Master of Science Thesis in Communication systems
Department of Electrical Engineering, Linköping University, 2016

Implementation of an SDR in Verilog

Anders Skärpe

LINKÖPING UNIVERSITY

Master of Science Thesis in Communication systems

Implementation of an SDR in Verilog

Anders Skärpe

LiTH-ISY-EX–16/5001–SE

Supervisor: **Kent Palmkvist**
ISY, Linköping University
Daniel Nordgren
Syntronic

Examiner: **Håkan Johansson**
ISY, Linköping University

Division of Communication systems
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

This report presents an implementation of the software part in a software defined radio. The radio is not entirely implemented in software and therefore there are certain limitations on the received signal. The parts implemented are oscillator, decimation filter, carrier synchronization, time synchronization, package detection, and demodulation. Different algorithms were tested for the different parts to measure the power consumption. To understand how the number of bits used to represent the signal affects the power consumption, the number of bits was reduced from 20 bits to 10 bits. This reduction reduced the power consumption from 2.57mW to 1.89mW. A small change in the choice of algorithms was then made which reduced the power consumption to 1.86mW. Then the clock rate was reduced for some parts of the system which reduced the power consumption to 1.05mW.

Acknowledgments

I would first like to express my gratitude to the people at Syntronic for giving me the opportunity to make this thesis possible. I would also like to thank my supervisors at ISY, Håkan Johansson and Kent Palmkvist, for discussing ideas and providing feedback. I would also like to thank Altera for providing a license for DSP Builder and Quartus Prime which was very helpful when implementing the system.

Contents

Notation ix

List of Figures 1

List of Tables 3

1 Introduction 5
- 1.1 Problem statement . . . 5
- 1.2 Delimitations . . . 6
- 1.3 Thesis outline . . . 6

2 Theory 9
- 2.1 Radio . . . 9
  - 2.1.1 Software Defined Radio . . . 9
- 2.2 Distortions . . . 10
  - 2.2.1 Additive White Gaussian Noise . . . 10
  - 2.2.2 Inter Symbol Interference . . . 11
  - 2.2.3 Phase and Frequency Offset . . . 11
  - 2.2.4 Time Delay . . . 11
- 2.3 Measurement . . . 12
  - 2.3.1 Bit Error Rate . . . 12
  - 2.3.2 Signal Noise Ratio . . . 12
  - 2.3.3 Eye Diagram . . . 12
- 2.4 Receiver . . . 13
- 2.5 Digital Decimation Filter . . . 15
  - 2.5.1 Polyphase Filter . . . 15
  - 2.5.2 Multistage Filter . . . 16
  - 2.5.3 Square Root Raised Cosine Filter . . . 17
  - 2.5.4 Matched Filter . . . 17
- 2.6 Synchronization . . . 18
  - 2.6.1 Carrier Synchronization . . . 18
  - 2.6.2 Symbol Timing . . . 21
  - 2.6.3 Packet Detection . . . 24

2.7 Modulation and Demodulation 24
2.7.1 Quadrate Phase Shift Keying 24
2.8 Power Optimization 25
2.9 Orthogonal Frequency Division Multiplexing 27
2.10 Frequency Hopping Spread Spectrum 28

3 Method **29**
3.1 Tools 29
3.2 Simulink Reference Model 30
3.3 DSP Builder Reference Model 30
3.4 Transmitter 30
3.5 Receiver 31
3.5.1 Decimation Filter 32
3.5.2 Carrier Synchronization 33
3.5.3 Time Synchronization 35
3.5.4 Package Detection 37
3.6 Power Optimization 39

4 Result **41**
4.1 Carrier Synchronization 42
4.2 Timing Synchronization 43
4.3 Package Detection 44
4.4 Bit Reduction 46
4.5 Clock Rate Reduction 47
4.6 Resource Allocation 47

5 Discussion **49**
5.1 Result 49
5.2 Method 51

6 Conclusion **53**
6.1 Discussion 53
6.2 Future Work 54
6.2.1 Unimplemented work 54
6.2.2 Lower Power Consumption 54

Bibliography **55**

A Detailed Descriptions of the systems **59**

Notation

Abbreviations

Abbreviation	Meaning
ADC	Analog to Digital Converter
AGC	Automatic Gain Control
ALUT	Adaptive Lookup Table
ASK	Amplitude Shift Keying
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
dB	Desibel
DD	Decision Direct
DSP	Digital Signal Processing
F-FH	Fast Frequency Hopping
FHSS	Frequency Hopping Spread Spectrum
FPGA	Field-Programmable Gate Array
FSK	Frequency Shift Keying
HDL	Hardware Description Language
I	In-phase
ICI	Inter Carrier Interference
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
ISI	Inter Symbol Interference
LSB	Least Significant Bit
ML	Maximum Likelihood
MSK	Minimum Shift Keying
NCO	Numerically Controlled Oscillator
NDA	Non Data Aided
OFDM	Orthogonal Frequency Division Multiplexing
OSI	Open System Interconnection
PSK	Phase Shift Keying
Q	Quadrature
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
S-FH	Slow Frequency Hopping
SDR	Software Defined Radio
SNR	Signal to Noise Ratio
SRRC	Square Root Raised Cosine
TED	Timing Error Detector

