well; cross phase modulation is one of them and is caused by interaction between the two phases. These impairments have become more important to consider as baud rates have increased and higher modulations demand higher Optical Signal to Noise Ratio (OSNR).

### 2.1.3 Digital Signal Processing

Data from the oscilloscope is sent to the computer in batches for processing in MATLAB. As neither the transmission rate to the PC nor its computational speed can achieve 100 Gb/s, everything is done as post-processing. The two frequency bands are separated by demodulating each channel from 14 GHz and 24 GHz down to baseband. This is done by digital I/Q down-conversion and low-pass anti-aliasing filtering. I and Q are represented by a complex number for each sample.

CD is compensated for in a Finite Impulse Response (FIR) filter in the frequency domain, this is done in a static filter to remove most of the effect[8]. The residual CD is removed in the later adaptive filters [9].

The carrier frequency recovery starts by taking the Fast Fourier Transform (FFT) of the data raised to the power of four to move the carrier frequency outside the spectrum of the data[10]. The frequency can then be isolated and is divided by four to provide a frequency estimation. This method allows for recovery of the carrier frequency only using the transmitted data.

As the sample rate is 50 GS/s and the baud rate only 7 GS/s the signal needs to be resampled. By sorting the samples into batches and aligning them based on phase, it is possible to extract a synchronous 2 samples/symbol signal [11].

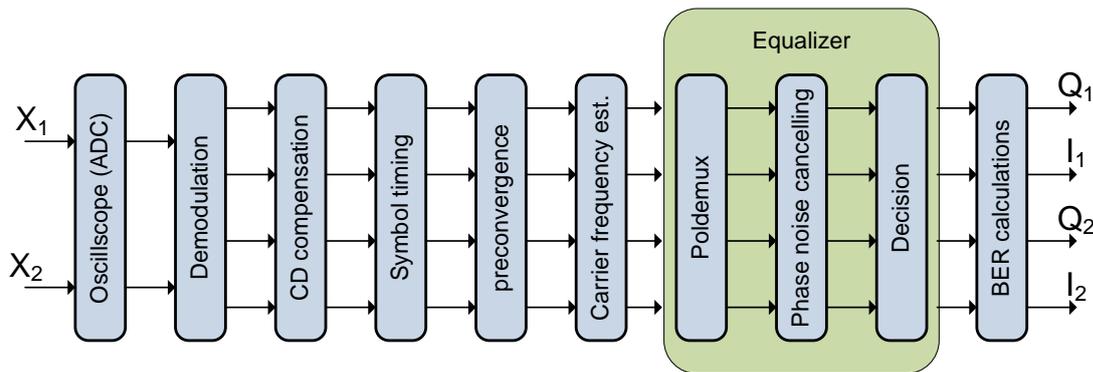


Figure 2.1.3: Digital system overview

The equalizer is constructed of a butterfly structure FIR filter for polarization demultiplexing and a phase recovery loop for phase-noise cancellation. To initialize the system, the Constant Modulus Algorithm (CMA) is used as a first estimation of the FIR filter coefficients [12][13]. CMA assumes that all points in the constellation have equal amplitude and is commonly used for QPSK modulation.

In this system CMA is only used for pre-convergence and then the system switches to a Decision Directed (DD) mode. In this mode the constellation is divided into 16 boxes and depending on which of these the sample is assigned to, an error is estimated. This method is much more accurate for 16-QAM modulation but may be unable to lock on a constellation since it is slow to converge.

Phase noise compensation is calculated by dividing the sample by the decided value to generate an estimation of the error. This is averaged over the last  $N$  samples and fed back as a phase correction.

The last stage is to check whether the signal is correct or not. The signal is a known pre-generated PRBS. By correlating the processed signal with the stored data and counting the number of bits that does not align, the Bit Error Rate (BER) can be calculated. This is used as a way of checking the quality of the signal and processing in an easy manner.

### 2.1.4 Error Correction

After the decision mapping to the 16-QAM constellation there will be errors in the data, but there are ways to attenuate this problem. By adding some overhead for Error Correcting Code (ECC) it is possible to greatly reduce the amount of errors in the received data. A simple example of this is repetition code. By coding each bit with three equal bits and use majority voting, in each 3 bit sequence, one bit error can be tolerated.

|          |     |     |     |     |     |     |     |     |
|----------|-----|-----|-----|-----|-----|-----|-----|-----|
| received | 000 | 001 | 010 | 100 | 011 | 101 | 110 | 111 |
| decision | 0   | 0   | 0   | 0   | 1   | 1   | 1   | 1   |

Table 2.1.1: Repetition code decision (3,1)

This is a very crude method of ECC which require  $2/3$  of the transmission rate as overhead, so it is not very useful for high speed applications. There are much more advanced methods for error correction, one being Forward Error Correction (FEC) that uses blocks of ECC interleaved with the data. These bits are each linked to several of the data bits making the FEC require much less overhead.

With the CI-BCH eFEC it is possible to use only 6.7% overhead and improve BER from  $4.6 \cdot 10^{-3}$  to  $1 \cdot 10^{-15}$  [14], but this does depend on the type of error and other factors. To have a low BER so that the FEC will work optimally is important. This sets the upper limit on the BER we can accept in the system, so the goal is to stay below  $3 \cdot 10^{-3}$  to allow for some margin of error. FEC systems are usually sold on a separate ASIC but are being converted to VHDL in order to be integrated with large FPGA systems.

## 2.2 Hardware Overview

### 2.2.1 Hardware Platforms

There are many hardware platforms available today and they are all specialized for different tasks. General-purpose processors are extremely flexible and easy to program, but lack the raw computational power needed. Using a graphics card is an interesting concept as this is highly parallel and has a large amount of computational power.

When it comes to extreme demands on performance, power and size, an ASIC is the most powerful solution available, since it can be designed to do one task only and be extremely efficient for it. The drawback is the large development cost and that it is not configurable once manufactured, so if a mistake is made, large parts will have to be redesigned and the chip remanufactured.

FPGAs have a massive amount of computational power which is reconfigurable to be both highly parallel and pipelined. As they are reprogrammable they are easier and much cheaper to develop than ASICs and can fit most given functions. They are however more expensive per unit and are not as energy efficient nor fast as an ASIC.

### 2.2.2 FPGAs and VHDL

VHSIC Hardware Descriptive Language (VHDL) was originally developed as a method to describe the functionality of circuits instead of having a large, complex manual. Another goal was to make it implementation independent so it does not matter which platform you implemented it on as it only describes function. VHDL is used to develop and implement systems and can be used in FPGA and ASIC development.

The FPGA consists of small, usually 4-6 bits, LookUp Table (LUT) in series with a D-flip-flop which can be bypassed if desired [15]. These blocks are placed in larger clusters with programmable interconnects between them. In modern FPGAs dedicated functions

are put in special hardware blocks, such as multipliers, memory or special I/O functions. These are important to both reduce the area, power and improve the speed of the system as you can build them more efficiently in fixed hardware.

The FPGA differs from a processor since you implement a structure for the processing of data rather than instructions that the processor performs serially. It is even possible to implement the structure of a processor on an FPGA. This may not be as efficient as using a general-purpose processor but can still be used for rapid prototyping or as a control module.

