High speed FPGA based scalable parallel demodulator design

Master's Thesis
by

H.M. (Mark) Beekhof

Committee:
prof.dr.ir. M.J.G. Bekooij (CAES)
dr.ir. A.B.J. Kokkeler (CAES)
ir. J. Scholten (PS)
G. Kuiper, M.Sc (CAES)

University of Twente, Enschede, The Netherlands
April 10, 2017

Abstract

Nowadays applications have to process data at high data rates. These data rates are increasing faster than the frequencies on which Field Programmable Gate Arrays (FPGAs) operates. In this thesis a parallel design is presented so that the FPGA still can be useful to process data at high rates.

At the CAES group a ML605 FPGA evaluation board is available which is interfaced with an Analog to Digital Converter (ADC). On this FPGA board a multiprocessor system is installed named Starburst. It is possible to create hardware accelerators and integrate them into this system. An Universal Software Radio Peripheral (USRP) box is available, combined with the GNURadio software it is possible to create a software defined radio.

First a reference conventional demodulator is created which processes the samples in sequential order. The performance of this demodulator was tested using GNURadio. The performance was tested by detecting packages at the receiver side. The amount of packages combined with a certain level of noise that was added resulted in a Packet Error Rate (PER) for different Signal to Noise Ratios (SNRs).

Thereafter, the design was implemented as a hardware accelerator on the FPGA. The performance of this implementation was compared to the one created using GNURadio. The performance, measured with the PER for different SNRs, of the implementation on the FPGA was comparable with the one created in software for low SNRs. For high SNRs the implementation on the FPGA has a certain floor in the PER.

After the reference conventional design was implemented a parallel design has been created. The conventional design was a basis for this design. The performance of this design was compared to the conventional one. The performance was not as good as the conventional design. The design was less robust to a timing difference between the clocks of the transmitter and the receiver. However, the reasons for this are explained in this report together with possible solution directions. Due to time constraints it was not possible to create an implementation that addresses the discussed issues. However it is expected that it is possible to create a parallel structure with an equal PER assuming no clock difference. However the design will be less robust to a clock difference between transmitter and receiver. A disadvantage of the design is that it will take up a lot of resources on the FPGA which will limit the amount of parallel paths that can be used. For the presented design, the maximum amount of parallel paths will be around 16, which is enough for the 5 GS/s ADC that is available.

As future work it would be interesting to implement the design in combination with a high speed ADC that delivers samples in parallel. Further research is required to improve the design so that it can be used in applications with other data rates.

Contents

Abstract **i**

1 Introduction **1**
1.1 Context . . . 2
1.2 Problem Description . . . 2
1.3 Related Work . . . 3
1.4 Outline . . . 4

2 BPSK Basics **5**
2.1 Phase Shift Keying . . . 6
2.1.1 BPSK . . . 6
2.1.2 Non-Coherent versus Coherent . . . 6
2.1.3 Performance . . . 7
2.1.4 Implementation . . . 8
2.2 Modulation . . . 8
2.3 Demodulation . . . 10
2.4 Symbol Time Recovery . . . 13
2.4.1 Early Late . . . 13
2.4.2 Gardner . . . 14
2.4.3 Mueller Muller . . . 14
2.5 Summary . . . 15

3 Conventional DBPSK Demodulator **16**
3.1 Design . . . 17
3.1.1 Demodulation . . . 17
3.1.2 Sampling . . . 18
3.1.3 Top Level Design . . . 23
3.2 Implementation . . . 25
3.2.1 GNURadio . . . 25
3.2.2 FPGA . . . 26
3.3 Results . . . 29
3.3.1 Set-up . . . 29
3.3.2 PER . . . 29
3.3.3 Measurements . . . 30

3.4 Summary 36

4 Parallel DBPSK Demodulator **37**
4.1 Design 38
4.1.1 Switch 38
4.1.2 Demodulation 40
4.1.3 Symbol Time Recovery 41
4.2 Implementation 42
4.2.1 Switch 43
4.2.2 Demodulation 44
4.2.3 Top Level Implementation 44
4.3 Conclusion 45

5 Results and Comparison **47**
5.1 Results Parallel Receiver 48
5.2 Scalability 50
5.3 Alternative Parallel Structure 53
5.4 Summary 53

6 Conclusion and Recommendations **55**
6.1 Conclusion 56
6.2 Recommendations 57

List of Figures **59**

Bibliography **63**

Acronyms

ADC Analog to Digital Converter

BER Bit Error Rate

BPSK Binary Phase Shift Keying

DBPSK Differential Binary Phase Shift Keying

FIR Finite Impulse Response

FPGA Field Programmable Gate Array

I In-Phase

LUT Lookup Table

PER Packet Error Rate

PSK Phase Shift Keying

Q Quadrature

QAM Quadrature Amplitude Modulation

QBL-MSK Quasi-Bandlimited Minimum Shift Keying

QPSK Quadrature Phase Shift Keying

RAM Random Access Memory

ROM Read-only Memory

RF Radio Frequency

SNR Signal to Noise Ratio

SPS Samples per Symbol

TCXO Temperature Compensated Crystal Oscillator

USRP Universal Software Radio Peripheral

XOR Exclusive Or

CHAPTER 1

Introduction

1.1 Context

The amount of data generated by applications is increasing. At NXP they are researching the use of polymer waveguides in cars. These polymer waveguides have several advantages over the cables that are used at the moment. One of the advantages is that it is possible to use higher data rates. These data rates are higher than the clock frequency of currently available FPGAs. However FPGAs can still be useful to process data. By processing data in parallel it is possible to process data streams with a high data rate.

1.2 Problem Description

A high speed ADC is available which can read samples up to a rate of 5 GS/s. The maximum clock frequencies of FPGAs is currently much lower, therefore samples should be processed in parallel. This ADC is not interfaced yet with the multiprocessor system called Starburst, which is installed on the available Virtex 6 FPGA board. Because it will cost a lot of time to interface the ADC with the platform, a proof of concept will be created with an available set-up. The set-up consists of a narrow band Radio Frequency (RF) receiver frontend that has been interfaced with a Virtex 6 FPGA board such that software defined radio receiver applications can be prototyped on an embedded multiprocessor system. This set-up can be used to implement a reference conventional demodulator and a parallel demodulator as a proof of concept.

The objective of this graduation project is the creation of a demodulator on a FPGA which can process data at a rate of 5 GS/s where data is processed in parallel. This demodulator is implemented on a Virtex 6 FPGA which has been interfaced with a RF receiver frontend. The demodulator should run on a lower frequency than the frequency at which the samples arrive. The implementation should be made such that it should be able to work with the high speed ADC. The implementation should be made scalable, so that it can be used at higher data rates. To keep the demodulator simple Phase Shift Keying (PSK) is used as modulation technique. Binary Phase Shift Keying (BPSK) is chosen because it is more suitable for the application in combination with polymer waveguides. The reason is that these waveguides have a relatively high damping. BPSK will have a better performance under low SNRs, from Bit Error Rate (BER) perspective, than higher order modulation schemes. The demodulator should be kept simple so that the focus is on creating a parallel hardware structure. Design options should be explored and the achieved performance of the system should be compared with theoretical results. This implementation should be scalable such that the same concept can be used to process samples of a high speed 5 GS/s ADC.

1.3 Related Work

There is some work done in the field of processing data in parallel in demodulators. In [13] a parallel demodulation structure is presented. This structure is based on a frequency domain implementation of a matched filter. Besides that a Symbol-Timing Recovery Loop is discussed that uses an adapted version of the Gardner algorithm. This adapted version is suitable for implementation on an FPGA. The structure is tested with an uncoded BPSK signal. Simulation results show that their presented design is performing as well as the serial design.

In [4] a parallel demodulator structure suitable for implementation on a FPGA for a high order Quadrature Amplitude Modulation (QAM) signal is presented. An architecture is presented for a 5 GS/s demodulator of a 64QAM signal. In this paper a symbol time recovery method was used that was presented in [7], which is not sensitive to SNR and carrier frequency offset. This timing recovery method is more complex but useful for QAM signals.

In [9] trade-offs for serial and parallel demodulation are discussed for Quasi-Bandlimited Minimum Shift Keying (QBL-MSK). The parallel implementation in this case consists of 2 parallel paths, a In-Phase (I) and Quadrature (Q) path. This parallel structure is not useful because it does not run on a lower clock frequency than the rate at which the samples arrive. Two synchronisation methods are used to for synchronisation. The first one is using average zero crossing and the other one uses the maximum eye opening for synchronisation. They conclude that zero crossing is providing either less or the same BER degradation as that obtained with maximum eye opening synchronisation. This is significant because zero crossing synchronisation is implemented much easier in hardware.

In [20] a frequency-domain parallel demodulation structure is discussed. In this paper an improved structure is discussed which should be more useful in high speed systems. The timing synchronisation is combined with the matched filter. Because only the best points at the output of the matched filter are used for timing synchronisation. Simulation results are given for a system with Quadrature Phase Shift Keying (QPSK) modulation. They conclude that the structure they presented is suitable for high-speed implementations.

All the related work above uses a frequency-domain parallel demodulation structure. There was no related work found which uses a time-domain parallel demodulation structure, except [9] where two time-domain parallel paths are used which do not run on a lower frequency. In the related work architectures and algorithms were discussed for symbol time recovery which can be useful for the demodulator that is designed during this thesis.

1.4 Outline

First in Chapter 2 the basics of BPSK are discussed, which is necessary to understand the working of the BPSK demodulator. After that a conventional Differential Binary Phase Shift Keying (DBPSK) demodulator design and implementation are discussed in Chapter 3. This conventional demodulator is created so that it is possible to compare the performance of the parallel implementation with a conventional implementation. The design of the conventional demodulator is used as a basis to design a parallel demodulation structure which is discussed in Chapter 4. In Chapter 5 the results of the measurements of both designs are compared with each other. Besides that there is a section about the scalability of the parallel design in this chapter. Chapter 6 will conclude this thesis with a conclusion and recommendations.

CHAPTER 2

BPSK Basics

In this chapter the modulation technique of BPSK will be discussed. After which other demodulation techniques are discussed. A demodulation technique will be chosen which is used in the rest of this thesis.

2.1 Phase Shift Keying

PSK is a digital modulation technique where the information is modulated by changing the phase of the carrier signal. Because it is a digital modulation technique the number of distinct phases that used is finite.

To demodulate a PSK signal the received signal can be compared to a reference signal, this method is called coherent demodulation. In case there is no reference signal used the method is called non-coherent demodulation. In the last case the received signal should be compared with itself. Differential modulation should be used to be able to demodulate a signal with a non-coherent technique.

2.1.1 BPSK

The simplest form of PSK uses only two distinct phases to convey the data. This method is called BPSK. Symbols are used to transmit data from transmitter to receiver. These symbols represent a certain bit or multiple bits. For BPSK each symbol represents one bit. A constellation diagram can be used to compare modulation techniques with each other. In Figure 2.1 the constellation diagram of BPSK is depicted. In a constellation diagram the signal is depicted in the complex plane at symbol sampling instants. The points in the diagram represents the possible symbols that can be transmitted, in this case two symbols are depicted with values 0 and 1. The points in the diagram are chosen arbitrary, they can be depicted anywhere on the unity circle as long as they are 180 degrees apart from each other. For non-differential modulation each symbol represents a fixed bit value, 1 or 0. This is not the case for differential modulation, where a symbol transition represents a fixed bit value. All modulation techniques use a defined period in which they send one symbol. This period is called the symbol period.

2.1.2 Non-Coherent versus Coherent

As mentioned earlier, a coherent or a non-coherent method can be used for demodulation. A choice has to be made which of the two is more suitable to use in the research presented in this thesis. There are basically two points worth considering for this decision namely: performance with respect to the BER and ease of creating a parallel implementation.

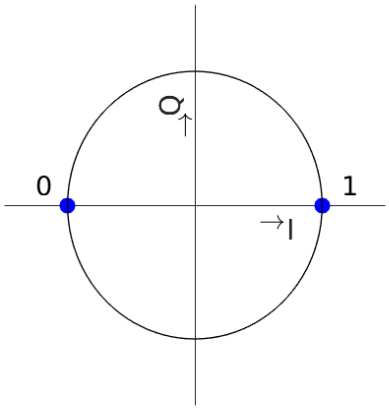

Figure 2.1: Constellation Diagram of BPSK

2.1.3 Performance

The performance can be measured by the BER compared to the SNR per bit. In [8] the theoretical relation between the BER and the SNR per bit are given for the case of non-differential modulation and differential modulation. These relations are depicted in Figure 2.2. It can be seen that the chance of an error for differential method is larger than for the coherent method for the same SNR per bit. That is due to the fact that each symbol period should be compared with the previous symbol period. That means that the chance of a symbol error depends on two symbol periods. The chance that there is an error in two symbol periods is higher than the chance of an error in a single symbol period.

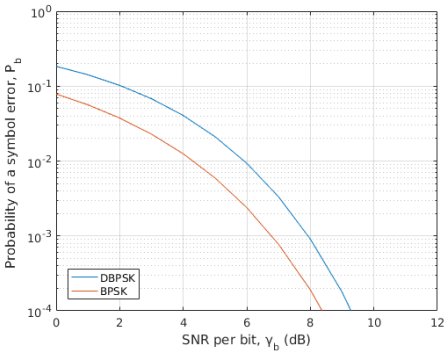

Figure 2.2: Probability of a symbol error for BPSK and DBPSK

2.1.4 Implementation

As discussed before for non-coherent BPSK no reference signal is required at the receiver because the info is encoded in changes of the phase. That means that no circuit is necessary to generate this reference signal. Therefore the implementation of non-coherent BPSK can be less complex than the implementation of a coherent demodulation technique. Assuming that creation of the reference signal is part of the demodulator, when that is the case the circuit required to create the reference signal should be made parallel too.

Because the focus in this thesis is on parallelism in demodulators the non-coherent method is used. The reason therefore is that there is no reference signal required at the receiver and therefore it is easier to create a parallel structure, because there is no circuit required for the creation of the reference signal. The degradation in BER is not problematic as long as the parallel demodulator is compared with a demodulator that uses the same technique. In future designs it is possible to add a circuit to the design that creates the reference signal.