List of Figures

2.1 Transmitter, Channel and Receiver. 10
2.2 ISI, the black lines are the sample instant. 11
2.3 (a) Eye with a large amount of ISI (b) Eye with a small amount of ISI. 13
2.4 (a) Constellation with a large amount of ISI (b) Constellation with a small amount of ISI. 14
2.5 Overview of the receiver. 14
2.6 Decimation filter transformed to a polyphase decimation filter. . . 16
2.7 Luise algorithm with M=3 and N=512. 20
2.8 Costas loop. 20
2.9 DD QPSK costas loop. 21
2.10 A multirate time synchronization. 22
2.11 QPSK modulation. 25
2.12 BER versus SNR for QPSK. 26
2.13 QPSK constellation diagram. 27

3.1 Overview of the transmitter. 31
3.2 The frequency spectrum around 2 MHz. 31
3.3 QPSK Costas implemented with XOR and multiplexer. 35
3.4 Simple overview of the carrier synchronization. 36
3.5 A multirate time synchronizaiton. 37
3.6 Implementation of Gardner's algorithm. 38
3.7 Linn's algorithm with one input lookup table. 38
3.8 Calculation before the matched filter. 39

4.1 (a) Eye diagram before synchronization. (b) Eye diagram after synchronization. 43
4.2 Constellation plot comparing (a) before carrier synchronization (b) after synchronization. 44
4.3 Constellation plot after time synchronization (a) Gardner (b) Linn. 45
4.4 Magnitude comparison. 45
4.5 Graph over power consumption decrease with bit reduction. . . . 46
4.6 Graph over SNR when reducing the number of bits. 47

A.1 An overview how the subsystems are connected. 60

A.2 Overview of the whole receiver. 61
A.3 Overview of carrier synchronization. 62
A.4 The time synchronization. 63
A.5 Overview of the matched filter and package detection. 64
A.6 The package detection. 65

List of Tables

3.1 Comparing QPSK implemented with multipliers or multiplexers. . 35

4.1 The power consumption of the oscillator and filter. 41
4.2 The subsystems implemented in the final system. 42
4.3 Comparison in power consumption between Luise implementations. 42
4.4 Comparison between fine carrier synchronization power consumption. 43
4.5 Comparison between TEDs power consumption. 44
4.6 Comparison between magnitude calculation power consumption. 46
4.7 Comparison between power consumption at different clock rates. . 47
4.8 Resource allocation of the different subsystems. 48

1 Introduction

This chapter starts with outlining the problem for this thesis and its delimitations. At the end it goes through the structure of the report and what each chapter contains.

1.1 Problem statement

Syntronic is developing a radio receiver where a part of the radio shall be implemented in software. The purpose of the software defined radio (SDR) is to be used out in the field; therefore it has to have a low power consumption and good noise reduction. For the radio to be able to receive more data, orthogonal frequency division multiplexing (OFDM) shall be supported. To hinder the chance of jamming, frequency hopping spread spectrum (FHSS) shall also be supported.

SDR is a relatively new technology, first coined by Mitola J. in 1993 [1], and since then extensive research has been carried out on the subject. The goal of SDR is to be able to update the radio with unforeseen changes without changing the hardware. These changes can be the desired frequency, bandwidth, modulation or any other part of the software [2].

OFDM is a technique to split a high-rate datastream into lower rate datastreams. These lower rate datastreams are then transmitted simultaneously with different carriers [3].

FHSS means that the carrier frequency for the transmitted signal changes rapidly according to a predefined pattern [3].

In this thesis only the software part of the radio without OFDM and FHSS will be implemented but the report will cover the theory behind OFDM and FHSS.

The question the thesis will strive to answer is:

How can optimization in software and hardware reduce the energy consumption of a software defined radio receiver while still maintaining a bit error rate (BER) at 10^{-6}*?*

1.2 Delimitations

Only a part of the SDR will be implemented in this thesis, more precisely the software part. The software for the SDR will consist of the following systems:

- Oscillator
- Decimation filter
- Synchronization
- Demodulation

The software will receive a signal from an analog-to-digital converter (ADC). The signal received will have the following specifications:

- Bandwidth is 1MHz
- Bandwidth of the signal that shall be retrieved is 25kHz
- Center frequency is 2MHz
- Modulation with quadrature phase shift keying (QPSK)

Further the sampling frequency of the ADC is 5MHz.

When transmitting the signal the channel will distort the signal with the following distortions:

- Additive white gaussian noise (AWGN)
- Phase and frequency shift
- Time delay

The signal can also be distorted by other 25kHz signals sent in the proximity of the signal that shall be retrieved.

The final restriction is that the system will only be implemented and optimized for Cyclone V, a field-progammable gate array (FPGA).

1.3 Thesis outline

- Chapter 2, Theory: This chapter gives a theoretical overview of the problem and explain different solutions to the sub problems.