### **2.2.3 Hardware Considerations**

When developing a system for an FPGA it is important to use pipelining to increase the throughput. By serializing a large function into two, the maximal throughput is doubled but the latency is also increased. There are some limitations as to how much you can pipeline a system. Some blocks, like FPGA hardware multipliers, can not be serialized and there is a maximal clock frequency that the system can handle.

Another method of increasing throughput is by building parallel systems. This approach is not limited by a minimum size of blocks or clock frequency but instead the implementation area grows linearly with the number of parallel stages. Some parts which depend on previous data, like feedback loops, may prove harder to parallelize. This is since a delay resulting from the calculations performed in the feedback loop will be introduced, and this might affect the system performance.

In the system the sample rate is 7 GS/s and the max clock frequency is 600 MHz, so pipelining alone will not be enough to reach this demand. Therefore it is needed to process at least 12 samples each clock cycle in parallel. Pipelining is also used as it is impossible to do all the processing needed in a single cycle.

# Chapter 3

## Method

### 3.1 Prestudy

To be able to build a representation of the system in VHDL, a good knowledge of the underlying system and the sources of noise are important. To fill the gaps of knowledge, a thorough prestudy was conducted. During the prestudy, possibilities and limitations of current hardware technology were investigated, focusing on high end FPGAs. The signal processing algorithm was also studied. Implementing the entire system is too large for a single thesis; hence a limitation was set to focus primarily on the most computationally intense part of the system, the equalizer.

For an easy overview the work flow will be split into “MATLAB” and “Hardware Implementation” but in the project these are intertwined. The hardware puts limitations on how much that can be done each cycle, how much that is possible to fit on a chip and how to implement different equations. MATLAB is then used for fast testing of different setups and changes in the system, as this might require extensive work in VHDL.

### 3.2 MATLAB

When developing or changing an algorithm it is important to have a way of measuring the effect of the changes. Throughout this project relative change in BER has been used to determine the effect a modification has had on the system.

The first step was to investigate the effect of changing different variables in the system, like filter taps and smoothing factor, to see how it affected the system performance. Further, the effect of delays in the feedback loops was investigated. In the last phase, the system was modified to replace hardware inefficient functions like exponential, argument and division with LUTs or specially designed functions.

The simulation data that were used is data transmitted in the test system setup described in section 2.1.2.

### 3.3 Hardware Implementation

Using an FPGA instead of post-processing the data in MATLAB introduces limitations on performance. There is a challenge in finding a balance between hardware efficiency and signal quality. During the process of developing a hardware description, each finished sub-block or system was tested against a corresponding MATLAB representation to ensure that it produced the correct result.

Development of the VHDL code was started by building synthesizable basic key components such as complex math and LUTs. After verification these blocks were combined into smaller subsystems and further into larger parts of the system as FIR filters and the

phase feedback loop. In the filter bank the main concern is area and power so this subsystem was designed with this in mind, prioritizing area and power efficiency over latency. The phase feedback is designed for minimal latency as it is only a small part of area and power consumption.

Instead of developing a VHDL code manually there is a possibility of using the Xilinx System Generator [16]. This takes a MATLAB Simulink representation of the system and translates that into a VHDL code. This allows for fast prototyping of DSP systems but it would still require a remake of the system into Simulink; this approach was taken in an internship at Ericsson [17]. However building the system bottom-up in VHDL gives a greater degree of freedom and control over the generated hardware. Hence this method was chosen over System Generator.

### 3.4 Test Setup

Test data used in the system is generated as a 1023 bit long PRBS, this data is modulated as described in section 2.1.2 and sent through the fiber. There is a 824 km fiber connection between Stockholm and Hudviksvall that is used in these tests so there is an actual hardware system generating the test data. This data is sampled in batches for post-processing in MATLAB, these batches are 5 mega samples large.

The testing is done as an iterative process, changing one variable at the time to isolate the effect of that variable. There are however variables that are linked, e.g. smoothing and phase delay, the effect of these changes depending on the value of the other. When assessing results, changes in BER compared to the original MATLAB code is used as a reference. This is done as no simple method of assessing signal quality is available.

# Chapter 4

## Signal Processing

### 4.1 Introduction

A general overview of the processing steps required is shown in figure 2.1.3 on page 4. Efforts have been focused on implementing the equalizer, phase recovery loop and decision part of the system. For the equalizer, only the Decision-Directed mode of operation has been considered.

### 4.2 Equalizer Overview

The system performs adaptive equalization using four adaptive FIR filters in a butterfly structure, see figure 4.2.2. The input to the four FIR filters is the received I/Q data corresponding to the two different polarizations. The data has been demodulated, CD compensated and re-timed before arriving at the FIR filter. A basic overview of the equalizer is shown in figure 4.2.1 on the next page.

The task of the equalizer is to perform demultiplexing of the signal and mitigate any linear impairment such as PMD and residual CD [9]. The FIR filter takes two samples per symbol as input and outputs one sample per symbol, performing an effective downsampling of two.

#### 4.2.1 FIR Filter

A general overview of the FIR butterfly structure is shown in figure 4.2.2. The filter coefficients are updated using a Least Mean Square (LMS) algorithm to find the coefficients that will produce the smallest error between the desired signal and the actual signal. The cost function  $\varepsilon$  is used to update the filter coefficients where the new coefficients are calculated as:

$$w_{mn}^{k+1} = w_{mn}^k + \mu \varepsilon_m x_n, \quad m = 1, 2; \quad n = 1, 2; \quad (4.2.1)$$

where  $x_n$  is the input to the FIR filter,  $\mu$  is the equalizer step size,  $w_{mn}^k$  is the old filter coefficients and  $w_{mn}^{k+1}$  is the new filter coefficients [9]. The subscript indices  $m$  and  $n$  correspond to the two polarizations and index  $k$  represents symbol number.

The cost function  $\varepsilon$  used in the equalizer is the difference between the actual output  $y^k$  and the desired output  $d^k$ . A rotation by  $\phi^k$  (described in section 4.2.2 on the following page) is also applied to the LMS error in order to decouple the adaptation of the equalizer's taps from the phase noise tracking. The error is calculated as:

$$\varepsilon_m = e^{j\phi^k} (d_m^k - y_m^k) \quad (4.2.2)$$

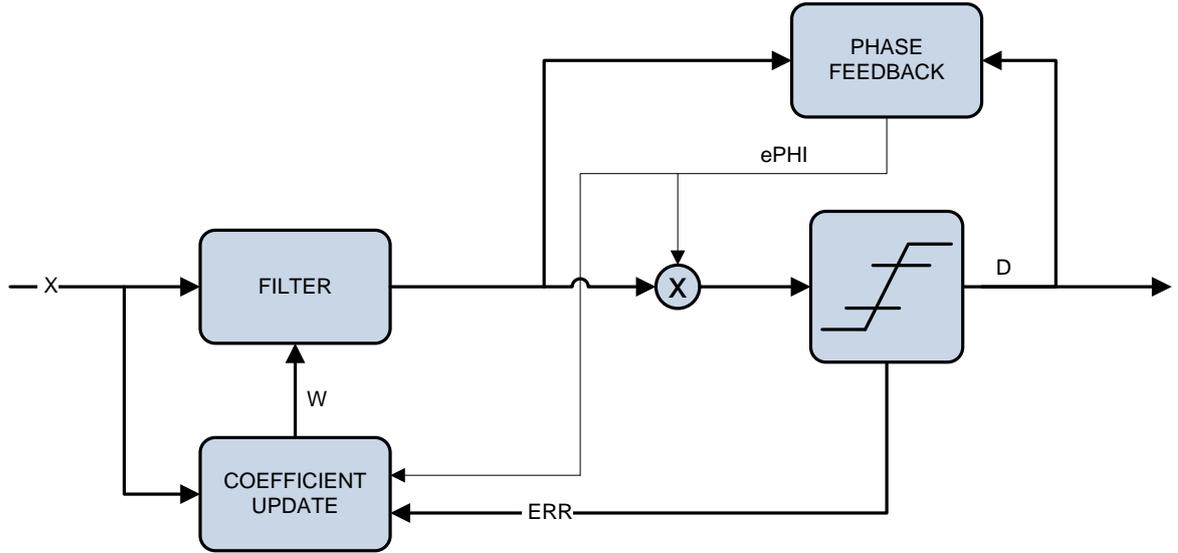


Figure 4.2.1: Overview of the equalizer.

### 4.2.2 Phase Recovery Loop

Before taking any decision on which symbol the output of the FIR filter corresponds to, compensation for varying frequency offset and laser phase noise must be tracked and compensated for. The phase error is estimated from the  $N$  previous symbols ( $N$  is called smoothing factor) and subtracted from sample  $k$ , polarization  $m$  as [9]:

$$\phi_m^k = \arg \left( \sum_{i=1}^N y^{k-i} / d^{k-i} \right) \quad (4.2.3)$$

The rotation  $\phi_m^k$  is fed back and applied after the FIR filter, before the decision part. It is applied by multiplying the filter output with  $e^{-j\phi_m^k}$ .

### 4.2.3 Decision

The decision part of the system is a simple level comparator that maps the filtered and phase noise compensated output  $y_m^k$  to a point in the 16 QAM constellation. Gray mapping of the constellation diagram is used to improve the BER, since the closest points to each symbol only differs by 1 bit.

## 4.3 Hardware Impact on DSP Algorithm

In the task of transferring the algorithm implemented in MATLAB to a real-time implementation in FPGA hardware, simulations were performed to emulate the effect resulting from the limitations of current FPGA technology. A number of limiting factors arise due to the high throughput demand on the system combined with the FPGA platform.

### 4.3.1 Parallelization

In order to reach the throughput requirements of 112 Gb/s on current FPGA technology, the algorithm must calculate symbols in parallel. The latest FPGA:s from Xilinx has a

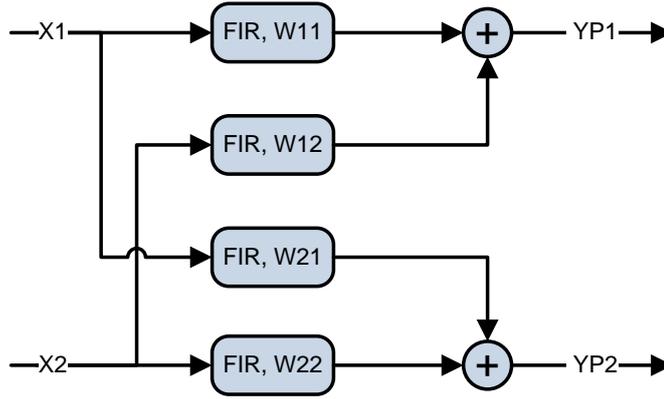


Figure 4.2.2: Butterfly structure FIR filters. X1 and X2 corresponds to different polarizations.

maximum clock frequency of 600 MHz [18]. Calculating symbols at this rate would give a bit rate of: 2 frequency channels  $\cdot$  2 polarizations  $\cdot$  4 bits per symbol = 9.6 Gb/s. For a throughput of 112 Gb/s the system has to calculate  $112/9.6 = 11.67$  samples in parallel, rounded upwards makes it 12 samples in parallel.

### 4.3.2 Filter Taps

One of the limiting factors is the number of multiplications that has to be done in the FIR filter. The number of taps in the filter determines how many multiplications that needs to be performed for each sample. The FPGA has a fixed number of dedicated hardware multipliers, called DSP48 slices [19]. The available number of DSP48 slices varies for different FPGA models. Table 4.3.1 lists some FPGA models from Xilinx and their respective number of DSP48 slices. This means that the number of taps that can be used in the FIR filter depends on the number of available hardware multipliers.

| Model  | XC6VVSX475T | XC7VX550T | XC7VX850T | XC7VX1140T |
|--------|-------------|-----------|-----------|------------|
| DSP48  | 2016        | 2880      | 3960      | 5280       |
| Slices | 74400       | 86100     | 133500    | 178000     |

Table 4.3.1: Hardware resources for different FPGA models.

The input to the FIR filter is I/Q data, represented by complex numbers. Multiplying two complex numbers  $x = a + jb$  and  $y = c + jd$  yields the result:

$$xy = (a + jb)(c + jd) \quad (4.3.1)$$

$$= ac + jbc + jad - bd \quad (4.3.2)$$

$$= (ac - bd) + j(ad + bc) \quad (4.3.3)$$

Implementing the last line of the above equation in hardware would require four multipliers and two adders. Since the number of multipliers is limited in an FPGA, performing as few multiplications as possible in the hardware implementation is desired. The above expression can be manipulated to use three multiplications and four adders:

$$\Re[(a + jb)(c + jd)] = ac - bd \quad (4.3.4)$$

$$\Im[(a + jb)(c + jd)] = (a + b)(c + d) - ac - bd \quad (4.3.5)$$

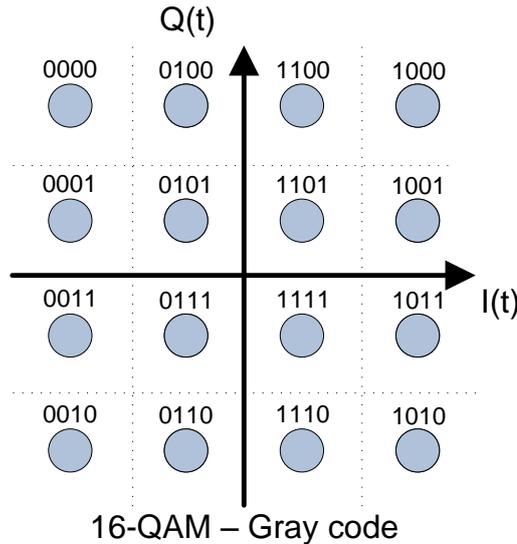


Figure 4.2.3: 16 QAM constellation map with gray mapped symbols.

Implementing complex multiplication using three multiplications instead of four allows for more taps with the same number of available multipliers. The FPGA model XC7VX1140T (Table 4.3.1 on the preceding page) has 5280 DSP48 slices. Using 3 multipliers per complex multiplication, 2 polarizations, 2 FIR filters per polarization and 12 parallel calculations gives a maximum of:  $5280/(3 \cdot 2 \cdot 2 \cdot 12) \approx 36$  taps.