2.2 Modulation

Equation 2.1 can be used to modulate a signal using BPSK. Where $x(n)$ is the data signal consisting of bits and k is the sensitivity given by Equation 2.2 where f_0 is the frequency used and f_s is the sampling frequency. $x(n)$ should be interpolated with the number of Samples per Symbol (SPS). SPS is the number of samples that is used to send one symbol it is equivalent to the symbol period.

$$y(m) = cos\Big(km + \pi \cdot x(n)\Big) \tag{2.1}$$

$$k = 2\pi\frac{f_0}{f_s} \tag{2.2}$$

To be able to demodulate without a reference signal the signal should be modulated differential. To make the signal differential each bit should be compared with the previous bit, when they are the same $x(n)$ should be 0 otherwise it should be 1. This can be accomplished by using an Exclusive Or (XOR) of the current bit with the previous bit.

In Figure 2.3 an example is depicted of a BPSK signal that is modulated differential. The upper plot shows the modulated signal and the lower one shows the bits that were modulated. In this case the symbol time is exactly one period of the carrier wave. It is best for the frequency response to use exact periods of the carrier wave for the symbol time, because the bandwidth needed to represent a signal is smaller in case the symbol transitions are at the zero crossing of the carrier signal.

In Figure 2.4 the frequency response of two possible BPSK signals are depicted. Each frequency response plot corresponds to the BPSK signal that is depicted below it. It can be seen that the frequency response is damping faster when the bit transition takes place when the carrier is at a zero crossing (right case). When instead the bit transition takes place at the maximum value of the carrier (left case) the frequency response is damping much slower.

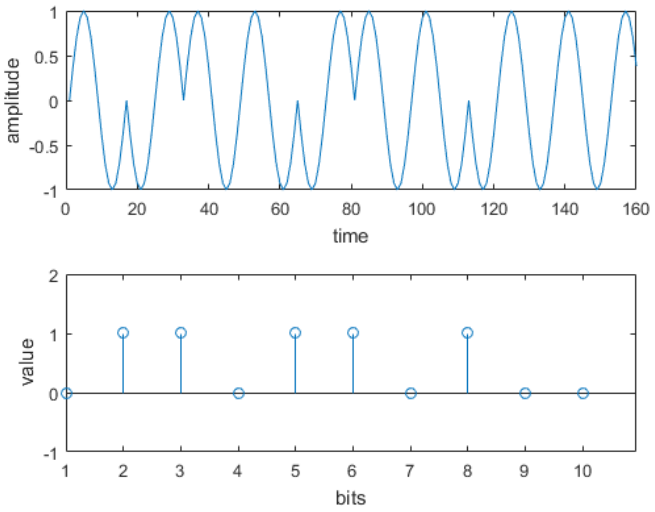

Figure 2.3: Signal modulated using DBPSK (above) and bits that were used (below)

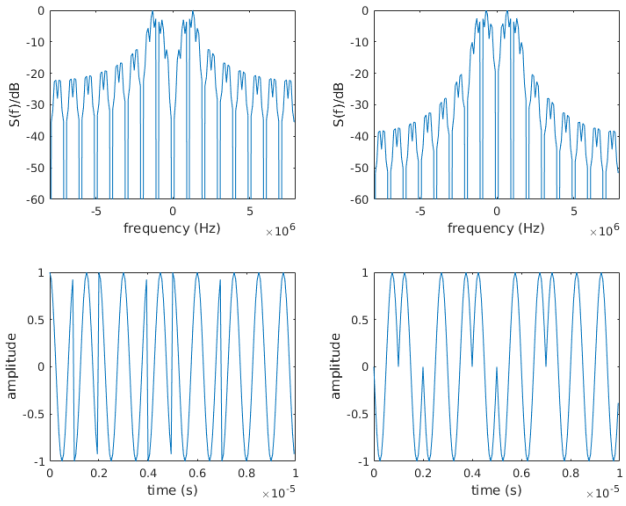

Figure 2.4: Frequency response of two possible BPSK signals with 16 samples per symbol

2.3 Demodulation

It is possible to demodulate a DBPSK signal in time domain or in the frequency domain. In this thesis the focus is on the time domain demodulation of the signal. The reason therefore is that I have more knowledge with demodulating signals in time-domain. The demodulation of a BPSK signal starts with a mixing process to remove the carrier frequency out of the signal. This can be done by multiplying the received signal with a cosine at the carrier frequency. This demodulation step can be described mathematically. The multiplication of two cosines can be rewritten using Equation 2.3. The result is a sum of two cosines, one with the frequency difference and one with the sum of the two frequencies.

$$cos(\alpha) \cdot cos(\beta) = \frac{cos(\alpha + \beta) + cos(\alpha - \beta)}{2} \tag{2.3}$$

A simplified version of the received signal $x(t)$ is given by Equation 2.4. Where $m(t)$ is the modulation signal and ω_c is the carrier frequency. To demodulate the signal it can be multiplied with a signal at the same frequency, see Equation 2.5. This signal is either the reference signal for coherent demodulation of a signal that is generated using the local oscillator in case of non-coherent demodulation. The result of this multiplication is given by Equation 2.6. One part of the signal is now independent of the carrier frequency, this part is useful to further demodulate the signal. With a low-pass filter the high frequency part can be removed from the signal.

$$x(t) = cos\Big(\omega_c t + \pi m(t)\Big) \tag{2.4}$$

$$y(t) = x(t) \cdot cos(\omega_c t) \tag{2.5}$$

$$y(t) = \frac{cos\Big(2\omega_c t + \pi m(t)\Big) + cos\Big(\pi m(t)\Big)}{2} \tag{2.6}$$

Assuming that an ideal low-pass filter is used, the result of the filtering process is given by Equation 2.7. It is known that $m(t)$ contains only zeros and ones (the actual bits). This signal is multiplied with π within the cosine. The cosine of 0 and π is respectively 1 and -1. That means that $y(t)$ contains directly the bits where a 0 is represented by a value of $-\frac{1}{2}$ and a 1 by $\frac{1}{2}$.

$$y(t) = \frac{1}{2} cos\Big(\pi m(t)\Big) \tag{2.7}$$

So given that m(t) only takes values of 0 and 1, $y(t)$ can be simplified to:

$$y(t) = -\frac{1}{2} | m(t) = 0 \tag{2.8}$$

$$y(t) = \frac{1}{2} | m(t) = 1 \tag{2.9}$$

Which means the signal is completed demodulation. Concluding the demodulation of a BPSK signal can be done with a mixer and a low-pass filter. This demodulation method is assuming that there is no phase or frequency difference between the received and local signal.

When there is an unknown phase difference between the local signal and the received signal another demodulation method should be used. Which is the case for non-coherent demodulation. A second mixer should be added to the demodulator. This second mixer should use a local signal that is orthogonal with the other local signal. When $x(t)$ has an unknown phase difference it is described by Equation 2.10.

$$x(t) = cos\Big(\omega_c t + \pi m(t) + \phi\Big) \tag{2.10}$$

where ω_c is the carrier frequency, $m(t)$ the modulation signal and ϕ the unknown phase. If the received signal is mixed with a cosine and sine separately the multiplication results are given by Equation 2.11 and 2.12. The result of the multiplication with a cosine will be referred to as the I component of the signal. The result of the multiplication with a sine will be referred to as the Q component of the signal.

$$I(t) = \frac{cos\Big(2\omega_c t + \pi m(t) + \phi\Big) + cos\Big(\pi m(t) + \phi\Big)}{2} \tag{2.11}$$

$$Q(t) = \frac{sin\Big(2\omega_c t + \pi m(t) + \phi\Big) + sin\Big(\pi m(t) + \phi\Big)}{2} \tag{2.12}$$

In Figure 2.5 the signals $I(t)$ and $Q(t)$ are depicted. In this case the received signal and the local generated signal at the receiver are in phase with each other.

When both multiplication results are filtered with an ideal low-pass filter the results become:

$$I(t) = \frac{1}{2} cos\Big(\pi m(t) + \phi\Big) \tag{2.13}$$

$$Q(t) = \frac{1}{2} sin\Big(\pi m(t) + \phi\Big) \tag{2.14}$$

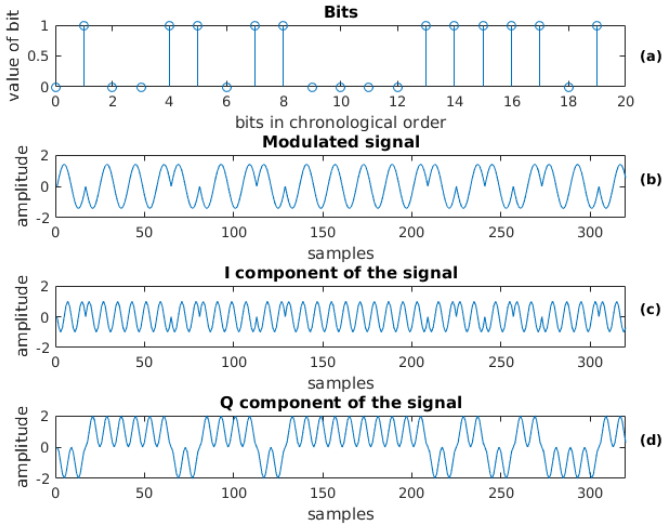

Figure 2.5: Signals showing the bits that are modulated (a), modulated DBPSK signal (b), I component (c) and Q component (d) of the signal after mixing

Assuming that $m(t)$ can only be 0 or 1 the results can be split up:

$$I(t) = \frac{1}{2}cos\Big(\phi\Big)|m(t) = 0 \tag{2.15}$$

$$I(t) = -\frac{1}{2}cos\Big(\phi\Big)|m(t) = 1 \tag{2.16}$$

$$Q(t) = \frac{1}{2}sin\Big(\phi\Big)|m(t) = 0 \tag{2.17}$$

$$Q(t) = -\frac{1}{2}sin\Big(\phi\Big)|m(t) = 1 \tag{2.18}$$

Now there are two signals ($I(t)$ and $Q(t)$) that both contain a part of the modulated signal. It depends on the phase difference how the signal is divided over these two signals. When the received signal and the local signal are in phase, $I(t)$ will contain all information. When they are 90 degrees out of phase all information will be in $Q(t)$. If the phase difference is somewhere in between the information is divided over $I(t)$ and $Q(t)$. For non-coherent demodulation it is not known if the signals are in phase or not.

Because the bits are now varying between 1 and -1 the signal can be differential decoded by multipying the signals with a delayed version of themselves. The result will be a signal that contains the symbols varying between -1 and 1. After this differential multiplication $I(t)$ and $Q(t)$ can be summed and the

result is the completely demodulated signal. In Figure 2.6 the resulting signals are depicted. The last signal is the demodulated signal. When this signal is sampled at the right sample moments the bits can be recovered. How the right sampling moment is determined will be explained in the next section.

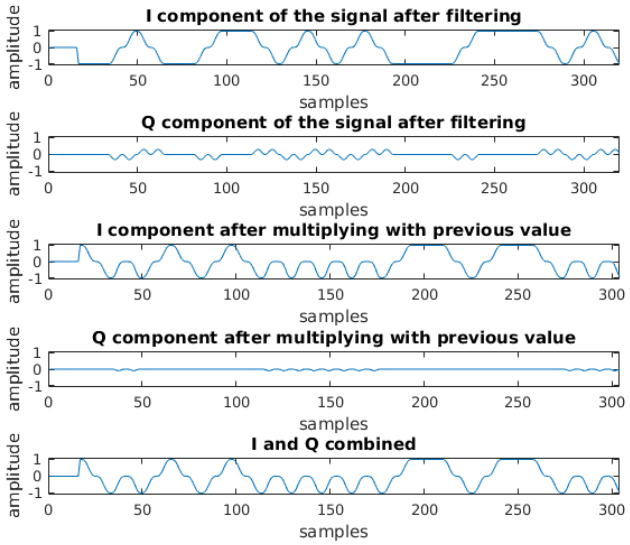

Figure 2.6: Signals showing respectively the filtered I and Q component of the signal, differential I and Q signal and the sum of both differential signals.

2.4 Symbol Time Recovery

In the previous sections it was mentioned that the signal should be sampled at the right sampling moments. When the signal is not sampled at the right moments the change of an error increases. When the worst sampling moment is used the signal is completely lost. For non-coherent demodulation to get the correct sample moments the demodulator should include a symbol recovering algorithm. A few possible algorithms are discussed below. The discussed algorithms are algorithms consisting of simple operations so that they are suitable for implementation on an FPGA.

2.4.1 Early Late

The timing recovery consists of an error function to estimate the error that was made. The error is passed through a loop filter to get the right values that are necessary to correct the timing error. There are a lot of error functions that can be used, one is the early late algorithm [5]. The error function is given

by Equation 2.19. There are three samples per symbol used to estimate the error, one just before the actual sample and one just after the actual sample.

$$e(n) = \Big(x[nT + T_s] - x[nT - T_s]\Big)x[nT] \tag{2.19}$$

where $e(n)$ is the error function at sample moment n, $x(n)$ is the oversampled demodulated signal at sample moment n, T is the symbol period and T_s is smaller or equal to halve the symbol period.

2.4.2 Gardner

Another algorithm is the Gardner algorithm [3], of which a simplified version is also used in [13]. The error function for the Gardner algorithm is given by Equation 2.20. Where $e(n)$ is the error for sample n. For each symbol two samples are required to estimate the error. One at the optimal sampling time and one halfway the symbol period.

$$e(n) = \Big(x[nT] - x[(n-1)T]\Big)x[nT - T/2] \tag{2.20}$$

where $e(n)$ is the error function at sample moment n, $x(n)$ is the oversampled demodulated signal at sample moment n and T is the symbol period.

2.4.3 Mueller Muller

The algorithm that uses the least samples per symbol is the Mueller Muller Algorithm [6]. The error function for this algorithm is given by Equation 2.21. The hat indicates the symbol decision that was at that sampling instance. Advantage of this algorithm is that it only needs 1 sample per symbol. This will in turn result in less robustness of the symbol recovery.

$$e(n) = \Big(\hat{x}[nT]x[(n-1)T]\Big) - \Big(x[nT]\hat{x}[(n-1)T]\Big) \tag{2.21}$$

where $e(n)$ is the error function at sample moment n, $x(n)$ is the oversampled demodulated signal at sample moment n and T is the symbol period.

All of the above discussed algorithms need bit transitions to be able to find the right sampling moment. Besides that the algorithm needs a certain time amount for finding the right sample moment. Therefore the first bits that were transmitted will have a higher change of error. When a constant stream of bits will be transmitted this will not be a problem. However when there are a lots of zeros or ones in a burst this can become a problem. Which will result in less robustness against a clock difference.

2.5 Summary

In this chapter the choice for BPSK was discussed. After that, the difference between coherent and non-coherent was made clear. Thereafter, the basics of modulation and demodulation for BPSK were explained. DBPSK modulation is chosen because it is simpler to implement and that technique is easier to create a parallel structure for. Because I have more knowledge about time domain demodulation that one is used in the demodulator instead of frequency domain demodulation. Three symbol time recovering algorithms were discussed and their importance in the demodulation process.