- Chapter 3, Method: This chapter describes how the different problems were solved. It first explains how the transmitter and receiver were set up and then goes through how the power consumption was reduced.
- Chapter 4, Result: In this chapter the results from the different methods used to reduce the power consumption are presented.
- Chapter 5, Discussion: This chapter discusses the results gained and the methodology used to implement the system.
- Chapter 6, Conclusion: This chapter discusses the conclusions that can be drawn from the results gained. After the discussion future work for the system is presented.

2
Theory

This chapter presents a theoretical overview of the problem. The chapter starts with defining what a radio and an SDR are. Then the different distortions that will affect the signal and the different measurements are presented.

Then the purpose of the different subsystems and different solutions to these subsystems are presented. Finally the theory behind OFDM and FHSS is presented.

2.1 Radio

To understand what an SDR is, the first step is to define what a radio is. IEEE and the wireless innovation forum (formerly SDR forum) defines a radio as one of the following [4]:

(a) Technology for wirelessly transmitting or receiving electromagnetic radiation to facilitate transfer of information.

(b) System or device incorporating technology as defined in (a).

(c) A general term applied to the use of radio waves.

In this thesis the first definition (a) is the most applicable definition of the radio.

Figure 2.1 displays a simple overview of a transmitter sending information over a channel to a receiver.

2.1.1 Software Defined Radio

The definition for an SDR by the IEEE and wireless innovation forum states [5]:

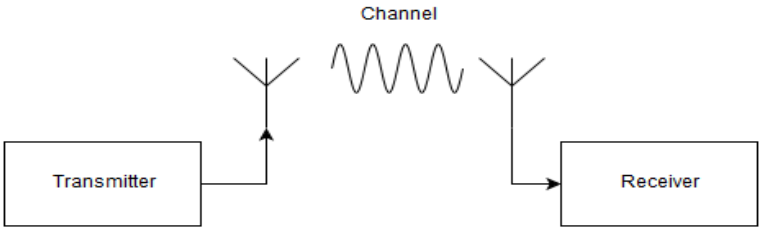

Figure 2.1: *Transmitter, Channel and Receiver.*

A radio in which some or all the physical layer functions are software defined.

The physical layer is one of the seven different layers defined by the Open System Interconnection (OSI) model [6]. Each layer in the model is a small part of a communication system where a layer provides services to the layer above and receives services from the layer below it. The physical layer is the lowest of the seven layers [4].

By this definition many radios are not software defined but rather software controlled. A software controlled radio is a radio that for example has different modulation blocks and the software can switch between these. The difference between software controlled and software defined is not entirely black and white, depending on how much the software can control the hardware the software controlled radio will eventually be a software defined radio instead [4].

To draw a line between software defined and software controlled it comes down to that the software controlled functionality is limited by its design, whereas the software defined may be reprogrammed for new functionality [4].

2.2 Distortions

Distortion of the signal can occur in many ways and in different places when transmitting and receiving the signal. When preparing to transmit the signal the signal can be distorted by inter symbol interference (ISI). While transmitting the signal through a channel the signal can be exposed to different kind of noise sources, i.e. AWGN. When receiving the signal the clocks at the receiver and transmitter might be unsynchronized which causes phase and frequency offset, and time delay.

2.2.1 Additive White Gaussian Noise

AWGN is used to simulate different kinds of additive white noises. Additive noise means that the noise is added to the transmitted signal, i.e $y(t) = x(t) + w(t)$ is the observed signal, where $x(t)$ is the signal and $w(t)$ is the noise. A signal is said to be white if the random variables in the signal are uncorrelated and have the same statistical distribution [7].

AWGN can for example simulate thermal noise or radiation noise. Thermal noise is caused by random movements of electrons in the transmitter and receiver. Radiation noise is caused by the atmospheric and earth blackbody electromagnetic radiation [7].

In electric engineering the noise is often assumed to have a gaussian (normal) distribution with zero mean. The noise is assumed to have zero mean because otherwise the mean could just be subtracted [7].

2.2.2 Inter Symbol Interference

If bits were to be ideally transmitted through a channel they would be represented as rectangular pulses and these pulses would have an infinite bandwidth. Because it is not possible to transmit with an infinite bandwidth the bandwidth has to be finite. To transmit these pulses over a channel with finite bandwidth some distortion has to be tolerated. This distortion could cause the pulses to overlap in time and then the sampling instant at the receiver will not depend only on the current symbol pulse but on one or more preceding pulses. When the sampling instant depends on more than the current symbol pulse it is called ISI and is illustrated in Fig. 2.2. There it can be seen that the sample will not depend only on one symbol but other nearby symbols as well [8].