Using as few filter taps as possible in the hardware implementation is desirable since cheaper FPGA:s with less DSP48 slices can be used. The trade-off is that with fewer filter taps, the BER increases (section 4.4.1).

## 4.4 Simulations

### 4.4.1 Filter Taps

Figure 4.4.1 shows a simulation performed on the original algorithm implemented in MATLAB where BER versus number of taps in the FIR filter is plotted. Each data point represents the average BER of the two polarizations at corresponding number of filter taps. The data used is 500k symbols from a 824 km fiber transmission.

Figure 4.4.2 shows how different smoothing factors effects the BER value. A smoothing factor of  $N=40$  for the phase compensation loop was used in the following simulations.

Figure 4.4.1 shows that there are diminishing returns when increasing the number of taps while keeping all other variables constant. The number of taps chosen for the hardware implementation should allow for a high enough BER value but also take into account the number of available DSP slices on the FPGA, as well as the power dissipation due to the large number of multipliers and logic.

Designing with some margin, the FPGA with the second most multipliers, XC7VX850T (3960 DSP48 slices) is chosen with an estimated achievable clock frequency of 500 MHz. Using the same approach as in section 4.3.1, the number of parallel calculations needed to reach a throughput of 112 Gb/s at 500 MHz is 14. With these constraints, a reasonable number of filter taps is 21, resulting in  $3 \cdot 2 \cdot 2 \cdot 14 \cdot 21 = 3528$  multipliers. Hence for the simulations in the following sections, the number of filter taps used will be 21.

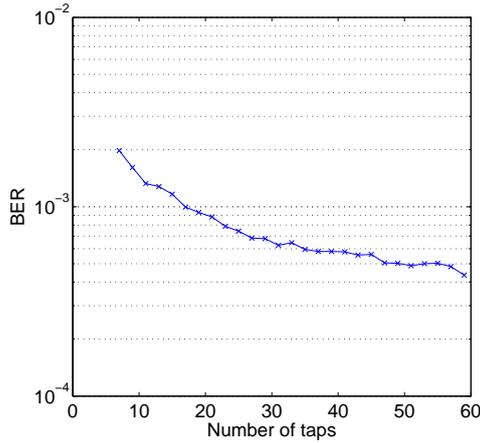


Figure 4.4.1: Effect of varying the number of filter taps.

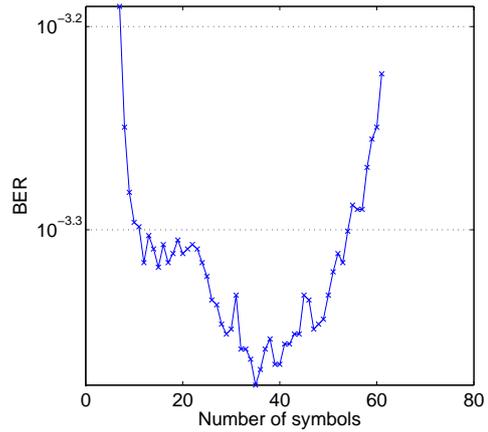


Figure 4.4.2: BER versus number of symbols used for smoothing.

#### 4.4.2 Coefficient Delay

A delay in the update of the filter coefficients is introduced in the hardware implementation. This delay is the result of calculating several symbols in parallel as well as the number of clock cycles it takes to calculate a new set of filter coefficients (pipelining). The effective symbol delay is the number of parallel calculations performed each clock cycle multiplied by the number of cycles it takes to calculate the new set of filter coefficients. Total delay in symbols can be expressed as:  $p \cdot n$ , where  $p$  is the number of parallel calculations each clock cycle and  $n$  is the number of cycles needed for the calculations. The effect of the coefficient delay on the BER value is shown in figure 4.4.3. The number of parallel calculations was set to 14 and the number of filter taps to 21.

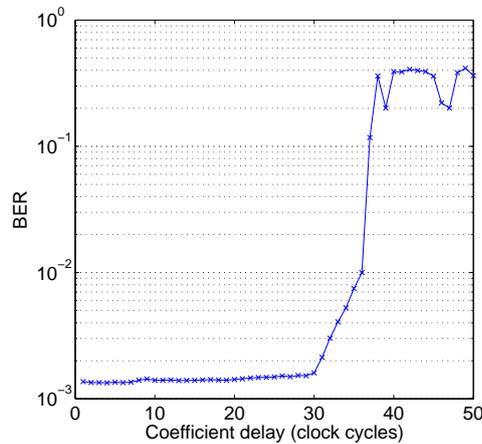


Figure 4.4.3: Effect of delaying the update of filter coefficients.

#### 4.4.3 Phase Feedback Delay

As with the feedback for updating the filter coefficients, a delay will be introduced in the phase correction loop. This is again the result of the parallelization of the system as well as the number of clock cycles it takes to calculate and compensate the phase error  $\phi^k$ . The effect of phase feedback delay is shown in figure 4.4.4 on the following page.

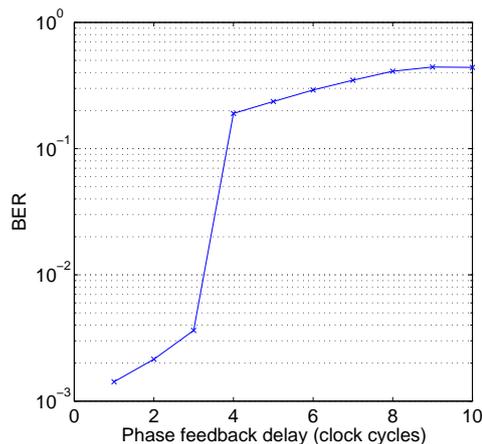


Figure 4.4.4: Effect of delaying the phase feedback loop.

#### 4.4.4 Fixed Point Representation

In a hardware implementation, the aspect of how numbers are represented needs to be considered. In MATLAB by default, all numbers are represented in 64 bit double precision floating point format. In this format 52 bits are used to store the fraction, 11 bits for the exponent and 1 bit for the sign of the number. It is possible to perform floating point operations in the FPGA. The trade-off is that the number of cycles required will increase, as well as the amount of logic needed for the implementation. In terms of hardware efficiency, using a fixed point representation throughout the system is preferable. However reducing the number of bits and removing the exponent will reduce the precision as well as the dynamic range in the system.

To study the effect of fixed point representation, and verify that the system would maintain its functionality, a MATLAB model using the Fixed Point Toolbox was developed. Using this toolbox, a fixed point representation of numbers can be used throughout the system and it allows for bitwise manipulation. Bit widths were initially set to maintain good precision while taking constraints of the FPGA architecture into consideration, e.g. longest bit width for the input to the DSP48 slices can be 25x18 for the two operands [18].