CHAPTER 3

Conventional DBPSK Demodulator

In the previous chapter the basic principals of the demodulation technique for BPSK were discussed. In this chapter the actual design of a sequential DBPSK demodulator is discussed. Sequential means that the samples are processed one after each other in the order that they arrive at the demodulator. In this thesis we refer to this design as the conventional design. After that the GNURadio software is discussed and the implementation of the demodulator in this software. Thereafter the implementation on the FPGA is discussed. The set-up used for the measurements is clarified after that. Finally some measurement results are discussed. This whole chapter will be a basis for our parallel DBPSK demodulator design.

3.1 Design

As discussed earlier it is possible to demodulate the signal in the time domain as well as in the frequency domain. We have chosen for a time domain implementation because there is more personal experience with this implementation. The focus in this thesis is on parallelising a demodulator, therefore it is best to implement a well known demodulation process. Besides that there is more knowledge and information available about time domain demodulation. The design is roughly split in two parts, known as the demodulation part and the sampling part. First the demodulation part will be discussed after which the symbol recovery part will be discussed. At last in this section the top level design which combines these two parts will be discussed.

3.1.1 Demodulation

A demodulator consists of a mixer and a low-pass filter. In the created design a differential demodulation technique is used. A demodulation structure was created which was discussed in [8]. This structure uses an I and Q path to demodulate the signal. These paths are created by multiplying the received signal with a cosine and a sine at the carrier frequency. Both paths are filtered with a moving average filter. Both paths contain a differential multiplier that is used to multiply a delayed version of the signal with the signal. In this way it can be determined if there was a transition in the signal from positive to negative, from negative to positive or that there was no transition. The signals can be added together after this multiplier.

As discussed the received signal should be multiplied with a sine and cosine which are generated at the receiver side with a local oscillator. In Figure 3.1 the block scheme of a part of the demodulator is depicted, this block is referred to as the "demod block". It was chosen to create the block in this way so that the same block can be used for both the I and Q demodulation path. The inputs are the received signal and a local signal. This local signal should be

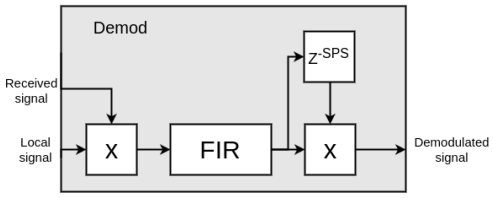

Figure 3.1: Schematic of demodulation block

a cosine and a sine, for respectively the I and Q path. The outputs are the I and Q part of the demodulated signal.

The Finite Impulse Response (FIR) filter in the demodulation block should sample over exactly one symbol period. A matched filter should be used to filter the signal. The modulation signal used by BPSK is a square wave. For a square wave a moving average filter is the matched filter.

For a complete demodulation two demodulation blocks are necessary. One of them has as input a sine, the other one a cosine, the output of both blocks can be added and result in the completely demodulated signal. The block diagram of this demodulation block is depicted in Figure 3.2.

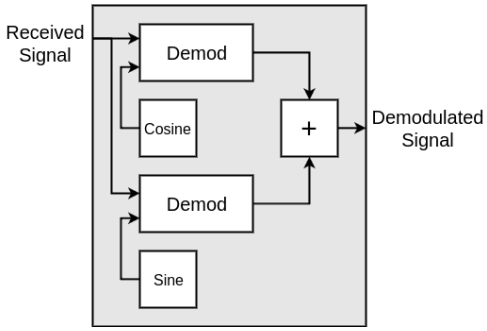

Figure 3.2: Schematic of demodulation block

3.1.2 Sampling

The next step in the demodulation is finding the optimal sampling moment. In the previous chapter a few algorithms were discussed. In our design the early late algorithm is implemented. This one is chosen because it is a relatively robust algorithm. Disadvantages is that it requires an oversampling rate of 3 times, but the extra hardware that it costs is available and is small compared to the rest of the design. The early late algorithm is a bit simplified to be able to implement it on the FPGA without using too much resources. The simplified version only uses the sign of the samples to determine the error.

A shift register is used to store the samples, every time a new sample is available it will shift in the register and the oldest sample will shift out of the register. From this register for each symbol 3 samples are read, this is illustrated in Figure 3.3. One sample is before, one is exact at and the last one is after the used sample moment. The sample register has $1.5 \cdot SPS$ number of places. This minimal size is required because the best sampling moment could occur everywhere in a series of $1 \cdot SPS$ samples. The early sample should be taken $0.25 \cdot SPS$ before the used sample and the late sample should be taken $0.25 \cdot SPS$ after the used sample. Therefore the register should be $0.5 \cdot SPS$ longer than $1 \cdot SPS$. In the depicted case there are 8 SPS and the used sample moment is at 0. In Figure 3.4 the same register is depicted but now the used sample moment is at 7.

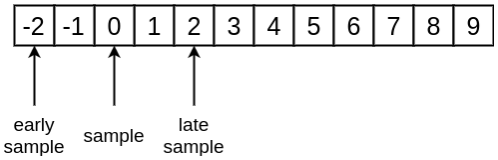

Figure 3.3: Scheme of the sampling shift register (sample moment = 0, SPS = 8)

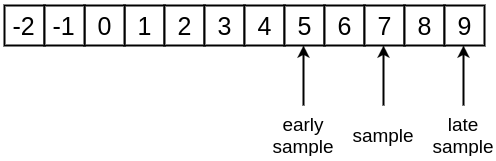

Figure 3.4: Scheme of the sampling shift register (sample moment = 7, SPS = 8)

With the three samples that are read from the register it is decided if the next sample moment should be earlier or later than the current one. The working is described with the following code:

```
if(sample < 0) {
  if(early_sample < 0 && late_sample >= 0){
    sample_moment--;
  } else if(early_sample >= 0 && late_sample < 0){
    sample_moment++;
  }
} else {
  if(early_sample < 0 && late_sample >= 0){
    sample_moment++;
  } else if(early_sample >= 0 && late_sample < 0){
    sample_moment--;
  }
}
```

The code listed above will change the sample moment every time that the sample moment is close to a bit transition. The ideal sampling moment is

halfway between two bit transitions. When the sample moment is at the end of the register and it should move more to the right it is reset to 0 again. The opposite is implemented for the beginning of the register. In Figure 3.5 this is illustrated. This figures illustrates a simplified version of the register where the early and late sample are ignored. The sample moment can change along the black arrow, when it reaches the end it can also move along the red arrow. The red arrow indicates the resets that were described before.

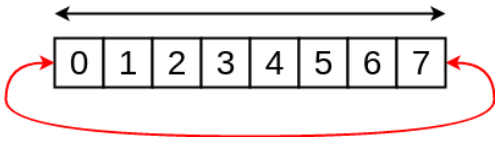

Figure 3.5: Scheme of the sampling shift register with arrows indicating how the sample moment can change in time

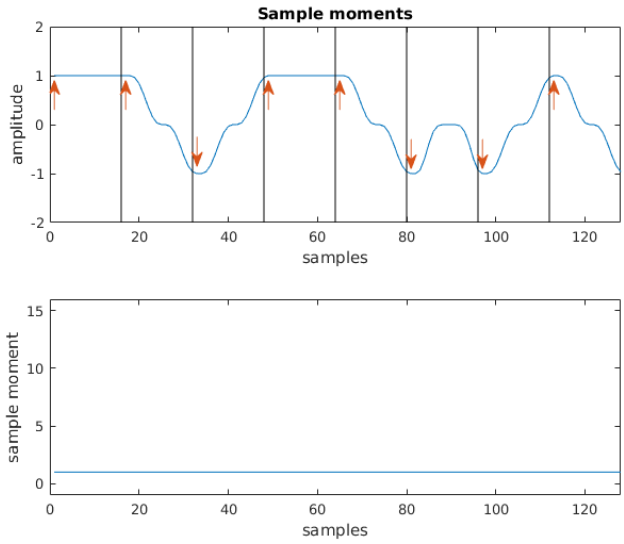

Figure 3.6: Plot of the signal with the sample moments (arrows) that are used

At these resets bits can be lost, but it is possible to compensate for this with additional hardware. When the sample moment is reset from the end of the register to the beginning the next sample will be the same as the previous. Therefore the extra hardware should skip a sample at this moment. When the sample moment is reset from the beginning to the end a sample gets lost without extra hardware. Therefore an extra sample should be taken at this moment. That the reset of the sample moment can cause problems is illustrated. In Figure 3.6 the ideal sample moment of the demodulated signal is depicted. In Figure 3.7 the non-ideal sample moment is depicted, in this case the sample moment is reset. Due to this reset one bit has been lost, which

can be seen by comparing the samples that are token in Figure 3.6 with the ones token in Figure 3.7.

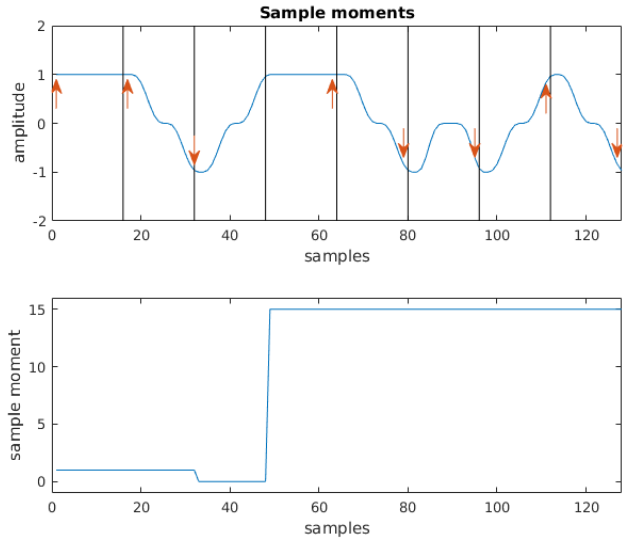

Figure 3.7: Plot of the signal with the sample moments (arrows) that are used

When there is a clock difference between the transmitter and receiver the sample moment will shift over time. Every time the sample moment reaches the end or beginning of the register it will be reset. Due to noise it is possible that the sample moment is reset multiple times at the edge of the register. Therefore it is better to use a sample moment that has a range that is twice as large. To accomplish that the size of the register should be increased to 2.5 times the symbol period. Instead of resetting it from the end to the beginning and the other way around it can be reset to halfway the register. In that way the number of resets is reduced, because resets can not occur right after each other, when the sample moment is halfway the register it cannot be reset. An illustration of this implementation is depicted in Figure 3.8. The sample moment can change along the black arrows when it reaches the end or beginning of the register it can move along the red arrows. The red arrows indicate the resets that were described. It can be seen that the resets cannot occur right after each other, because from the middle of the register the sample moment can only change along the black arrows.

In the case that there is no compensation for bit loss at the resets this can increase the performance. When there is compensation it is not necessary to increase the size of the register. For simplicity there is chosen for a larger register without compensation for bit loss when the sample moment is reset.

The effect of this choice will be discussed in the results section.

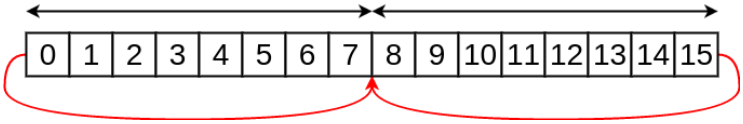

Figure 3.8: Scheme of the bigger sampling shift register with arrows indicating how the sample moment can change in time

The Early Late sampler requires a smooth input signal so that the algorithm functions. The signal that comes out of the demodulator is not smooth in between the bit transitions, this could be seen in Figure 2.6. The early late algorithm cannot always find the right sample moment. Therefore the input needs to be filtered to get a smooth signal so that the early late algorithm functions better. In Figure 3.9 the used sample moments are depicted. The circle indicates a local maximum of the signal which will be indicated by the early late algorithm as the best sample moment. The local maximum is caused by the differential multiplier, the signal should not be sampled at this moment because there is no actual symbol here. In Figure 3.10 the sample moments of the filtered signal are depicted. Now there is no local maximum any more. The sample moments will now be more precise.

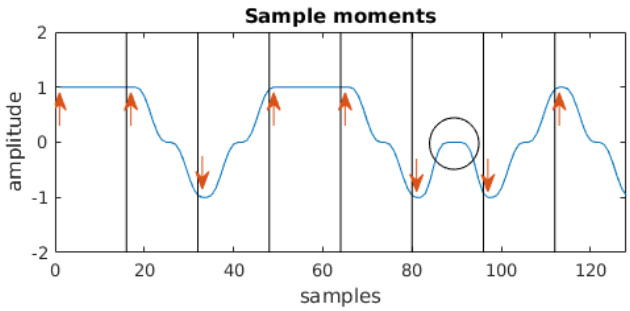

Figure 3.9: Plot of the unfiltered signal with the sample moments (arrows) that are used, the circle indicates a local maximum that is not a symbol

The symbol decision is made based on the unfiltered signal. In that signal the distance in amplitude between the distinct symbols is larger. This can be seen by looking at the eye diagrams which are depicted in Figure 3.11 and 3.12. What can be seen is that the eye of the unfiltered signal has a larger opening. For the filtered signal there are multiple positive levels and one negative and for the unfiltered signal there are only two levels.

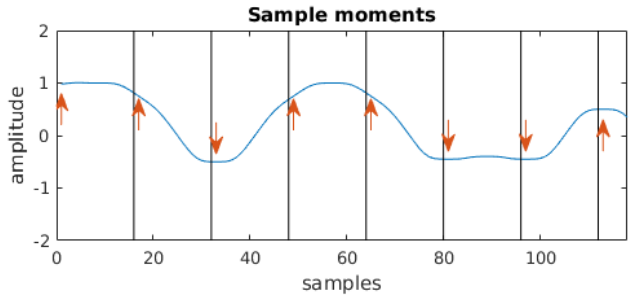

Figure 3.10: Plot of the filtered signal with the sample moments (arrows) that are used

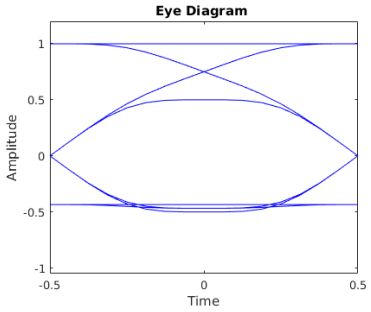

Figure 3.11: Eye diagram of the filtered signal

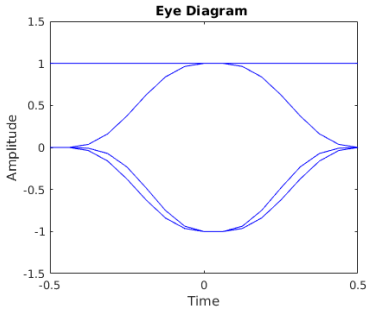

Figure 3.12: Eye diagram of the unfiltered signal

3.1.3 Top Level Design

The demodulator design has to be adapted because an intermediate frequency is used in the RF frontend. In the RF frontend two mixers are used to multiply the received signal with a sine and cosine at a frequency just below the carrier frequency. The resulting signals are the I and Q component of the signal at an intermediate frequency. Both signals are sampled and used for further demodulation. A method to convert the signals from the intermediate frequency to a zero intermediate frequency was described in [2]. This design uses four mixers for the conversion. The resulting signals are added and subtracted to get the right I and Q signals. In Figure 3.13 the design of a demodulator is depicted using the method with four multipliers, the part inside the grey block is the mixer design that was presented in [2].