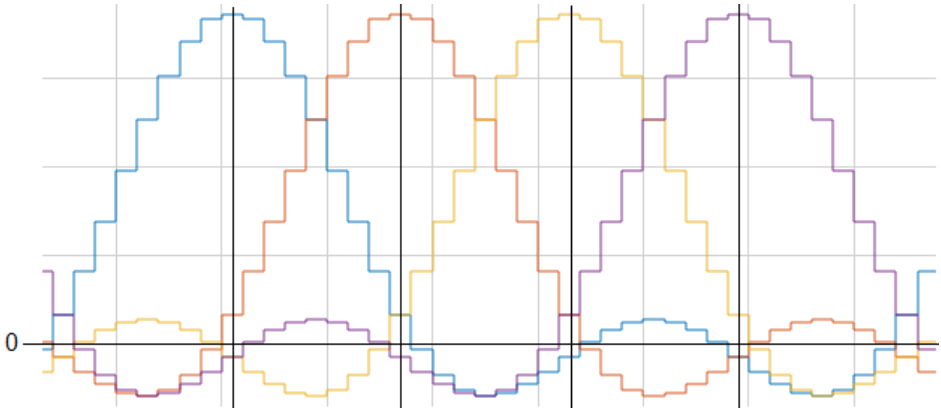

Figure 2.2: *ISI, the black lines are the sample instant.*

2.2.3 Phase and Frequency Offset

Frequency offset can occur because the oscillators in the transmitter and receiver drift apart and therefore do not have exactly the same frequency. Another reason for frequency offset is when the transmitter or receiver is in motion, then the frequency is subjected to a Doppler shift [7].

2.2.4 Time Delay

When a signal is converted from analog to digital it has to be sampled by an ADC at the correct time to gain a good result at the receiver [9]. Because the clocks at

the transmitter and receiver are asynchronous and the transmission time varies the receiver has to synchronize its clock to the received signal [7].

2.3 Measurement

To evaluate how the different algorithms compare to each other it is necessary to measure more parameters than the power consumption. Other than the power consumption the BER and signal to noise ratio (SNR) will be measured. To inspect how well the synchronizations are executing, eye diagrams will be used.

2.3.1 Bit Error Rate

Bit error rate measures how often a bit is retrieved incorrectly at the receiver. The BER can be measured at different stages in the receiver. In this implementation the BER will be measured directly after the signal has been demodulated.

2.3.2 Signal Noise Ratio

If the signal received is $y(t) = x(t) + w(t)$ where $x(t)$ is the signal sent and $w(t)$ is white noise then $y(t)$ can be sampled to yield

$$y[n] = x[n] + w[n], n = 1, 2, ..., 2BT \tag{2.1}$$

where B is the bandwidth and T is the time the signal is transmitted. SNR is then defined, in decibel (dB), as

$$SNR = 10 \log_{10} \frac{E(x[n]^2)}{E(w[n]^2)} [dB] \tag{2.2}$$

where $E(x)$ is the expected value of x. If $x(t)$ is modeled as a random signal then, in many cases, its samples $x[n]$ have the same distribution. It is then possible generate many signals and calculate a mean value by only using one sample from each signal to calculate the SNR which gives [7]

$$SNR = 10 \log_{10} \frac{E[x[1]^2]}{E[w[1]^2]} [dB] \tag{2.3}$$

2.3.3 Eye Diagram

The amount of ISI and the noise intensity can be displayed in an eye diagram. Figure 2.3(a) displays an eye diagram with a large amount of ISI and Fig. 2.3(b) displays an eye diagram with a small amount of ISI. As can be seen, the eye closes in the presence of a large amount of ISI. The eye also closes when the signal contains noise. It is also possible to see how sensitive the signal is to timing offset in the sampling. The optimal sample time is when the eye is most vertically open

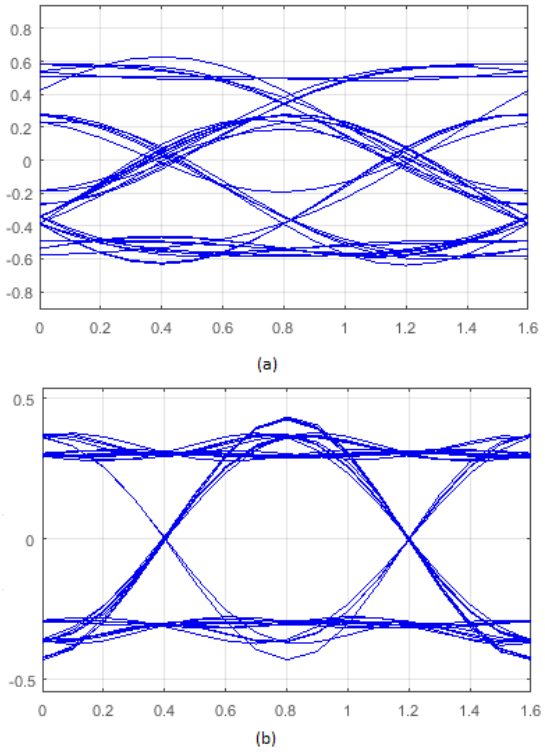

Figure 2.3: *(a) Eye with a large amount of ISI (b) Eye with a small amount of ISI.*

and the sensitivity in sample time depends on how wide the eye is. A narrow eye means that there is less time to be certain that the correct value is sampled [9].

When receiving a signal it can also be useful to plot a constellation diagram. The constellation diagram displays the sampled values and without noise and ISI it should display a number of distinct points. The number of points depends on the chosen modulation, for QPSK the number of points is four. Figure 2.4(a) displays a constellation diagram with a large amount of ISI and Fig. 2.4(b) displays a constellation diagram with a small amount of ISI.