Simulations of the system using fixed point number representation were performed. The delays in the feedback loops were set to 20 cycles for the coefficients and 3 cycles for the phase feedback. The result is shown in table 4.4.1 and 4.4.2.

|     | Fixed point | Floating point |
|-----|-------------|----------------|
| BER | 0.0032      | 0.0028         |

Table 4.4.1: Difference in BER using fixed and floating point.

|                | FIR Input | FIR output | FIR Coefficients | Phase loop | LMS |
|----------------|-----------|------------|------------------|------------|-----|
| Number of bits | 11        | 19         | 8                | 8          | 2   |

Table 4.4.2: Number of bits used in different parts of the system.

## 4.5 Algorithm Improvements

In sections 4.4.2 and 4.4.3 the effect of phase feedback delay was observed to have the largest impact on system performance. Figure 4.4.4 shows that, when calculating 14

parallel symbols, the system can tolerate approximately 3 clock cycles of delay before the BER starts to increase drastically.

These 3 clock cycles limits the complexity and the amount of calculations that can be performed. In the phase feedback loop, several complex expressions need to be calculated before the result is fed back in the system. The phase error  $\phi^k$  (eq. 4.2.3) needs to be calculated, as well as the complex exponential  $e^{-j\phi^k}$ . This requires several steps of calculations, including complex division, summation of  $N$  symbols and a complex exponential function. Performing all these calculations in 3 clock cycles is not feasible with the FPGA technology available.

### 4.5.1 Effect of Phase Delay

Figure 4.5.1 shows the effect of delaying the phase feedback loop. Note that the constellation gets increasingly rotated as the feedback delay is increased. This rotation causes the constellation to be misaligned with the 16-QAM decision grid and maps samples to the wrong symbol increasing BER. This system feeds back the error in the signal to compensate in coming cycles. If the sample is mapped to the wrong symbol it will estimate the error incorrectly and further increase the error. When the rotation is too large the phase feedback loses track of the constellation and the system becomes unstable.

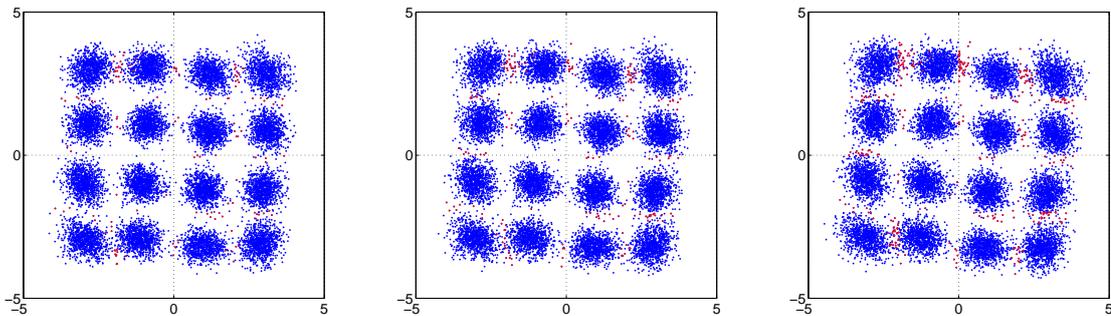


Figure 4.5.1: Effect of delaying the phase feedback loop. Left to right: 1, 2, 3 cycles delay.

### 4.5.2 Rotation Compensation

In order to mitigate the problem of the constellation getting rotated, the method Phase Rotation ESTimation (PREST) was developed to estimate the rotation and compensate for it. PREST uses a counter which sums all symbols that gets mapped to specific pre-defined areas of the constellation. The concept is shown in figure 4.5.2. Symbols that get mapped to a green area correspond to a clock-wise rotation of the constellation, whereas symbols mapped to a red area correspond to a counter clock-wise rotation.

When a symbol is mapped to either a green or a red area the counter will be decremented or incremented respectively, storing the difference. The value of the counter  $c$  is then divided by a constant  $a$  and subtracted from the phase error  $\phi^k$  as:

$$\phi_{m,new}^k = \phi_m^k - \frac{c}{a} \quad (4.5.1)$$

The ideal value of  $a$  is when the ratio  $\frac{c}{a}$  corresponds to the rotation of the constellation for symbol  $k$ . The constant  $a$  is decided empirically but is dependent on how the limits of the red and green areas are defined and latency of the feedback. The rotation tracking was refined to include areas for each point in the constellation instead of only considering the edges. This was inspired by the Phase Lock Loop (PLL) used in earlier radio systems [20].

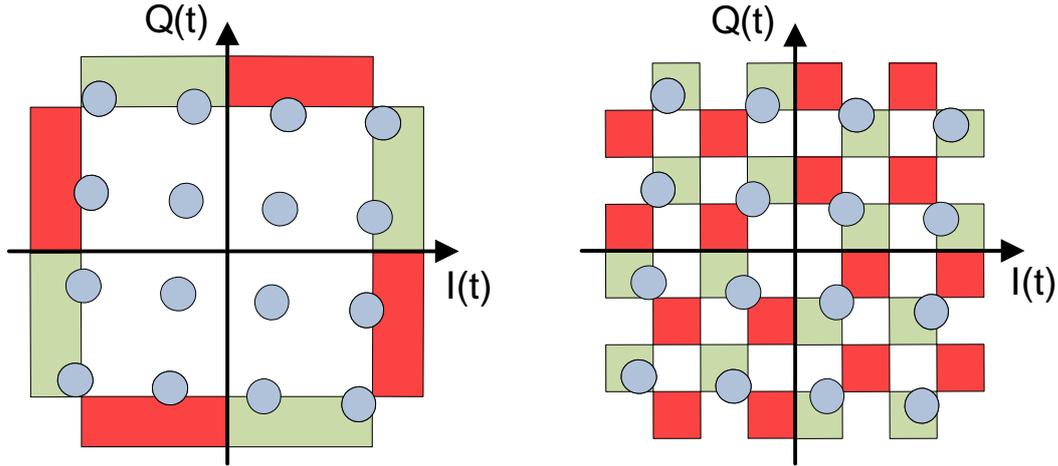


Figure 4.5.2: PREST1 and PREST2 mapping

### 4.5.3 Simulation Results Using PREST

Figure 4.5.3 shows the result on the BER when using the counter. The counter is delayed as long as the phase before it is added to  $\phi^k$  according to eq. (4.5.1). The constant  $a$  was set to 512 for PREST1 and 1024 for PREST2.

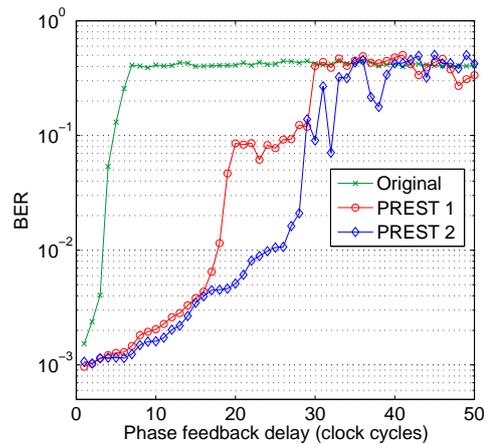


Figure 4.5.3: Comparison of unmodified and PREST versus phase delay.

Comparing the original unaided signal with the PREST aided, it can be seen that the strict requirements on the phase feedback loop can be relaxed. When the rotation tracking counter is not used the maximum possible delay in the phase feedback loop was 3 clock cycles. Using the rotation tracking counter allows for having 16 clock cycles of delay for the same BER value.