In our design a slightly different structure is used, in which the previous designed demod blocks can be used. The FIR filter is moved in front of the adders, which can be done be cause both operations are add operations. The most left adders can be combined with the most right adder by moving the differentiating operation in front of the left adders. The resulting signal is differential and therefore the subtracter should be changed in an adder. Now the previous discussed demodulation blocks can be used in the design. The resulting top level design is depicted in Figure 3.14. The demodulated signal

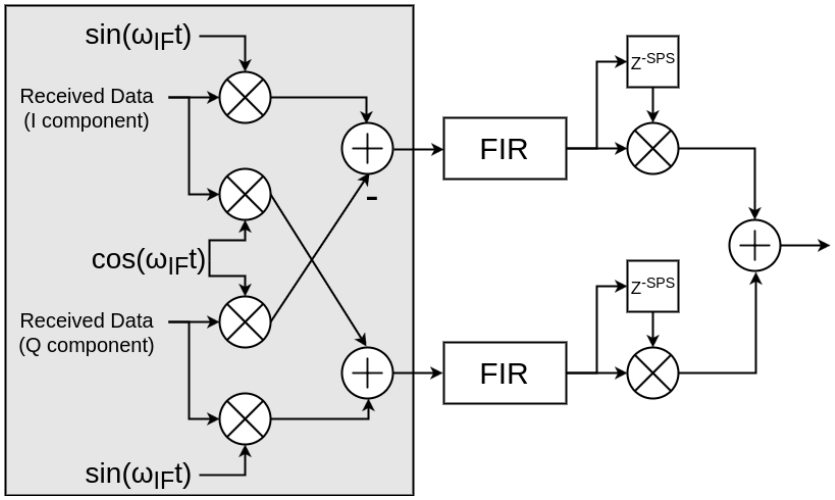

Figure 3.13: Schematic of the top level design demodulator using four mixers

will be sampled using the early late algorithm. The early late sampler also performs the bit decisions. The used design is not optimal from a resources perspective, the number of multipliers and FIR filters has increased.

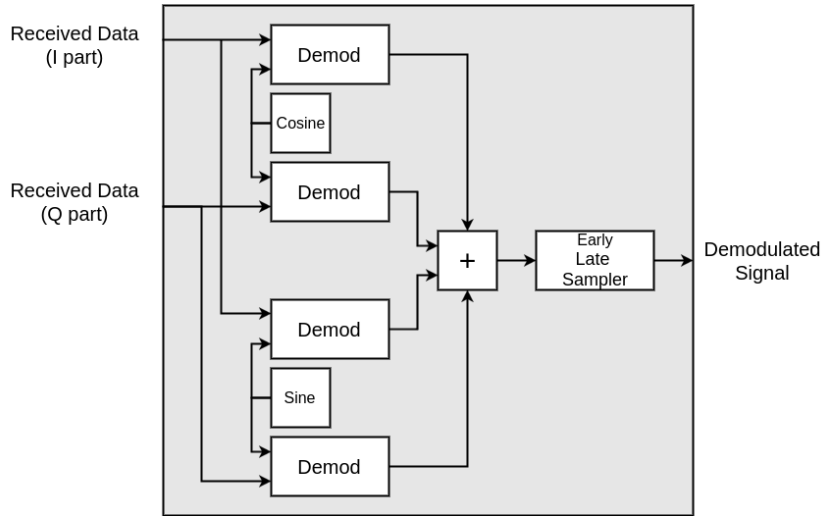

Figure 3.14: Schematic of the top level design demodulator including early late sampler

3.2 Implementation

3.2.1 GNURadio

The test set-up, of which the description will follow in Section 3.3, makes use of software called GNURadio. With this software it is possible to design software defined radios. In GNURadio it is possible to create signal processing flow graphs. The flow graphs can be created with blocks that are included in the software. Besides the standard blocks, the software offers the possibility to define custom blocks. These blocks can be written in C++ or Python.

At the CAES group there are USRP (N210) boxes from Ettus Research available. With these boxes it is possible to output an analogue RF signal that was defined in GNURadio. The boxes can also receive signals from their input.

In GNURadio a demodulator was designed as described in the previous section. The demodulator part consists of standard blocks that are by default available in GNURadio. The early late sampler was developed during the thesis by using C++. In Figure 3.15 the design of the demod block in GNURadio is depicted. The design is exactly the same as described above, it can be compared with Figure 3.1. The length of the moving average filter and the amount of delay are dependent of the number of samples per symbol. In Figure 3.15 for example there are 16 samples per symbol.

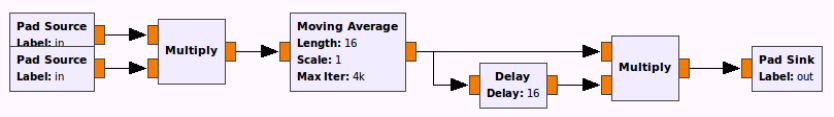

Figure 3.15: Demodulator design in GNURadio

This demod block is used in another block which does the complete demodulation. This block is depicted in Figure 3.16. The design of this block can be compared with the block scheme depicted in Figure 3.14. The demod block that was described above has as parameter the number of samples per symbol. The frequency of the sine and cosine are determined by dividing the sample rate by the number of samples. The input of this block is the signal that was received from the USRP. The output is the oversampled demodulated signal.

The top level flow graph of the demodulator in GNURadio is depicted in Figure 3.17. The left most block outputs the signal that was received by

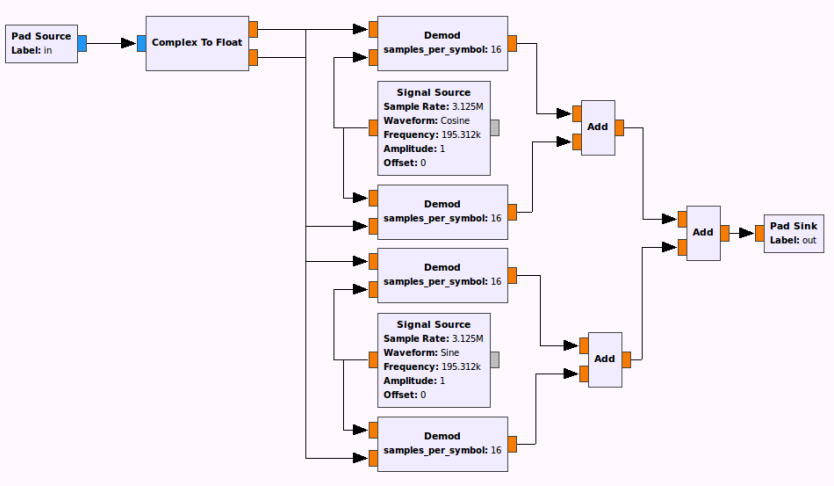

Figure 3.16: Higher order demodulator block design in GNURadio

the USRP. The demodulator block has as parameters the number of samples per symbol and the sample rate, both are necessary to demodulate the signal correctly. The early late sampler also needs both parameters, but the sample rate is in this case defined by the rate at which the samples appear at the input of the block. The early late sampling algorithm is implemented as described in Section 3.1.2 and is written in C++.

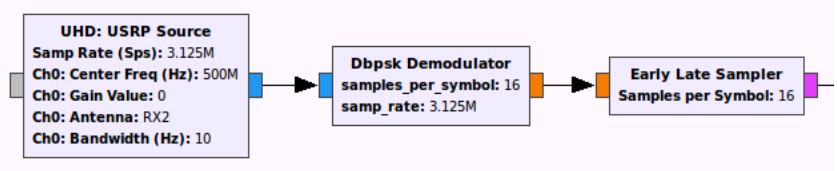

Figure 3.17: Complete signal flow graph at the receiver side

3.2.2 FPGA

The actual design of the parallel demodulator is made for an FPGA. How this demodulator is created will be discussed in the next chapter. In this subsection the implementation of a conventional DBPSK demodulator on an FPGA is described.

Hardware Accelerators

The demodulator is designed for a Xilinx ML-605 development board [19], on which the Starburst multi processor system is installed. The Starburst system

is created by the CAES group at the University of Twente. On the ML-605 board there is a Virtex 6 FPGA. A Bitshark FMC-1RX [12] is interfaced with the ML-605 board. Hardware accelerators can be created for this FPGA which can be integrated in the Starburst system. These accelerators can be connected to each other via a ring, via which data can stream from one accelerator to another.

Design

A hardware accelerator is available that can be used to read samples from an ADC in the RF frontend. A demodulator accelerator is created during this thesis. With another hardware accelerator it is possible to output samples to a buffer. In Figure 3.18 a flow graph of the hardware accelerators is depicted.

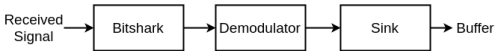

Figure 3.18: Flow graph of the hardware accelerators that are used

Xilinx modules are used to design the demodulator accelerator. With the Xilinx CORE Generator software it is possible to create hardware description files of these modules which can be used in the hardware accelerator. In Figure 3.19 the schematic of the demod block is depicted, in red Xilinx blocks are highlighted. A FIR filter was used as a moving average filter [14]. A multiplier block designed by Xilinx was used [16]. A delay block had to be created because such a block was not available in the CORE Generator software.

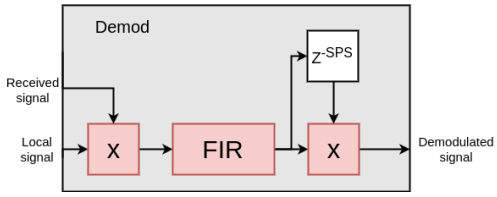

Figure 3.19: Schematic of the demod block (in red the Xilinx modules)

The early late sampler was, with exception of the FIR filter, completely designed during this thesis. The FIR filter is exactly the same as was used in the demod block. The sample register is created with a register file where each sample that comes in is stored. For every symbol three samples are read from this register file. These samples are with some logic combined to decide if the sample moment should change. This sample moment is the output of the early late block, which will be used to sample the demodulated signal.

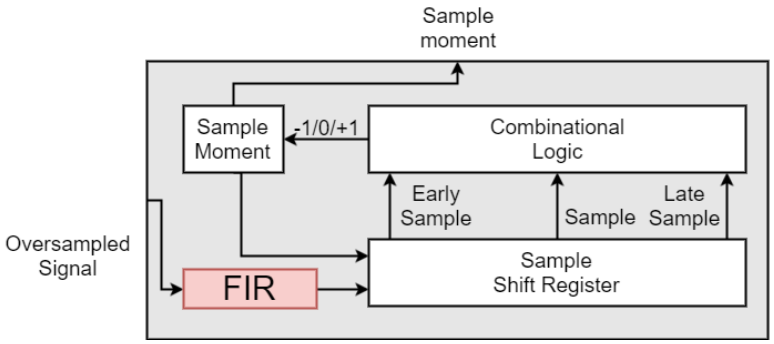

Figure 3.20: Schematic of the early late block (in red the Xilinx modules)

In Figure 3.21 the schematic of the complete demodulation block is depicted, again in red the modules from Xilinx that were used. The sine and cosine block are both Read-only Memory (ROM) blocks created with the block memory generator from Xilinx [18]. A read address pointer is used for reading values from the ROMs. In these ROMs samples of a cosine and sine are stored. The adder was created using the adder/subtracter block from Xilinx [15]. The demod and early late block in the schematic are the blocks that were described before. The sampler block consists of a shift register where the demodulated signal is stored. The sample moment determines which register place of the shift register will be forwarded to the output.

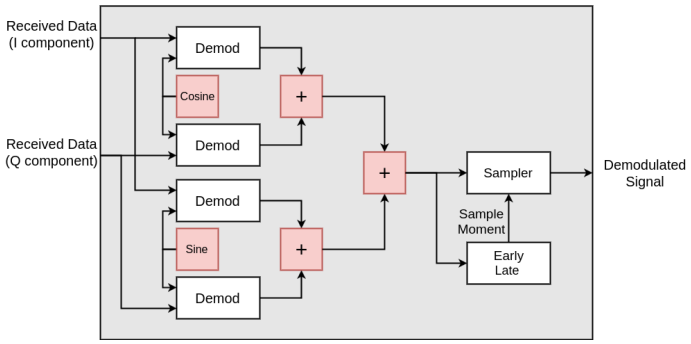

Figure 3.21: Schematic of the complete demodulation block (in red the Xilinx modules)

As described before, the delay block was created during this thesis. In the implementation of the delay a register file was used, which will be implemented with distributed Random Access Memory (RAM). It is probably better to use a RAM based shift register [17] instead of a custom made delay with file register. The reason is in that case the register will be implemented using block RAM instead of using distributed RAM. This will save resources on the FPGA, this depends however on the number of RAMs are available. A

implementation test shows that when a RAM based shift register is used this is optimised without using RAM blocks. However when the standard Xilinx block is used the chance of bugs in the design is smaller. The same block can also be used for the sampler block, in that case the option for variable length should be used to be able to use the right sample moment. When these modifications are applied to the design only the early late block contains self defined blocks. By using the blocks that are created by Xilinx the created hardware will probably be more efficient.

3.3 Results

Some measurements are done to validate if the demodulator is functioning as expected. First the set-up is discussed, after that the measurements are discussed.

3.3.1 Set-up

The schematic of the set-up is depicted in Figure 3.22. A signal is generated in GNURadio this signal is send to the USRP. The output of the USRP is via a cable connected to a Bitshark ADC. This ADC is interfaced with the FPGA, with the help of hardware accelerators the signal is demodulated. In the Linux core on the Starburst system it is possible to read outputs of the FPGA. A package detector block is added in the hardware accelerator that counts the number of detected packages. This number can be read via the USB port of the ML605 board.