2.4 Receiver

This section will explain the different subsystems that will be implemented in the receiver. Figure 2.5 displays an overview of the receiver and how these systems will be connected. The subsystems inside the black border is implemented in this thesis.

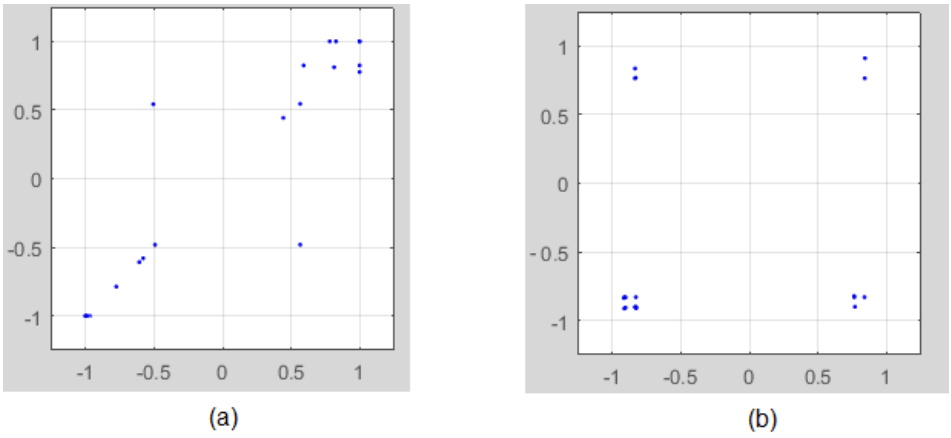

Figure 2.4: *(a) Constellation with a large amount of ISI (b) Constellation with a small amount of ISI.*

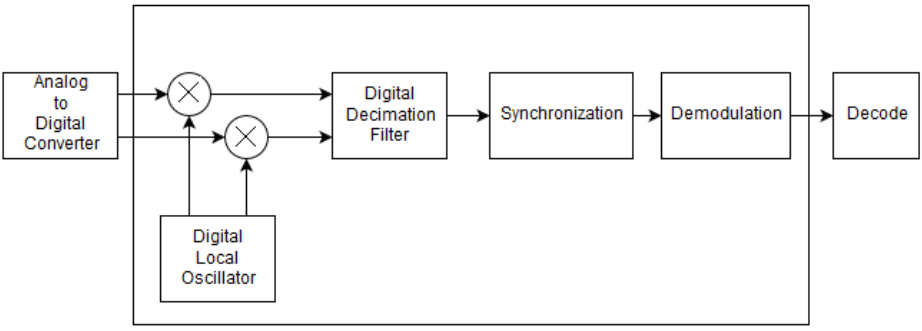

Figure 2.5: *Overview of the receiver.*

2.5 Digital Decimation Filter

A decimation filter is beneficial at the receiver when the ADC samples the signal at a higher frequency than the frequency of the signal. When multiplying the received signal with the oscillators at the receiver, unwanted signals are produced. To remove these unwanted signals a lowpass filter is required.

Although the information can be retrieved from an oversampled signal it requires more calculations and therefore more power. The benefits from having more samples is that better synchronizations and estimates of the received symbol can be carried out.

2.5.1 Polyphase Filter

A polyphase filter reduces the workload by exploiting the fact that a number of samples will be thrown away when decimating the signal. This means that the number of operations per second is reduced by a factor of M, where M is the decimation factor, but the number of operations required for one sample is the same as a straightforward solution [10].

To create a polyphase filter it is beneficial to first obtain the polyphase representation of the impulse response and/or transfer function. In polyphase representation the signal $h(n)$ is represented as a sum of M partial signals. To obtain $h(n)$ first M signals have to be created as:

$$h_i(n) = h(nM + i), i = 0, 1, 2, ..., M - 1 \tag{2.4}$$

Then $h_i(n)$ is upsampled by M to create M new signals:

$$h_i^{(M)}(n) = \begin{Bmatrix} h_i\left(\frac{n}{M}\right), & \text{if } n = 0, \pm M, \pm 2M, ... \\ 0, & \text{otherwise} \end{Bmatrix} \tag{2.5}$$

Finally $h(n)$ is obtained as the sum of shifted versions of $h_i^{(M)}(n)$:

$$h(n) = \sum_{i=0}^{M-1} h_i^{(M)}(n - i) \tag{2.6}$$

To represent the polyphase form for the z-transform first the z-transformation of the signal $h(n)$ is required:

$$H(z) = \sum_{n=0}^{\infty} h(n)z^{-n} \tag{2.7}$$

which can be rewritten as:

$$H(z) = \sum_{n=0}^{\infty}\sum_{i=0}^{M-1} h_i^{(M)}(n-i)z^{-n} = \sum_{i=0}^{M-1}\sum_{n=0}^{\infty} h_i^{(M)}(n-i)z^{-n} =$$
$$\sum_{i=0}^{M-1} z^{-i}\sum_{n=0}^{\infty} h_i^{(M)}(n)z^{-n} = \sum_{i=0}^{M-1} z^{-i}H_i^{(M)}(z) = \sum_{i=0}^{M-1} z^{-i}H_i(z^M) \tag{2.8}$$

The polyphase representation can then be used to create a polyphase filter. Each $H_i(z^M)$ represents a filter and z^{-i} represents the delay before each filter. Figure 2.6 displays how an M-fold decimation filter is converted to a polyphase decimation filter [10].