### 4.5.4 Final Simulations

A simulation using the fixed point representation of the system and the counting method was done. The bitlengths are the same as in table 4.4.2. The result is shown in table 4.5.1. Note that a smoothing factor of 28 was used in this simulation; it is hardware efficient to have smoothing factors that are a multiple of the parallelization. In earlier simulations,

a smoothing factor of 40 was used, but with longer phase feedback delays and the use of PREST a lower smoothing factor gives better results.

|                      |        |
|----------------------|--------|
| Taps                 | 21     |
| Parallel             | 14     |
| Coefficient Delay    | 20     |
| Phase Feedback Delay | 12     |
| Counter Delay        | 12     |
| Smoothing Factor     | 28     |
| average BER          | 0.0022 |

Table 4.5.1: Simulation using fixed point and counter.

Due to available computer limitations, batches of 500k samples from the 5M sample files were simulated. The resulting BER were 0.0021, 0.0022, 0.0025 and 0.0021. This simulation also includes the changes made for hardware efficiency such as replacement of divisions and argument.

# Chapter 5

## Hardware

### 5.1 FIR with LMS Updating

Parallelization of FIR filters is simple but requires large area. Identical FIR filters are created and the input data to them are shifted two samples to calculate the next value. The area required and power consumption grows roughly linearly with the number of parallel stages. This and the usage of a butterfly structure and complex numbers sets a limitation on how many taps the filter can have, see section 4.4.1.

Complex multipliers are created with the Xilinx CORE-Generator and “Complex multiplication 4.0” [21], this allows the compiler to effectively map multiplications to DSP48 slices [19] and effectively map surrounding logic. To reduce the number of calculations made, the downsampling of two is integrated into the filter by calculating every other sample. After the multiplications, the sum of all outputs is calculated with a tree structure of adders and the output from the butterfly structure is added.

The FIR filters are updated with an LMS algorithm according to equations 4.2.1 and 4.2.2. To save area and power all of the parallel filters are updated with the same coefficients. Since this update works slowly over a long time period this has a small effect on the signal. If no changes were made to this algorithm it would be larger than the FIR filters. By setting  $\mu$  to a power of two this becomes a fixed shift, which is hardware efficient. The other variables used only need to be a few bits in order to update the filters as it is averaged over long runs.

### 5.2 Phase Recovery

The phase feedback is the part of the system that has been modified the most in order to be implemented effectively in hardware. First an estimation of the phase error ( $err_k$ ) is taken by dividing the input data  $y^{k-i}$  with the decided one  $d^{k-i}$ . Here it is possible to take advantage of knowing the different values that the constellation is mapped to, and store the inverse of these, turning it into a complex multiplication instead.

$$err_k = e^{j \cdot \arg(\sum y^{k-i}/d^{k-i})} \quad (5.2.1)$$

The floating average is calculated by summing the incoming values and the  $N$  previous sums. To calculate the angle (argument) the two considered methods were COordinate Rotation DIGital Computer (CORDIC) and LUT. CORDIC is an iterative process which can be used to approximate trigonometric functions [22]. In the phase feedback loop latency is the largest issue, hence using a LUT was chosen instead. LUTs double in size per extra bit, and since the input is complex it will become a two dimensional LUT. The sign is removed from both I and Q which saves two bits in the LUT, these are restored

after the exponential. This operation saves a lot of area but also increases latency, so a small amount of latency can be saved by using the signed value throughout the loop.

The rotation tracking counter sums hits for clockwise and counter-clockwise rotation and stores the difference, this is then divided by a scaling factor and added to the calculated angle ( $\phi^k$ ). All scaling factors should be a power of two for ease of implementation as this can be replaced by a simple shift operation.

Complex exponentials can be calculated in the same manner as the angle but also with a power series. This is a good method for some cases but it is not suited for this FPGA implementation since it requires taking the  $N$ :th power of the input and dividing by  $N!$  where  $N$  depends on the resolution needed. With the need for low latency a LUT is used for the exponential as well.

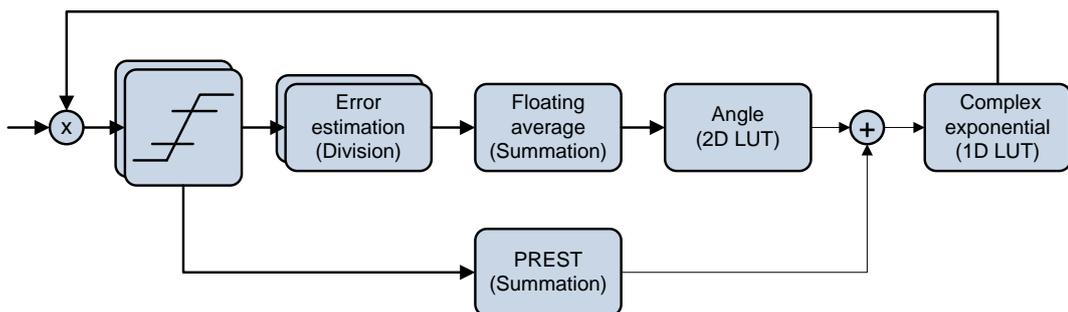


Figure 5.2.1: Hardware view of phase feedback

### 5.3 Area and Power

Due to lack of computer capacity only a single parallelization of the FIR filters and LMS update have been simulated. The estimation of total power is done by removing leakage and input/output port power and then multiplying by 14 for parallelization. The FPGA seems to have a constant 1-2 W leakage which grows slowly with logic, but only a fraction of the active power. The phase feedback loop is a much smaller design and has been simulated fully in parallel.

Table 5.3.1 gives an overview of the key parameters available in the target FPGA and the required resources for the main blocks. Due to lack of computational capacity only a single parallel line of the FIR filters and LMS update has been simulated. These were multiplied by the number of parallel lines in the design (14), to get an estimation for the entire design. The phase correction loop was small enough to simulate in full parallelization.

As the entire system could not be simulated only an estimation of the total area is given, there is some need for extra control logic, but this is considered to be a small part of the total design.

The first power simulation approach of the FIR filter used a 50% toggle rate at all nodes, including the DSP48 slices. This returned an active power of 4 W per parallel or 56 W for the whole system. This simulation does not take the structure or input data into consideration and is a high estimate. As data passes through logic the switching activity generally decreases and thus lowers power; this also applies to the DSP48 slices which were set to 50% toggle rate.