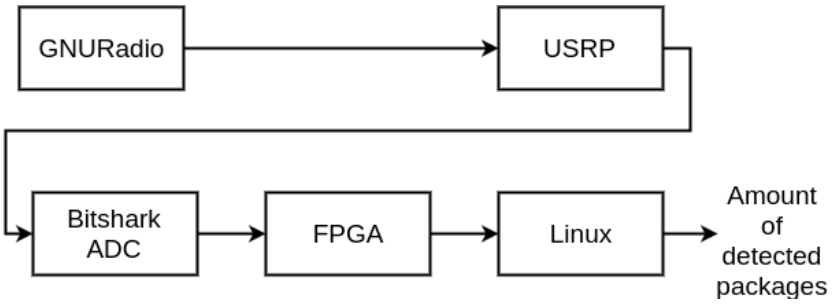

Figure 3.22: Schematic of the setup that is used to perform the measurements

3.3.2 PER

Test signals were created using GNURadio to test the performance of the designed demodulator. Both designs, the one in GNURadio and the hardware accelerator are tested and compared with each other. Because there is no

synchronisation between the transmitter and the receiver it is difficult to determine the BER. Therefore it is chosen to transmit packets and measure how much of the packets are received. This is done by adding a block in the modulator in GNURadio, this block adds a header to the signal. At the receiver side a detector is added to see if a header is received. The number of headers that is detected is used to determine the PER of the system. Disadvantage of this method is that the PER is probably a best case PER, because the header is designed to be detected easily. But the PER that is calculated can be useful to compare the conventional demodulator with the parallel design. In Figure 3.23 the flow graph that is used to create the test signals is depicted. The Simple Framer block adds a header, a counter and an end byte to the signal. Only the header is used at the receiver side to determine the PER. Drawback of this method is that there is not much information about the performance of the time synchronisation block. For actual data the performance can be worse, due to less bit transitions. The DBPSK modulator uses Equation 2.1 to modulate the signal, where $x(n)$ contains the interpolated differential bits. The differential bits are created inside the DBPSK modulator. The channel model is used to add noise and different sampling offsets to the signal, so that the performance of the demodulator could be tested.

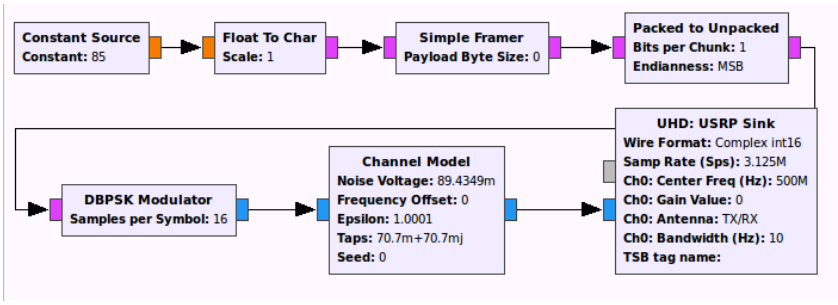

Figure 3.23: Flow graph that was used in GNURadio to create test signals

At the receiver side a frame detector block is added. In this block the last 64 bits are stored and compared with the correct header. When the stored bits are as expected a counter is increased. The counter and the exact sample moment are stored in a buffer. After a certain amount of time they are read and saved by a software program that runs on the Linux core.

3.3.3 Measurements

In Figure 3.24 the PER of the GNURadio and FPGA implementation of the demodulator is depicted. In yellow the theoretical value of the PER is depicted, however this is assuming that there is no correlation between errors. This theoretical PER relation is created by combining the theoretical value of

the BER with Equation 3.1.

$$p_p = 1 - (1 - p_e)^N \tag{3.1}$$

where p_p is the probability of a packet error, p_e the probability of a bit error and N the size of the packet in bits. As can be seen for low SNR the implementation of the demodulator is performing better than can be expected for uncorrelated errors. Which is probably caused by the fact that the errors are dependent on each other. For DBPSK this is indeed the case because the chance of paired errors is higher due to the differential encoding. In [11] the PER for DBPSK is calculated, conclusion is that for DBPSK it can not be assumed that the errors are independent of each other. However it seems that the measured PER is still better than the theoretical value that is given in [11], it is not clear why this is the case.

What also can be seen in the figure is that the PER of the FPGA has a certain floor. It does not matter how high the SNR is, 0.4% of the packets will never be detected. Most likely this is caused by an error in the VHDL code because this error is not present in the GNURadio implementation.

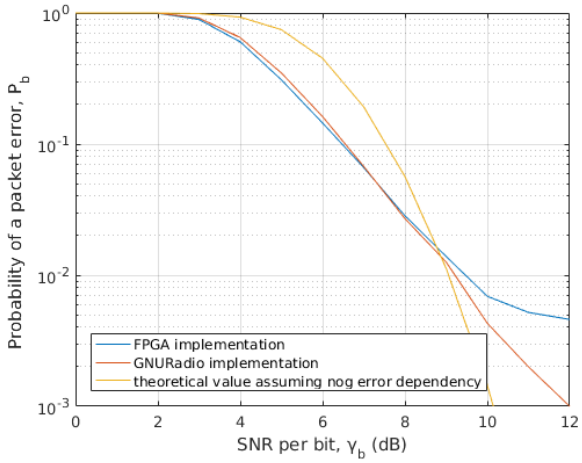

Figure 3.24: PER for the FPGA implementation, GNURadio implementation and the theoretical value assuming no error dependency

In Figure 3.25 the output of the FPGA implementation is depicted over a time interval. In theory it is possible to detect a packet at every sample that is depicted in the graph. In the graph it can be seen that the exact sample moment changes over time, this is caused by a difference in clock frequency between the crystal in the USRP and the one on the FPGA board. The difference between the two can be calculated by dividing the total number of

samples in a certain range by the amount the sample moment has changed. Their difference was:

$$\frac{number\ of\ samples}{change\ in\ sample\ moment} = \frac{80 \cdot 16 \cdot 10\ 000}{24} \approx 53.3 \cdot 10^4 \tag{3.2}$$

where the number of samples is calculated by multiplying the packet length with the number of samples per bit, which is multiplied with the total number of packets. The result indicates that every $53.3 \cdot 10^4$ samples the FPGA takes one sample less. Which means that the difference in clock frequency is 1.9 ppm. This difference is dependent on the accuracy of the crystals used in the USRP, the Bitshark ADC and the ML605 board. In the USRP a Temperature Compensated Crystal Oscillator (TCXO) with an accuracy of 2.5 ppm is used [10]. The ML605 has an oscillator with an frequency accuracy of 50 ppm [19]. There was no frequency accuracy given for the oscillator in the Bitshark RF frontend.

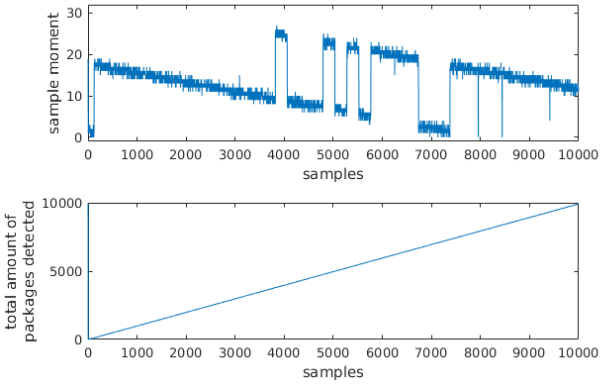

Figure 3.25: Exact sample moment and detected packages over time (SNR per bit = 10 dB)

In GNURadio it is possible to change the sample rate with a re-sampling factor. In this way it can be tested how well the system can handle a clock difference between transmitter and receiver. In Figure 3.26 the PER of the FPGA implementation is depicted for a few re-sampling rates. In Figure 3.27 the exact sample moment and number of detected packages is depicted for a re-sampling rate of 1.00001. This re-sampling rate corresponds to an additional ppm of 10. Using Equation 3.2 the total difference is calculated and is ≈ 12 ppm. For the re-sampling rate of 1.0001 the ppm is $\approx 1.0 \cdot 10^2$, it can be seen that for this value the BER is increasing compared to the situation without re-sampling factor. Given the values of frequency accuracies of the USRP and the ML605 it can be concluded that in worst case the performance can be affected by a difference in clock frequency. During the measurements the clock difference between the USRP and the FPGA were not that extreme that

the results were effected significantly, this can be concluded from the fact that the results for the GNURadio implementation are the same as for the FPGA implementation. The GNURadio implementation is not affected by a clock difference because the transmitter and receiver use the same clock.

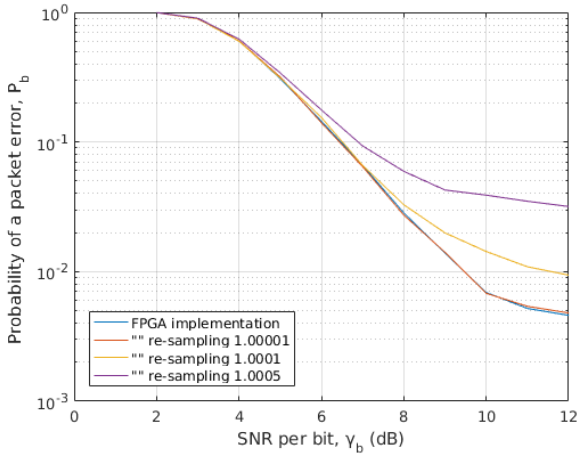

Figure 3.26: PER of FPGA implementation for different re-sampling factors

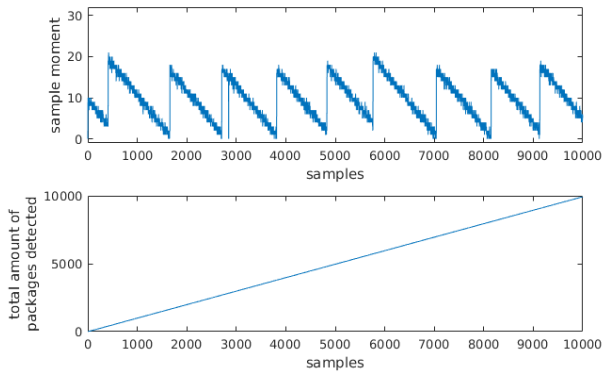

Figure 3.27: Exact sample moment and detected packages over time for a re-sampling factor of 1.00001 (SNR per bit = 10 dB)

But what effect causes the increase of packet loss when the frequency difference is larger? The problem can be found in the reset of the sample moment. Every time such a reset is done a bit is lost, and almost each bit loss will cause a packet loss. When it is assumed that each bit loss results in packet loss, the change of packet loss for a certain clock difference can be

calculated. This can be calculated by calculating the chance that a reset will occur during the time that a packet is received, which is the re-sampling rate multiplied with the packet length. When this chance is taken in to account a PER with correction can be calculated, which is depicted in Figure 3.28.

Probably the chance of a reset during a packet is slightly exaggerated because not all bit losses will result in a packet loss. But the results give an indication that an improvement can be made by adding extra hardware to prevent these bit losses. This extra hardware should add an extra sample when the sampling moment is reset from the end of the register to the beginning. And skip a sample when the sample moment is reset from the beginning of the register to the end. The frame detector should be edited such that it can handle these changes in sample rate. It was chosen not to add this in my project due to time constraints. Besides that, the results are good enough to compare with a parallel implementation. It is best to keep the design simple so that parallelising it will not become too difficult.

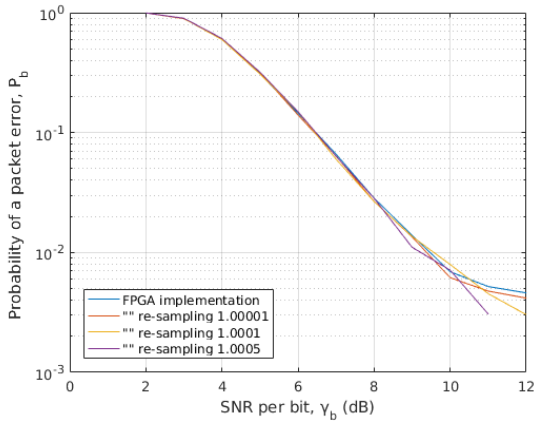

Figure 3.28: PER of FPGA implementation for different re-sampling factors with compensation for packet loss due to resets

During the design process it was chosen to double the length of the sampling register. Measurements were done with a smaller sampling register and it was studied what the effect was of the size of this register. The effect on the PER of a register that is exactly the length of a symbol period is depicted in Figure 3.29. It can be seen that there are a lot of jumps in the sample moment. Every time the sample moment reaches the edge of the sample register a lot of packages will be missed.

In Figure 3.30 the PER is depicted for the implementation with a small sampling register. It can be seen that the performance is a lot worse due to the described effect. Another thing that can be seen is that the result is

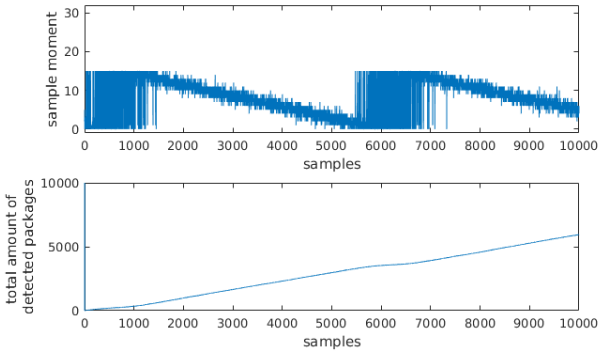

Figure 3.29: Effect of a sampling register that is exactly the length of the symbol period

less dependent of the re-sampling factor. This is according to the previous described theory that the packet loss is caused by the effect of resets in the sample moment. Because the total range in which the sample moment is changing frequently stays the same.

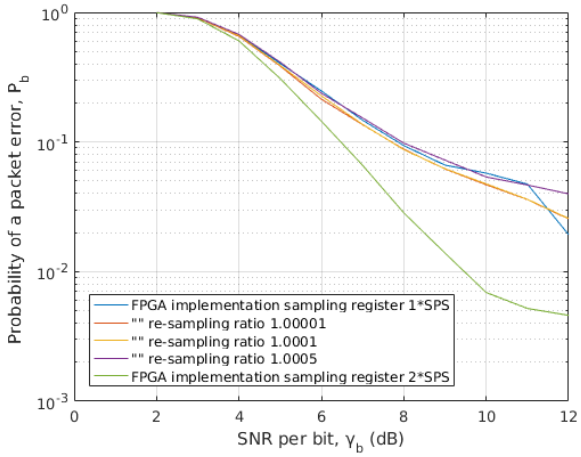

Figure 3.30: PER for FPGA implementation with a sample register that is exactly the length of the symbol period with different re-sampling factors. For comparison the PER of the FPGA implementation with a sample register with a length of twice the symbol period is depicted.