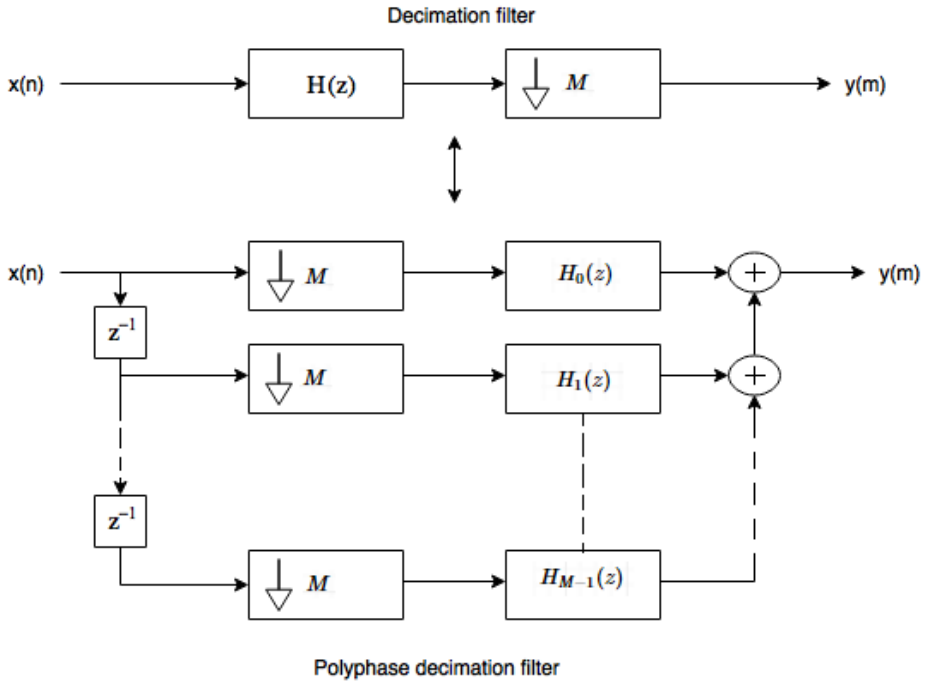

Figure 2.6: *Decimation filter transformed to a polyphase decimation filter.*

2.5.2 Multistage Filter

It is often beneficial to implement the decimation filter in multiple stages instead of a single stage. Each stage reduces the sampling rate by a fraction of the entire reduction. By reducing the sampling rate in multiple stages it is possible to relax the requirements on the filters, which can lead to a reduced overall complexity [10].

For example, in reference [11] the complexity of a single stage filter is compared to different multistage solutions. There a one stage decimation filter of conversion rate 20 is compared to a two stage decimation filter with conversion rates 5 and 4. The number of multiplications is reduced from 42 multiplications in the one stage design to 13 multiplications in the two stage design [11].

2.5.3 Square Root Raised Cosine Filter

When transmitting a signal it will go through the transmitter filter $H_T(f)$, the channel $C(f)$, and last the receiver filter $H_R(f)$. When transmitting a signal these three becomes cascaded, $H_T(f)C(f)H_R(f)$, and they should be designed to yield zero ISI [9].

Square root raised cosine (SRRC) are a specific type of raised-cosine filter where the transmitter and receiver filter are chosen to satisfy

$$H_T(f) = H_R(f) = \sqrt{H_{RC}(f)} \tag{2.9}$$

where $H_{RC}(f)$ is a raised cosine frequency response given by

$$H_{RC}(f) = \begin{Bmatrix} T_s, & 0 \le |f| \le \frac{1-\alpha}{2T_s} \\ \frac{T_s}{2}\left(1 - \sin\left[\frac{\pi T_s}{\alpha}\left(|f| - \frac{1}{2T_s}\right)\right]\right), & \frac{1-\alpha}{2T_s} \le |f| \le \frac{1+\alpha}{2T_s} \\ 0, & \text{otherwise} \end{Bmatrix} \tag{2.10}$$

$1/T_s$ is the symbol rate and α is the rolloff factor limited to $0 \le \alpha \le 1$. The greater the value of alpha is the larger excess bandwidth the filter will have [9].

A signal without excess bandwidth is confined to the frequency band $\left[-\frac{1}{2T_s}, \frac{1}{2T_s}\right]$ but with the excess bandwidth it becomes confined to $\left[-\frac{1}{2T_s} - \epsilon, \frac{1}{2T_s} + \epsilon\right]$, where T_s is the sample interval and ϵ is the excess bandwidth. With this extra bandwidth it is possible to choose the pulse shape and receiver filter more freely [7].