To get more accurate power values a Switching Activity Interchange Format (SAIF) file was generated for the FIR filter design. This method uses input data which it propagates

| Design part      | DSP48s     | Logic slices |
|------------------|------------|--------------|
| XC7VX850T        | 3960       | 133.5k       |
| FIR Filters      | 3528       | 37.4k        |
| Phase correction | 128        | 2.2k         |
| LMS update       | 0          | 49.5k        |
| Total            | 3656 (92%) | 89.1k (67%)  |

Table 5.3.1: Area estimation of the equalizer

through the system, storing the switching statistics at each node. For this test the input data used was 200 samples from the MATLAB simulation. Using real test data and propagating it through the system reduced the power estimation to roughly 1/4 of the simulation with 50% toggle rate.

| Design part                 | 1 parallel (W) | 14 parallel (W) |
|-----------------------------|----------------|-----------------|
| 21 tap butterfly FIR (SAIF) | 1.05           | 14.7            |
| LMS update                  | 1.68           | 23.5            |
| Phase recovery              | -              | 1.52            |
| Equalizer                   | -              | 39.7            |

Table 5.3.2: Active power estimation of the equalizer

The simulations for LMS and phase recovery were made with the 50% toggle rate estimation. These are likely to follow in the same trend as the FIR filters if a SAIF power test is done. Especially the LMS update can be refined for power efficiency by doing fewer updates to the coefficients and a more careful design, leading to an active power lower than the FIR filters.

## Chapter 6

# Discussion

Working in MATLAB has allowed for fast development and simple testing. If the development would have started earlier in VHDL a solution to the problem might not have been reached. Starting with simulations of the system in MATLAB, and changing it gradually to a more hardware accurate representation has proven to be a useful method. This since latency caused by the feedback loops causes the system to become unstable and developing a solution in MATLAB is much faster.

The simulation results show that the phase feedback loop is very sensitive to delay, making it a bottleneck of the system. Using the algorithm described in [9], the maximum allowed delay for an FPGA clocked at 500 MHz would be approximately 3 clock cycles (figure 4.4.4). This would not be implementable in an FPGA. This is due to the calculations needed in the feedback loop, which would not be allowed to take more than 3 cycles to complete.

PREST was devised to correct for the rotation of the constellation, and it reduced the sensitivity of the phase feedback loop significantly. As can be seen by comparing the methods in figure 4.5.3, the delay can be increased from 3 to 16 cycles while maintaining the same BER value. With an allowed delay of over 12 clock cycles it is possible to perform the calculations needed on an FPGA. This new method has only been tested in MATLAB so far. The possibility of implementing the system hence lies in the accuracy of the MATLAB model.

There is a possibility that the PREST2 rotation update with different weights and more zones where it counts values can be used as a replacement for the phase feedback loop entirely. This can simplify the feedback loop and reduce both latency and power. Even without a switch to PREST, using different weights in the smoothing on the feedback might prove very beneficial.

Table 5.3.2 shows that the LMS update requires more power than the FIR filters. A SAIF simulation of this needs to be done as the toggle rate should be relatively low. It can also be optimized for a smaller design by updating the filter coefficients at a lower rate. With this the LMS update should take much less power than the FIR filters.

In MATLAB simulations the LMS update has been tested with promising results of only using the sign of  $e$ ,  $d$  and  $y$  in equation 4.2.2, instead of a few bits of resolution. This would be extremely efficient to implement in hardware but simulations over a longer period might be needed to confirm this functionality.

The number of multipliers available in the FPGA is a limiting factor, since it limits the number of filter taps that can be used in the FIR filters. However the most advanced FPGAs of today have enough multipliers to allow for approximately 35 filter taps, using one FPGA per frequency band.

Different system setups can be considered. The focus has been on a system using two FPGAs, one for each frequency band. However the system could be split further, since

the two polarizations in the system can be split apart as well. This would allow twice the number of filter taps but doubling the number of FPGAs used from two to four. This would improve the BER value; however it will be more expensive since it requires twice the number of FPGAs. It is also possible to use one FPGA for the entire equalizer but this will reduce the number of taps.

Comparison with ASIC technology could also be made once the system is completely described in VHDL. The most interesting comparisons will be the difference in power consumption and latency in the feedback loops. The architecture of a Virtex7 is not as effective, but it is constructed with 28 nm technology and has dedicated hardware multipliers. It is not feasible to implement the system in this technology with an ASIC however; it is too expensive to develop and manufacture. Making a comparison between 28 nm FPGA and 65-90 nm ASIC would be interesting.

This thesis only considers the equalizer part of the DSP system, but as can be seen in figure 2.1.3 it is only one part. Similar work is needed to evaluate how the rest of the system would be affected by being implemented on an FPGA. At the start of the thesis the equalizer was evaluated to be the most crucial to evaluate, but the rest of the system will affect the end result as well.

There are no feedback loops in the rest of the system which is a large advantage when designing. There are however other complications such as frequency domain filtering and complex MATLAB functions. When these functions are converted to hardware they will introduce additional noise in the system which will affect the equalizer, hence it is only when the entire system is simulated that a final conclusion can be drawn.

## Chapter 7

# Conclusion

To implement the equalizer with FPGA technology should be possible. Simulations in MATLAB indicate that a setup with 14 parallel stages and 21 taps in each FIR filters give BER of  $2.2 \cdot 10^{-3}$ . This is accounting for a delay in the phase recovery of 12 cycles and 20 cycles for the filter coefficients. The MATLAB library “Fixed point” is also used to represent different bitlengths in the system, further complex equations have been modified to give the same result as in the hardware.

Using the Virtex7 850T, 21 taps can be implemented using only DSP48 slices as multipliers to save energy and hardware resources. The phase recovery loop has a latency of 12 cycles and can be optimized slightly to reduce this. In total the system uses 92% of DSP48 slices and 67% of the logic.

Power simulations have been performed and an estimate of the active power is 39.7 W. The active power of the FIR filters is 14.7 W and the LMS update 23.5 W. The LMS update code has not been simulated using SAIF, this will reduce the power estimation as the toggle rate will be lower. The VHDL code can be optimized as very few bits of resolution are used, but there was not enough time to implement this in the thesis.

It is thanks to the developed rotation tracking counter that the equalizer becomes implementable. Without it only 3 cycles of phase feedback delay would be possible. Even if this limit can be pushed to the double (6) cycles the FPGA can not calculate the feedback in this time. By using PREST it is possible to have a latency of 12 cycles for the phase feedback and still maintain a stable system and stay below  $3.0 \cdot 10^{-3}$  in BER.

# Chapter 8

## Future Work

### 8.1 Optimizations

#### 8.1.1 FIR Filter

The FIR filters take most of the area and power in the system hence it is here most optimizations can be done for area and power. By using symmetric filter coefficients it is possible to add together the inputs to the filter taps and save multipliers. In a symmetric filter there are two taps that has the same value (except for the odd tap in the middle). This can be exploited by adding the inputs to taps with the same coefficient, saving roughly half of the multiplications needed. It is not always possible to use symmetric filter coefficients, but if it is, a large amount of power and area can be saved in the design, ultimately allowing for a greater number of taps in total.

Another design that has been discussed is using a large static FIR filter without the butterfly structure to compensate for the CD since that is constant, and then use a smaller LMS updated butterfly FIR to handle PMD. Static filters can be designed with much higher efficiency than adaptive filters. This since one of the multiplication operands is known and can be hardcoded into the design. It can be further improved by slight modifications to the filter coefficients. For example multiplying with 47 ( $101111_2$ ) requires 5 adders but by changing this to 48 ( $110000_2$ ), only two adders are needed. This does change the coefficients and the result of that needs to be simulated.

As the filter coefficients towards the edges are of lower amplitude than the centre it could be possible to implement those taps in logic. The DSP48 slices used for multiplication are fixed to  $18 \cdot 25$  bits and only a few bits are needed for the coefficients in the outermost taps. This will free up some DSP48 slices and might allow for more filter taps and can probably save power.