3.4 Summary

In this chapter the design and implementation of a conventional demodulator was discussed. Time-domain demodulation of the signal was selected, because there is more knowledge to implement this and the focus is on parallelising the demodulation process. This design will be the starting point of creating a parallel demodulation structure. The test set-up was discussed together with its disadvantages. Measurements were done and the results were discussed. These results were useful to determine the performance of the design but also to discover some disadvantages of the design. One of the things that was discovered was that increasing the size of the sampling register can improve the PER. A disadvantage of the design is that bit losses will occur when the sample moment is reset. With extra hardware this can be prevented. It was chosen not to implement this due to time constraints. However the current design will still be useful to compare with the parallel implementation. Measurements show that the implementation has a certain PER floor. Due to that floor the PER will never reach 0 %. The results of the parallel design described in the next chapter will be compared to the results that were presented in this chapter.

CHAPTER 4

Parallel DBPSK Demodulator

Subject of this thesis is to design a demodulator on an FPGA which can handle high data rates. To do so a parallel demodulator is designed. In this chapter the design of such a demodulator is discussed. In the previous chapter the design of a conventional DBPSK demodulator was discussed, this design will be the basis of the parallel design. This parallel design will be discussed in this chapter. After which the discussion of the implementation follows. The implementation that is discussed in this chapter will be used in the next chapter to test the performance and compare it with the conventional design.

4.1 Design

An FMC125 ADC of 4DSP [1] was available at the CAES group which has a possibility to sample at 5Gs/s. At this high speed it delivers its samples using 8 parallel channels. However it is from time perspective decided not to use this because it is not interfaced with the Starburst platform yet. Instead a proof of concept is made which can be used to work with parallel samples. It is chosen to create a switch module that switches packets of samples to different demodulation modules. In the next section the functioning of this switch is discussed.

4.1.1 Switch

The switch stores samples from the signal that is received from the ADC in a register. From this register packets of samples with a certain length are forwarded to different demodulation modules. These packets should have a certain overlap to be able to get all bits from the signal. In Figure 4.1 an example is depicted to explain how the packets are divided over the cores. The numbers in the model represent symbol periods of the signal. What can be seen is that the first two symbol periods that are received are forwarded to core 0. These two packages contain the bits that are demodulated by core 0. Two older packages are needed to be able to demodulate the signal correctly. Therefore also symbol period -1 and -2 are forwarded to core 0. The reason for this will be explained later.

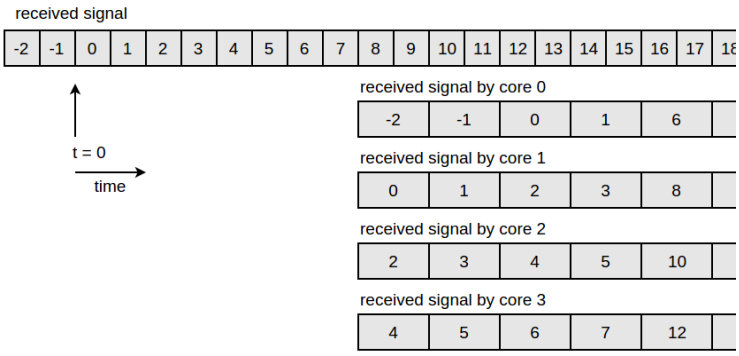

Figure 4.1: Illustration of the functioning of the switch

In the design of the switch it is assumed that samples arrive one after another. For high speed ADCs the samples will probably arrive in parallel. The design will however still function. In Figure 4.2 this is illustrated. In this example the numbers represents samples instead of symbol periods. It can be seen that the samples arrive in parallel. Each core receives a part of the input signal in the correct order. Only 2 cores are depicted in the illustration, but in a real design there should be more cores to keep up with the rate at which the samples arrive.

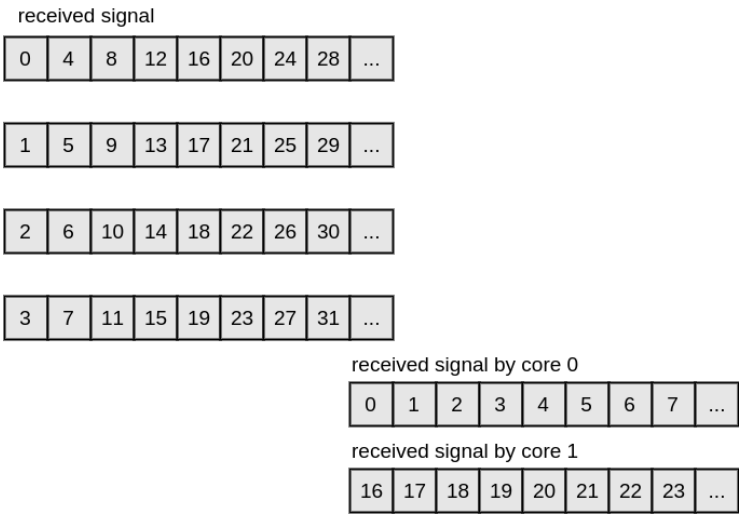

Figure 4.2: Illustration of the functioning of the switch when the samples arrival in parallel

4.1.2 Demodulation

In Figure 4.3 the different blocks within a core are depicted with their inputs and outputs. The symbol periods are coloured to indicate if the periods are valid or invalid. Red means that the data of this symbol period is invalid, green means that the data is valid. It can be seen that after the FIR filter 1 out of 4 symbol periods becomes invalid, that is due to the fact that the filter averages over one symbol period. For the first symbol period in a packet this will result in invalid data because it is averaged with the last symbol period of the previous packet. For the differential multiplier another symbol period becomes invalid, due to the fact that each symbol period is multiplied with the previous. Because of that the symbol period after the invalid symbol period will also become invalid. For this specific example halve of the final output will be invalid. Each package of 4 symbol periods will result in two valid bits. For that reason an overlap in packages between cores is used so that every symbol period will result in a valid bit.

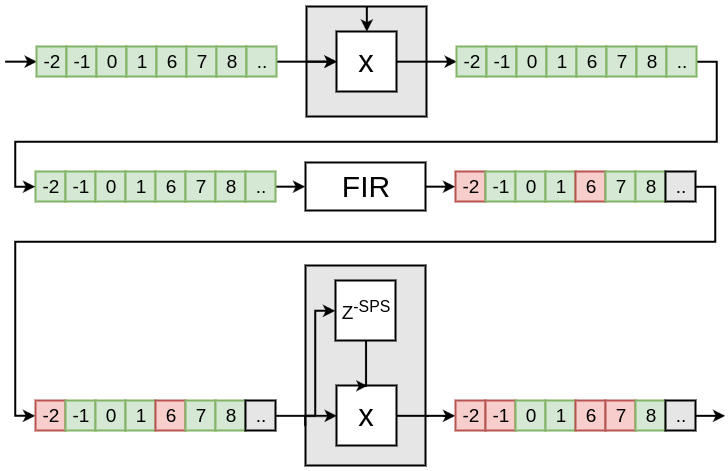

Figure 4.3: Parallel demodulation

For the example it was chosen to use a package length of 4 symbol periods. When 4 demodulation cores are used the clock frequency in the demodulator can be halved. A formula can be created to calculate with which factor the clock frequency will be changed:

$$n = \frac{N}{(N - X)M} \tag{4.1}$$

where N is the length of the packages expressed in symbol periods, X the overlap between the packages expressed in symbol periods and M is the number of demodulation cores. For the example this results in a factor of $\frac{1}{2}$.

In the previous chapter it was discussed how long the sampling register should be. When a sampling register is used that has a length of twice the symbol period the overlap between packets should increase to 3 symbol periods instead of 2. In that case the previous example will not result in a decrease of the clock frequency. The length of the packages or the number of cores should be increased to still have a lower clock frequency in the demodulator. For example 8 demodulation cores could be used instead to still be able to halve the clock frequency. In that case each core would extract one bit out of each package.

4.1.3 Symbol Time Recovery

The demodulation design did not change much, however the symbol time recovery has to change. The most easy implementation would be to place a symbol time recovery on each core that decides for that core what the sample moment should be. This will however cause problems when the separate cores use different sample moments. When the sample register has a length of two symbol periods than this will most likely cause bit loss. In Figure 4.4 an example for the position of the best sample moments in the sampling register are depicted. These ideal sample moments are indicated with arrows. When the different cores do not use the same sample moments this will cause problems.

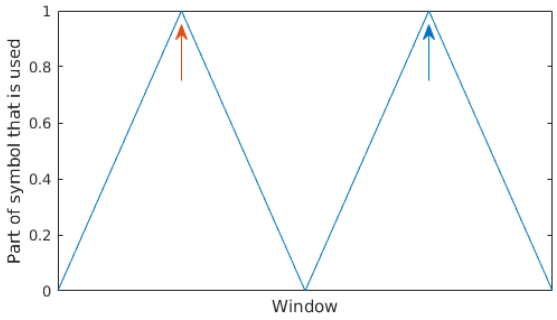

Figure 4.4: Example of the best sample moment within a sampling register with a width of twice the symbol period

For a sampling register with exactly one symbol period length the chance that all cores use the same sample moment is bigger. In Figure 4.5 two examples for the best sample moment are depicted for the smaller register. For the blue example all cores will use the same sample moment. The orange example gives no certainty that all cores will use the same sample moment. Some cores will reset their sample moment earlier then others. For example a core which did not had a symbol transition will keep its old sample moment. Due to noise

it is also possible that one core will reset it and another will not. In that case some will sample at the end and some at the beginning of the period.

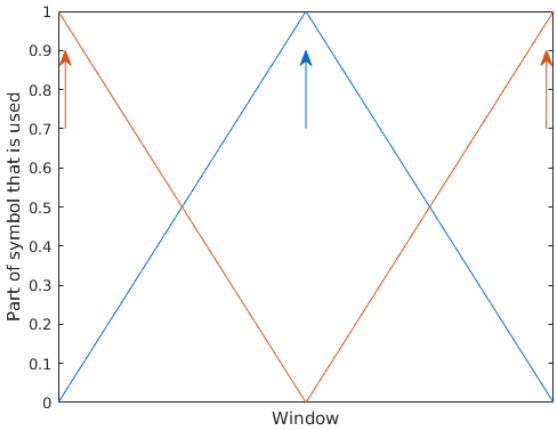

Figure 4.5: Two examples of the best sample moment of a sampling register with a width equal to the symbol period

The simplest solution to the described problem is to decide the sample moment based on just one symbol recovery core. For example the output of the first demodulation core can be used to determine the best sample moment. This sample moment can be forwarded to each core. In this way it is guaranteed that all cores will use the same sample moment. This solution has however a disadvantage, namely the sample moment is not determined very often. When there is no bit transition in the signal received by the first core the sample moment can not change. This reduces the robustness against a clock difference between transmitter and receiver.

It is possible to create a better solution. An example solution is depicted in Figure 4.6. The outputs of multiple symbol recovery loops are added together. With this combined signal it is decided if the sample moment should be changed or not. In this way the design becomes more robust. The algorithm will now function if there is at least one bit transition in one of the cores.

4.2 Implementation

The above discussed design has been implemented on an FPGA. The implementation that has been created did not utilise all optimisations that were discussed above. No parallel implementation has been made in GNURadio. However experiments were done in GNURadio that simulates the parallel design. The biggest part of the parallel design consists of blocks that were

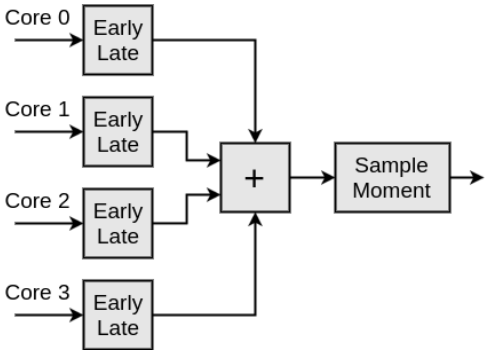

Figure 4.6: Parallel demodulation

created for the conventional design. A block called "switch" was added to perform the switching of packets to the multiple demodulation blocks.

4.2.1 Switch

The switch consists mainly of two registers. In the first register the received signal is stored. The second is used as an output buffer. When the input register is completely filled the contents are copied to the output register. The output register is read and forwarded to four different outputs. This reading is done at a lower rate than the rate at which the samples arrive at the input register. In Figure 4.7 the design of the switch is illustrated. For the design that was created the input register is filled with 8 symbol periods. Two extra symbol periods (indicated in red) are necessary for the overlap that is required. Another output of the switch block indicates that a new package of samples was sent. This information is important for the sampling of the signal, because sampling should not be done on the data that is invalid. In Section 4.1.2 it was explained why parts of the output signal are invalid. When a new package of samples arrives the sampling should pause for 2 symbol periods on each core.

There was chosen for a package length of 4 symbol periods and 4 demodulation blocks. Afterwards this was not the best choice as was discussed in Section 4.1.2. To be able to reduce the clock speed the sampling register can only have a width of one symbol period. This will not give the best results possible. Due to a lack of time the design has not been changed.

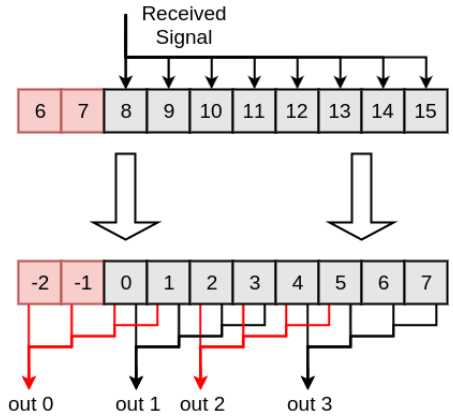

Figure 4.7: Illustration of the functioning of the switch block

4.2.2 Demodulation

The demodulation method of the conventional implementation can be used in the demodulation cores that are used in the parallel implementation. In Figure 4.8 the block scheme of the demodulation core is depicted. The demod block within this design is the same as for the conventional design.