2.5.4 Matched Filter

A matched filter is used to maximize the SNR after the filter but the matched filter does not take ISI into account. Therefore there will be ISI and noise present even when using an ideal SRRC transmitter and receiver filter [9].

To maximize the SNR, the receiver filter must satisfy $H_R(f) = H_{TC}^*(f)$ where $H_{TC}(f) = H_T(f)C(f)$, $H_T(f)$ is the response from the transmitter and $C(f)$ is the response from the channel [9].

2.6 Synchronization

The synchronizations required in a radio can vary depending on how the radio shall be implemented. In the radio implemented in this thesis the following synchronizations are required:

- Carrier synchronization
- Symbol synchronization
- Packet synchronization

2.6.1 Carrier Synchronization

Carrier synchronization is required when the signal can be distorted by phase and/or frequency shift, or if the oscillators in the transmitter and receiver drift apart in frequency. For carrier synchronization there are two basic approaches used. The first is to use a preamble that the receiver can use to synchronize the phase and frequency. The second is to synchronize the carrier with only the modulated signal [12].

Carrier synchronization is often split up in two categories: coarse carrier synchronization and fine carrier synchronization. The coarse carrier synchronization synchronizes at intervals which makes drifting clocks a problem when only performing a coarse carrier synchronization. The fine synchronization synchronizes continuously but can only synchronize small differences. By doing a coarse carrier synchronization followed by a fine carrier synchronization the large error will be reduced to a manageable error for the fine carrier synchronization to correct.

Coarse Frequency Synchronization

The signal will be modulated by QPSK which is an M-phase modulation were M=4, some other M-phase modulation techniques are minimum shift keying (MSK) and quadrature amplitude modulation (QAM). After an M-phased modulated signal has been sampled it can be represented as [9]:

$$s[n] = A[n]\cos\left[2\pi\frac{f_c}{F_s}n + \theta + \frac{2\pi}{M}(m-1)\right] \tag{2.11}$$

where m=1,2,...,M, f_c is the carrier frequency, F_s is the sampling rate, and $\frac{2\pi}{M}(m-1)$ is the information bearing component of the signal phase. By using a power of M carrier recovery the goal is to remove the information bearing component and estimate the error with the unmodulated carrier $\cos\left(2\pi\frac{f_c}{F_s}n + \theta\right)$. By raising $s[n]$ by a power of M, $(s[n])^M$ will be received. This yields

$$(s[n])^M = \left[\cos\left(2\pi\frac{f_c}{F_s}n + \theta + \frac{2\pi}{M}(m-1)\right)\right]^M = \cos\left(2\pi M\frac{f_c}{F_s}n + M\theta\right) \tag{2.12}$$

The term

$$\frac{2\pi}{M}(m-1)M = 2\pi(m-1) \tag{2.13}$$

loses all its information and can therefore be excluded [9].

Luise algorithm is a maximum likelihood (ML) estimation presented in [13]. There exist many other algorithms for estimating the coarse frequency, a few of these are presented in [14]. They are similar to Luise algorithm in implementation but requires more computations because the maximum of different summations has to be found. Therefore Luise algorithm is implemented to find out the performance and power consumption of the algorithm.

To estimate the frequency with ML the maximum of the equivalent function has to be found

$$\Lambda(\Delta\tilde{f}) = \left|\sum_{i=1}^{N} y_i e^{-j2\pi\Delta\tilde{f}iT_s}\right|^2 = \sum_{k=1}^{N}\sum_{m=1}^{N} y_k y_m^* e^{-j2\pi\Delta\tilde{f}T_s(k-m)} \tag{2.14}$$

where

$$y_k = e^{j(2\pi\Delta f k T_s+\theta)} + w_k, 1 \le k \le N \tag{2.15}$$

is the observed signal, T_s is the sample time, θ is the phase error, w is gaussian noise, and N is the number of samples. The problem with finding the maximum is that it is a time consuming and complex task [13].

The calculation for estimating the error in Luise algorithm is

$$\Delta\hat{f} = \frac{1}{\pi T_s(M+1)}\arg\left(\sum_{k=1}^{M} R(k)\right) \tag{2.16}$$

where $R(k)$ is defined as

$$R(k) = \frac{1}{N-k}\sum_{i=k+1}^{N} y_i y_{i-k}^*, 0 \le k \le N-1 \tag{2.17}$$

Figure 2.7 displays an implementation of Luise algorithm [13].