#### 8.1.2 Feedback Loops

The phase feedback loop only requires a fraction of the area and power of the FIR filters so optimizing that will not have as large of an effect on total power. However reducing latency in the feedback loops, especially for phase, will lower the BER. The LMS feedback loop is already relatively fast, and does not change BER much so the focus should be on the phase correction.

Some optimizations have already been done to the phase feedback loop, for example by using LUTs to calculate complex functions. At these LUTs the sign bit is removed to reduce the 2D LUT to one fourth of its size, this saves area but gives some extra latency so this can be removed at the cost of larger LUTs. When the bitlength in the feedback loop has been fully decided it should be possible to reduce latency by fully utilizing each

cycle of calculation by moving the flipflops. The adder tree which sums together the values after the divider can probably be optimized to save latency as well.

A large part of the latency in the feedback comes from the two complex multiplications that are needed. The first is before the Decision block and the second is used in the pseudo-division. These take a total of 6 clock cycles, and reducing this can be very beneficial.

## 8.2 Compatibility with 400G-1T

An important aspect is how well this design will translate to future generations of optical transmission. This section will however only consider the equalizer part of the DSP in the receiver as this is in the scope of the project. As previously discussed the two main areas of interest are the total area and power of the FIR filters and the latency of the phase feedback loop.

There are many possible ways to increase transmission rates through the system, increase the baud rate, number of frequency bands and bits/sample. Changing the number of frequency bands will scale linearly with the number of FPGAs required and will not change the design of the equalizer. This is a straightforward change and is likely to be used in future generations [23].

It is also possible to change which modulation scheme that is used, in this thesis 16-QAM is used and codes for 4 bits/sample. By changing to 64-QAM, 6 bits/sample, throughput is increased by 50% but the symbol density in the constellation is increased fourfold. To be able to achieve the same BER, the signal and processing quality needs to be greatly increased. Another possibility is to go down to 8-Phase Shift Keying (PSK), 3 bits/sample, this does lower throughput by 25% but has the advantage of not being as sensitive to noise as the 16-QAM and may allow for higher baud rates.

Another possibility is to increase the overall baud rate of the system, this will scale linearly with the clock frequency of the FPGA or with the level of parallelization in the equalizer. The clock frequency of the FPGA is not likely to increase significantly in the coming generations so a change in baud rate will translate to a linear increase in FPGA size. This increase in parallelization will also lead to a linear increase in latency for the feedback loops. As the phase correction loop is especially sensitive to higher latency this will need to be addressed.

# Bibliography

- [1] Celtic-Plus, “100GET,” <http://www.celtic-initiative.org/Projects/Celtic-projects/Call4/100GET/Project-default.asp>, May 2011.
- [2] A. Rhodin and J. Mårtensson, “SCM System Analysis: Part 6 - DSP sub-system,” 100GET-ER (SE), Tech. Rep., 2010.
- [3] T. Pfau, S. Hoffmann, and R. Noe, “Hardware-Efficient Coherent Digital Receiver Concept With Feedforward Carrier Recovery for M-QAM Constellations,” *J. Lightwave Technology*, vol. 27, no. 8, pp. 989–999, April 2009.
- [4] A. Lucent, “Welcome to the 100g era,” <http://www.alcatel-lucent.com/features/100g-era/>, Tech. Rep., June 2011.
- [5] B.-E. Olsson, C. Larsson, and A. Alping, “RF-assisted Optical Transmission Systems,” Ericsson Research - Mölndal, Tech. Rep., 2011.
- [6] National Instruments, “What is I/Q data?” <http://zone.ni.com/devzone/cda/tut/p/id/4805>, February 2011.
- [7] B.-E. Olsson, “SCM - DSP Introduction,” Ericsson Research - Mölndal, Tech. Rep.
- [8] S. J. Savory, “Digital Filters for Coherent Optical Receivers,” *Opt. Express*, vol. 16, no. 2, pp. 804–817, January 2008.
- [9] J. Mårtensson, “SPEED Project D1.5 - Report on DSP implementation,” Acreo, Tech. Rep., 2010.
- [10] S. J. Savory, G. Gavioli, R. I. Killey, and P. Bayvel, “Electronic Compensation of Chromatic Dispersion Using a Digital Coherent Receiver,” *Opt. Express*, vol. 15, no. 5, pp. 2120–2126, March 2007.
- [11] C. R. S. Fludger, T. Duthel, D. van den Borne, C. Schullien, E.-D. Schmidt, T. Wuth, J. Geyer, E. DeMan, G.-D. Khoe, and H. de Waardt, “Coherent Equalization and POLMUX-RZ-DQPSK for Robust 100-GE Transmission,” *J. Lightwave Technology*, vol. 26, no. 1, pp. 64–72, January 2008.
- [12] P. Winzer and A. Gnauck, “112-Gb/s Polarization-Multiplexed 16-QAM on a 25-GHz WDM Grid,” in *34th European Conf. Optical Communication*, 2008.
- [13] I. Fatadin, D. Ives, and S. J. Savory, “Blind Equalization and Carrier Phase Recovery in a 16-QAM Optical Coherent System,” *J. Lightwave Technology*, vol. 27, no. 15, pp. 3042–3049, August 2009.
- [14] M. Scholten, T. Coe, J. Dillard, and F. Chang, “Enhanced FEC for 40G / 100G,” Vitesse, Tech. Rep., 2009.

- [15] M. Hutton, "Field Programmable Gate Arrays: a 2011 Overview," Ericsson research - Mölndal, Tech. Rep., 2011.
- [16] *System Generator for DSP*, Xilinx, <http://www.xilinx.com/tools/sysgen.htm>.
- [17] M. Möller, "Implementation of a 16-QAM Receiver in a FPGA," Ericsson AB - Microwave and High Speed Electronics Research Center, Tech. Rep., 2010.
- [18] *7 Series FPGAs Overview*, Xilinx, [http://www.xilinx.com/support/documentation/data\\_sheets/ds180\\_7Series\\_Overview.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf).
- [19] *7 Series DSP48E1 Slice*, Xilinx, [http://www.xilinx.com/support/documentation/user\\_guides/ug479\\_7Series\\_DSP48E1.pdf](http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf).
- [20] A. Mouaki Benani and F. Gagnon, "Comparison of Carrier Recovery Techniques in M-QAM Digital Communication Systems," in *Canadian Conf. Electrical and Computer Engineering*, 2000, pp. 73–77, vol. 1.
- [21] *LogiCORE IP Complex Multiplier v4.0*, Xilinx, [http://www.xilinx.com/support/documentation/ip\\_documentation/ds793\\_cmpy.pdf](http://www.xilinx.com/support/documentation/ip_documentation/ds793_cmpy.pdf).
- [22] R. Andraka, "A Survey of CORDIC Algorithms for FPGA Based Computers," in *ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 1998, pp. 191–200.
- [23] S. Chandrasekhar and X. Liu, "Enabling Components for Future High-Speed Coherent Communication Systems," in *Optical Fiber Communication Conf.*, 2011.