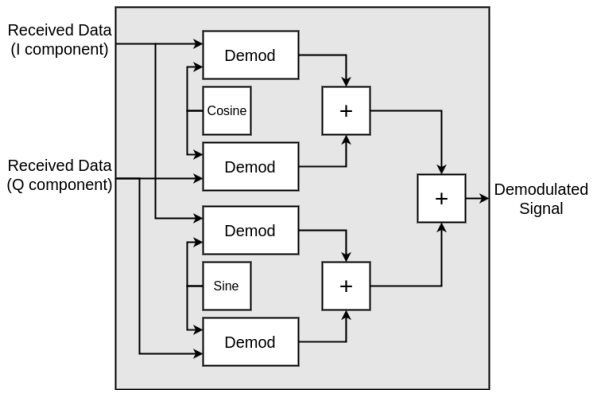

Figure 4.8: Demodulation block used in the parallel implementation

4.2.3 Top Level Implementation

In Figure 4.9 a block scheme of the top level implementation is depicted. As can be seen in the figure each core has its own demodulation and sampling block. The symbol time recovering is only based on the output of the first core. This was chosen for simplicity and due to a lack of time this has not been changed to an improved symbol time recovering loop.

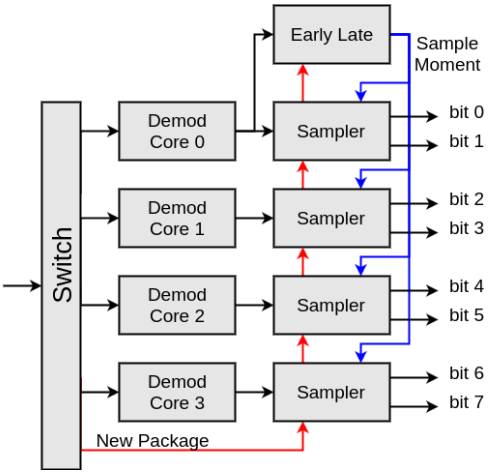

Figure 4.9: Parallel implementation

Symbol Time Recovering

The early late block needs a small adjustment for the parallel implementation. It should not run always but only when valid data is available. The reason why parts of the signal are invalid was discussed in Section 4.1.2. A signal is used to indicate that the first sample of a new packet was transmitted, the first symbol periods after that are invalid and should be ignored. It was chosen to use this signal as a trigger for the algorithm. Therefore the algorithm runs only once for every 8 symbol periods. An improvement can be made here to do it for all symbol periods that are valid. The calculated sample moment is forwarded to each sampler.

Sampling

For the sampling the same adjustment is needed. The sampler has to know when a new package was transmitted by the switch, the data right after that signal will be invalid. In the implementation there is chosen to sample both bits of the core at the same moment, a shift register is used to implement that.

4.3 Conclusion

In this chapter a parallel design was presented. After that it was implemented on an FPGA. Some shortcomings of the design together with possible solutions were discussed. The shortcomings are related to finding the right sample moment. The first shortcoming is that there is bit loss when the sample moment is reset. This resets occur relatively often because the sampling register is exactly one symbol period. The other shortcoming is that the right sample moment is only determined based on one parallel path. Solutions were

presented that should overcome these shortcomings. However due to time constraint my design has still these shortcomings in it. In the next chapter measurement results of the presented parallel design are presented and the results are compared with the results obtained using a conventional demodulator that has been presented in Chapter 3.

CHAPTER 5

Results and Comparison

In this chapter the results of the parallel demodulator are discussed. The results are compared with the results that were presented in Chapter 3. The set-up that is used has already been explained in Section 3.3.1.

5.1 Results Parallel Receiver

In Figure 5.1 the PER of the parallel implementation on the FPGA is depicted for several re-sampling rates. For comparison also the PER of the GNURadio implementation is depicted. What can be seen is that the implementation is performing worse than the GNURadio implementation. But this is probably due to the fact that the sampling register has a width of only one symbol period instead of 2. Therefore it is useful to compare it with the other implementation on the FPGA that was made.

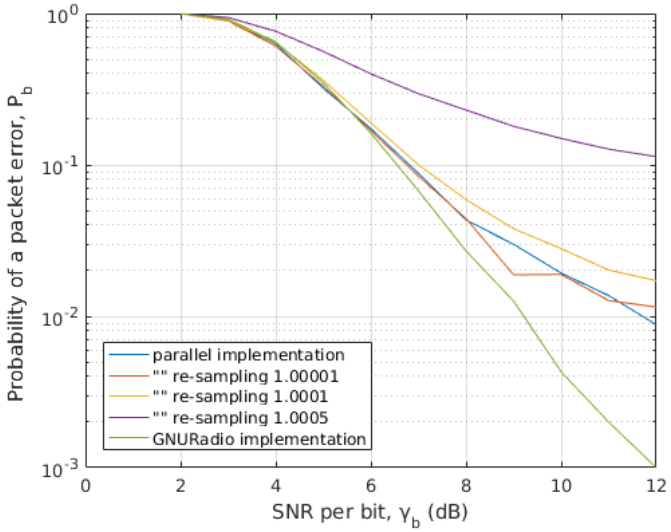

Figure 5.1: PER of parallel implementation for different re-sampling factors, for comparison the PER of the conventional implementation in GNURadio is also depicted.

In Figure 5.2 the PER of different implementations is depicted. What can be seen is that the implementation with the short sampling register performs worse than the parallel implementation which also uses this same register size. The reason for this can be explained with Figure 5.3, which can be compared with Figure 3.29. The sample moment of the parallel implementation has less jumps due to the fact that the sample moment can only change 1 out of 8 times. For comparison the conventional implementation was changed so that the sample moment was only calculated 1 out of 8 times. It can be seen in the graph that the performance of this implementation is comparable with the parallel implementation. This means that the parallel implementation was done correctly and that it only performs worse due to the known shortcomings

of the design. These shortcomings are the small sampling register and the fact that the early late algorithm is slower.

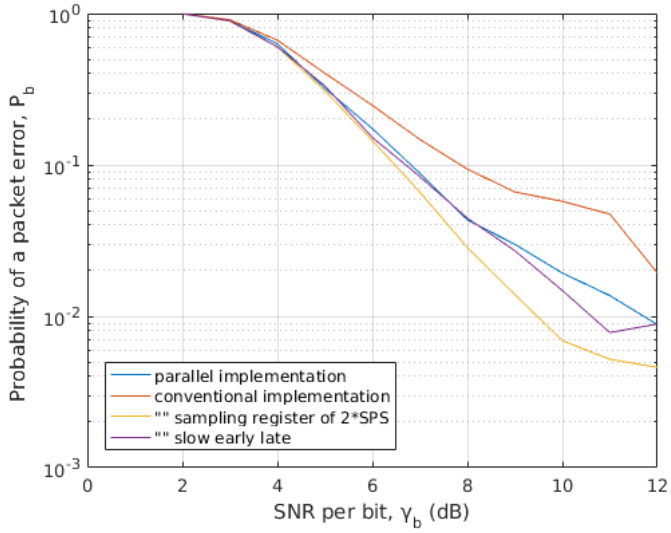

Figure 5.2: PER for different implementations

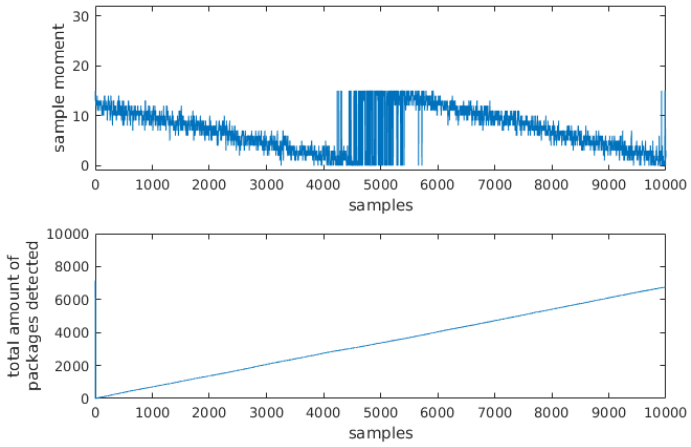

Figure 5.3: Sample moment and total number of detected packages over time for the parallel implementation

Figure 5.4 shows the same PERs as Figure 5.2 but with a re-sampling rate of 1.0005. What can be seen is that the parallel implementation is now worse than the conventional implementation. Due to the high re-sampling rate the early late algorithm is too slow and cannot all the time decide the right sampling moment. That the design is less robust to a clock difference

was already predicted during the design process.

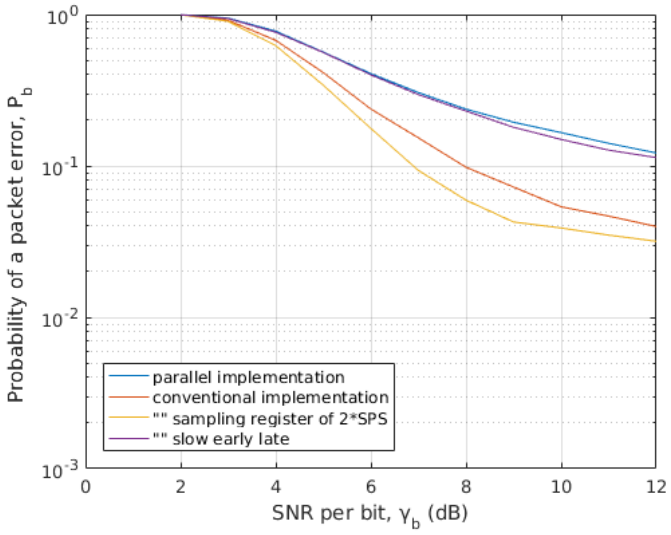

Figure 5.4: PER for different implementations when using a re-sampling rate of 1.0005

5.2 Scalability

Besides the performance of the design the scalability of the design is important. The implementation that was made used 4 demodulation cores with a reduced clock frequency of 2. Is it possible to further decrease the clock frequency with the same design? And is there a linear relation between the number of demodulation cores used in the design and the resources that are used on the FPGA?

The amount of resources used by the demodulation cores scales linear. Every core requires a certain amount of resources which is independent of the number of other cores or the size of the packages send by the switch. The same holds for the sampler block that is used. When there is a early late block for each core the amount of resources for them scales also linear. There is only a little bit of logic required to combine the outputs of these blocks.

The only block which can make the scalability non-linear is the switch block. As the number of cores and the size of the packages grows the size of the switch grows. The switch can be seen as a memory unit. On one side data from the received signal is written to the memory. On the other side data is read by the different cores. The cores read a certain part of the memory with a certain overlap between cores.

One possibility is to use shift registers in the switch. There should be one shift register for the input and one for the output, in that way the demodulation cores can read from the register before the data is overwritten. When there are multiple parallel inputs they should all have their own shift register. The total amount of places in the registers will increase with the package length and the amount of demodulation cores that is used. The total places in the shift registers can be calculated:

$$S = ((N-2)M + 2)SPS \tag{5.1}$$

where N is the length of the packages expressed in symbol periods, M the number of demodulation cores and SPS the number of samples per symbol. In this case it was assumed that the overlap is 2 symbol periods which was the case for the design that was created during this thesis. Apart from a constant part the resources required for this register scale linear. In Figure 5.5 the shift registers that are required for the switch are depicted. Every register place in the output shift register needs a multiplexer to be able to set the data. When the input shift registers all contain new data the information in the input shift registers is used to set the output shift register with the new values. In Figure 5.5 there is only a multiplexer depicted for output register place 0 but in reality every output register place needs such a switch.

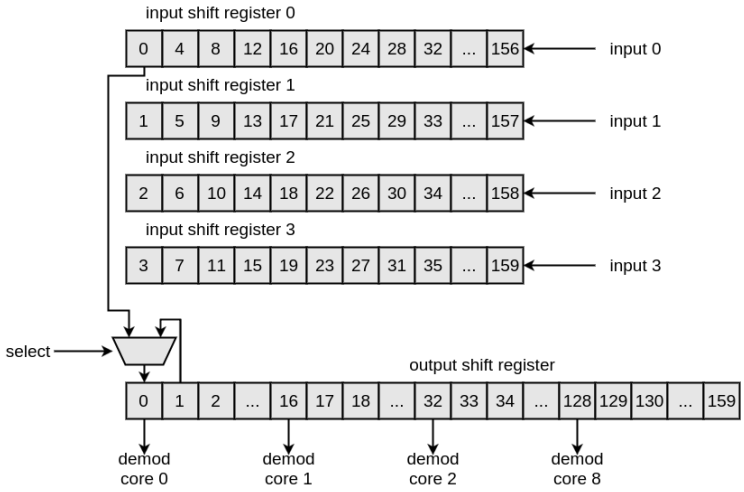

Figure 5.5: Switch design with shift registers

In Table 5.1 the slice registers and Lookup Tables (LUTs) used by the switch for different number of cores are listed. It can be seen that for a doubling of the number of cores the amount of resources is increased with a factor smaller than 2. Which means that the design of the switch is also scalable. The amount of resources required for the switch is a big part of the

FPGA. For the 32 cores design of the switch 22% of the slice registers and 12% of the LUTs are used. Conclusion is that the design is scalable but that the switch will become too big for certain number of cores.

Table 5.1: FPGA resources needed by the switch for different number of cores and a package length of 4 symbol periods

Cores	Slice registers	LUTs
4	9328	5195
8	17551	6188
16	34266	11799
32	67217	18052

Another possibility is to use block RAMs. The total number of required RAMs will then be the number of cores multiplied with the number of parallel inputs of the received signal. When there are for example 8 parallel inputs and 8 demodulation cores, there are at least 64 block RAMs required. For large package lengths it will be efficient to use block RAMs, but for a package length that is relatively small compared to the RAM size it is better to use registers. Otherwise a lot of RAM will be unused. The number of RAMs required will increase quadratically, for that reason it will in most cases be better to use the register file implementation.

In Table 5.2 some estimations are listed for an implementation with 32 parallel paths. For the estimations it is assumed that the blocks are implemented 32 times and that the switch block has a larger register file so that it can send packages to each parallel path. What can be seen is that with the limited resources of the FPGA the number of parallel paths is limited somewhere around 32. Probably it is not possible to implement the design with 32 parallel because also other hardware accelerators are used. The amount of LUTs used reaches 94%. Some improvements can be made to the design, for this estimations no RAMs were used. It is also good to note that 32 parallel paths means an input of 16 parallel samples which is twice the number of the high speed ADC that was available. For this ADC the design will probably fit on the FPGA according to this estimations. When 16 parallel paths are used all resources will be halve of what is listed in the table.

Table 5.2: Estimated resources per block for 32 parallel paths

Block	Slice registers	Slice registers (%)	LUTs	LUTs (%)
Switch	67217	22 %	18052	12 %
Demod	61440	20 %	97344	65 %
Early Late	34266	11 %	11799	8 %
Sampler	13568	5 %	8736	9 %

5.3 Alternative Parallel Structure

Because the hardware resources that are required by the parallel design is increasing fast for multiple cores a better structure is desirable. In Figure 5.6 an alternative parallel design is depicted. In this case four parallel paths are used, which means that the clock frequency is reduced with a factor of four. In Figure 5.6 seven parallel inputs are depicted, these inputs are actually four parallel inputs and three delayed inputs that are used for the filtering. Delay elements should therefore be added to the design.

In the demodulator each sample is multiplied with a different value of the cosine. In the alternative structure these multiplications are done in parallel. The results of these multiplications are filtered by a parallel FIR. This FIR filter sums the samples that arrive in parallel and outputs the result to the output. Another FIR filter performs the filtering with another selection of the input signals. When there are 4 parallel paths then there are also four parallel FIR filters in the structure. Each filtered signal is forwarded to a differential multiplier. The early late sampler determines which of the four samples is the sample that should be used.

This design does not require a switch for switching samples to the multiple cores which will reduce the amount of resources that are required. Another benefit is that there is no invalid data at the outputs and therefore no resources are spilled on invalid data. Due to a lack of time this structure has not been worked out in more detail. In future work this structure can be used to create a parallel structure that uses less resources.

5.4 Summary

In this chapter measurement results with the parallel implementation of the demodulator were presented. These results were compared with the conventional implementation. The results of the PER measurements were worse than the conventional implementation, however some improvements were discussed which could improve the design. When the design was compared with an adapted version of the conventional design the results are almost identically. The scalability of the design was discussed and the conclusion is that the design is scalable. However there will be too few resources on the FPGA when there are too many multiple paths. For the ADC this will most likely work but for future designs some optimisations should be made for the parallel design. The main goal of this thesis was to develop a parallel demodulation structure in FPGA hardware, which has been presented in this chapter.

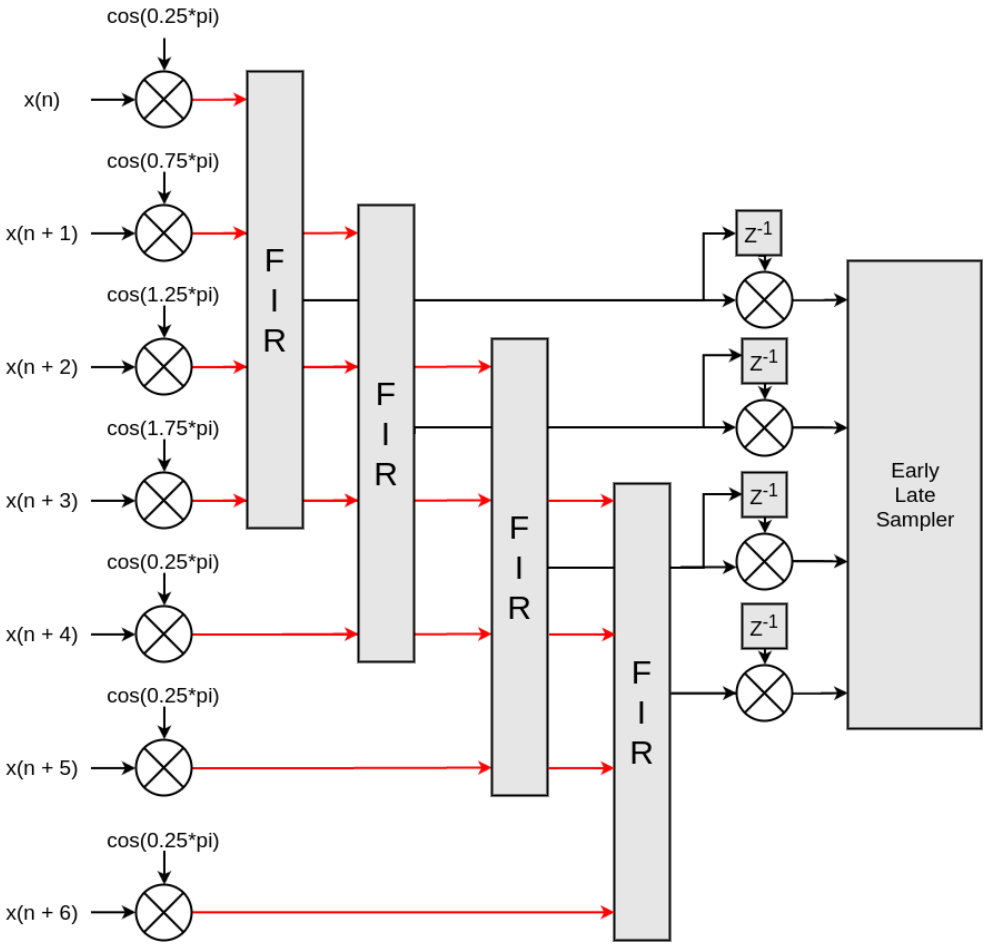

Figure 5.6: Alternative parallel design

CHAPTER 6

Conclusion and Recommendations

This thesis discusses the implementation of a parallel demodulator on a FPGA. The demodulator was compared with a conventional implementation which processes the data sequential.

6.1 Conclusion

The first demodulator that was implemented was a conventional demodulator. This demodulator was compared with a demodulator that was made in GNU-Radio. The implementation that was made on the FPGA has for low SNRs the same performance as the GNURadio implementation. For high SNRs the FPGA implementation has a certain floor, therefore it will never reach a PER of 0 %. Shortcomings of the design were discussed with relation to the sample moment. When the sample moment is reset a bit loss will occur. With extra hardware this could be prevented. It was chosen not to implement this improvement due to a lack of time. The design that was created formed the basis of the parallel design.

After the conventional demodulator was implemented a parallel demodulator has been implemented. The design of it and its shortcomings were discussed. The sampling register that was chosen to implement was actually too small. It was chosen to implement only one symbol time recovery circuit, however from a PER performance perspective it was better to implement one for each parallel path.

Measurements were done to compare the parallel implementation with the conventional one. From the measurements it was concluded that the parallel implementation performs worse than the conventional one, however the results look promising. With some improvements that are discussed in this thesis the design should perform the same as the conventional design. However when the clock difference between the two becomes too high the parallel design will perform worse. The main goal of this thesis, namely designing and implementing a parallel demodulation structure, is achieved.

The design that was presented can be scaled so that different numbers of parallel paths are used. In that way it can be used for different bit-rates and interfaced with ADCs running at higher clock frequencies. However, a lot of resources are used when everything is implemented multiple times. The number of parallel paths is limited by the amount of resources that are available on the used FPGA. For the design that was presented the number of parallel paths is limited to approximately 16.

Although the clock frequency of an FPGA does not keep up with the data rate required by nowadays applications they can still be useful. It has

been shown that it is possible to create a scalable parallel demodulator on an FPGA. These parallel demodulators can be used in combination with high speed ADCs which deliver samples in parallel. In that way it is possible to process the data from the 5GS/s ADC on the FPGA. The parallel structure presented in this thesis can be used in the future to create other parallel demodulation structures.

6.2 Recommendations

The main achievement in this thesis was to create a parallel structure for a demodulator. In this report a few improvements were discussed which probably will improve the performance a lot. In future work this improvements can be applied to the design and the performance can be tested. Another improvement would be to implement the structure depicted in Figure 3.13. This structure uses less hardware than the implementation that was made during this thesis.

Besides that it is also interesting to look at a frequency-domain parallel demodulator. The design that was discussed in this report was based on a time-domain parallel demodulator. It would be interesting to compare the designs with each other. There can also be made improvements in the structure that was presented. Afterwards splitting the input signal in packages was probably not the best solution to create a parallel demodulator. Because the switch that is required will use a lot of space, besides that the demodulator block will take up a lot of space. For future work it could be interesting to investigate other parallel designs principles that can be used for demodulation, for example the one presented in Section 5.3.

In this thesis there was no actual theoretical value for the PER used. It would be derive a theoretical upper bound on the PER for DBPSK. It is interesting to know how good the design is according to this theory.

Due to timing constraints it was chosen not to use the high speed ADC that was available. It would be interesting to interface the design with this ADC. In that way the performance of the system can be tested as it would be in a real high bit-rate receiver application.

List of Figures

2.1 Constellation Diagram of BPSK 7
2.2 Probability of a symbol error for BPSK and DBPSK 7
2.3 Signal modulated using DBPSK (above) and bits that were used (below) 9
2.4 Frequency response of two possible BPSK signals with 16 samples per symbol 9
2.5 Signals showing the bits that are modulated (a), modulated DBPSK signal (b), I component (c) and Q component (d) of the signal after mixing 12
2.6 Signals showing respectively the filtered I and Q component of the signal, differential I and Q signal and the sum of both differential signals. 13

3.1 Schematic of demodulation block 18
3.2 Schematic of demodulation block 18
3.3 Scheme of the sampling shift register (sample moment = 0, SPS = 8) 19
3.4 Scheme of the sampling shift register (sample moment = 7, SPS = 8) 19
3.5 Scheme of the sampling shift register with arrows indicating how the sample moment can change in time 20
3.6 Plot of the signal with the sample moments (arrows) that are used 20
3.7 Plot of the signal with the sample moments (arrows) that are used 21
3.8 Scheme of the bigger sampling shift register with arrows indicating how the sample moment can change in time 22
3.9 Plot of the unfiltered signal with the sample moments (arrows) that are used, the circle indicates a local maximum that is not a symbol 22
3.10 Plot of the filtered signal with the sample moments (arrows) that are used 23
3.11 Eye diagram of the filtered signal 23
3.12 Eye diagram of the unfiltered signal 23
3.13 Schematic of the top level design demodulator using four mixers . 24

3.14 Schematic of the top level design demodulator including early late sampler . . . 24
3.15 Demodulator design in GNURadio . . . 25
3.16 Higher order demodulator block design in GNURadio . . . 26
3.17 Complete signal flow graph at the receiver side . . . 26
3.18 Flow graph of the hardware accelerators that are used . . . 27
3.19 Schematic of the demod block (in red the Xilinx modules) . . . 27
3.20 Schematic of the early late block (in red the Xilinx modules) . . . 28
3.21 Schematic of the complete demodulation block (in red the Xilinx modules) . . . 28
3.22 Schematic of the setup that is used to perform the measurements . 29
3.23 Flow graph that was used in GNURadio to create test signals . . . 30
3.24 PER for the FPGA implementation, GNURadio implementation and the theoretical value assuming no error dependency . . . 31
3.25 Exact sample moment and detected packages over time (SNR per bit = 10 dB) . . . 32
3.26 PER of FPGA implementation for different re-sampling factors . . 33
3.27 Exact sample moment and detected packages over time for a re-sampling factor of 1.00001 (SNR per bit = 10 dB) . . . 33
3.28 PER of FPGA implementation for different re-sampling factors with compensation for packet loss due to resets . . . 34
3.29 Effect of a sampling register that is exactly the length of the symbol period . . . 35
3.30 PER for FPGA implementation with a sample register that is exactly the length of the symbol period with different re-sampling factors. For comparison the PER of the FPGA implementation with a sample register with a length of twice the symbol period is depicted. . . . 35

4.1 Illustration of the functioning of the switch . . . 39
4.2 Illustration of the functioning of the switch when the samples arrival in parallel . . . 39
4.3 Parallel demodulation . . . 40
4.4 Example of the best sample moment within a sampling register with a width of twice the symbol period . . . 41
4.5 Two examples of the best sample moment of a sampling register with a width equal to the symbol period . . . 42
4.6 Parallel demodulation . . . 43
4.7 Illustration of the functioning of the switch block . . . 44
4.8 Demodulation block used in the parallel implementation . . . 44
4.9 Parallel implementation . . . 45

5.1 PER of parallel implementation for different re-sampling factors, for comparison the PER of the conventional implementation in GNURadio is also depicted. 48
5.2 PER for different implementations 49
5.3 Sample moment and total number of detected packages over time for the parallel implementation 49
5.4 PER for different implementations when using a re-sampling rate of 1.0005 50
5.5 Switch design with shift registers 51
5.6 Alternative parallel design 54

Bibliography

[1] 4DSP. *FMC125 high pin count FMC ADC*, 2015.

[2] Jan Crols and Michiel S. J. Steyaert. *Low-IF topologies for high-performance analog front ends of fully integrated receivers.* In *IEEE transactions on circuits and systems II*, March 1998.

[3] F.M. Gardner. *A BPSK/QPSK timing error detector for sampled receivers.* In *IEEE transactions on communications*, May 1986.

[4] Changxing Lin, Beibei Shao, and Jian Zhang. *A high data rate parallel demodulator suited to FPGA implementation.* In *international symposium on intelligent signal processing and communication systems*, December 2010.

[5] Louis Litwin. *Matched filtering and timing recovery in digital receivers*, September 2001. Online.

[6] K.H. Mueller and M. Müller. *Timing recovery in digital synchronous data receivers.* In *IEEE transactions on communications*, May 1976.

[7] Martin Oerder and Heinrich Meyr. *Digital filter and square timing recovery.* In *IEEE transactions on communications*, May 1988.

[8] J.G. Proakis and M. Salehi. *Digital communications, 4th edition (p. 274-278)*, 2000.

[9] Donald Rasmussen and George Davis. *Serial and parallel demodulation trade-offs for QBL-MSK.* In *tactical communications conference*, May 1994.

[10] Ettus Research. *USRP^{TM} N200/N210 networked series datasheet*, 2012.

[11] James A. Roberts. *Packet error rates for DPSK and differentially encoded coherent BPSK.* In *IEEE transactions on communications*, February 1994.

[12] Epiq Solutions. *Bitshark FMC-1RX broadband configurable RF receiver*, 2010.

[13] M. Srinivasan, C.C. Chen, G. Grebowsky, and A. Gray. *An all-digital, high data-rate parallel receiver.* In *international conference on signal processing applications and technology*, 1997.

[14] Xilinx. *IP LogiCORE FIR compiler v5.0*, 2011.

[15] Xilinx. *LogiCORE IP adder/subtracter v11.0*, 2011.

[16] Xilinx. *LogiCORE IP multiplier v11.2*, 2011.

[17] Xilinx. *LogiCORE IP RAM-based shift register v11.0*, 2011.

[18] Xilinx. *LogiCORE IP block memory generator v7.3*, 2012.

[19] Xilinx. *ML605 hardware user guide*, 2012.

[20] Ronghua Zhou and Shuanghuan Li. *A low complexity frequency-domain parallel demodulation structure combining matched filter with timing synchronization.* In *information and communications technologies*, April 2013.