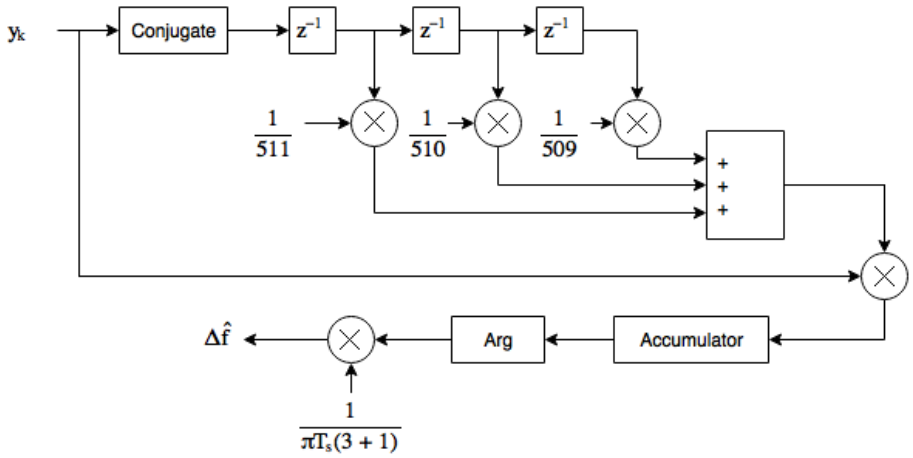

Figure 2.7: *Luise algorithm with M=3 and N=512.*

Fine Carrier Synchronization

Costas loop is a common fine carrier synchronization algorithm and there exist a few variations of it. In [15], two other fine carrier synchronization algorithms are presented. These synchronizations utilize lookup tables to minimize the computations. Although the use of lookup tables reduces the number of calculations there are still more calculations in the algorithms presented in [15] compared to Costas loop. Therefore, to reduce the number of computations, Costas loop will be studied later on. See Fig. 2.8 for the original Costas loop.

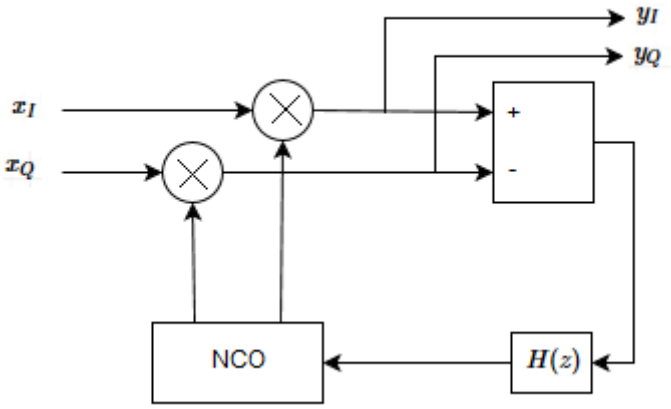

Figure 2.8: *Costas loop.*

A decision direct (DD) variation is presented in [16] and displayed in Fig. 2.9,

this variation will be called DD QPSK Costas loop. DD means that the detector takes a decision on how the signal should be interpreted at the demodulation.

This variation performs both carrier synchronization and timing synchronization. This works if the in-phase (I) and quadrate (Q) signals are near their correct values, then the sign will ignore small deviations caused by noise or not perfect synchronization. The ±1 symbol decision estimates the carrier error and feeds the estimation through the filter $H(z)$ to a numerically controlled oscillator (NCO) [16].

By removing the sign from the signal that is sent forward in the system the carrier synchronization will no longer be DD and then a timing synchronization can be performed after, this variation will be called QPSK Costas loop. The sign will only feed its signal forward to the multiplications to produce a carrier error estimation.

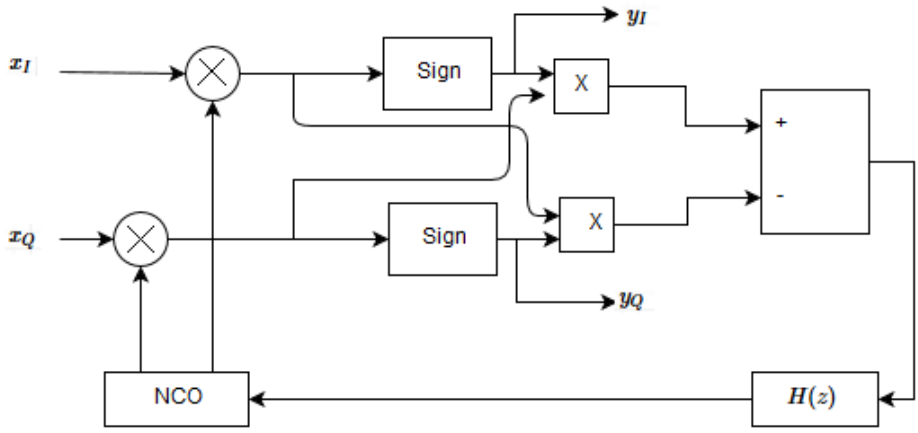

Figure 2.9: *DD QPSK costas loop.*

2.6.2 Symbol Timing

Symbol synchronization is usually required for the ADC to sample with the correct timing. This can be accomplished if either the symbol synchronization is performed in the analog part before the ADC or if the software can send a signal to the ADC to change the timing.

Another solution, called multirate timing recovery, is possible if the ADC samples at a frequency at least twice as high as the frequency of the received signal. Then

the synchronization can be performed in the software without changing when the ADC samples [9].

Figure 2.10 displays a multirate timing recovery. The signal is interpolated based on the value μ. The interpolation calculates an intermediate sample between two samples and μ affects were this intermediate sample is located. To calculate μ a timing error detector (TED) is first used to estimate the error. The estimation is then filtered and passed to an NCO [9].

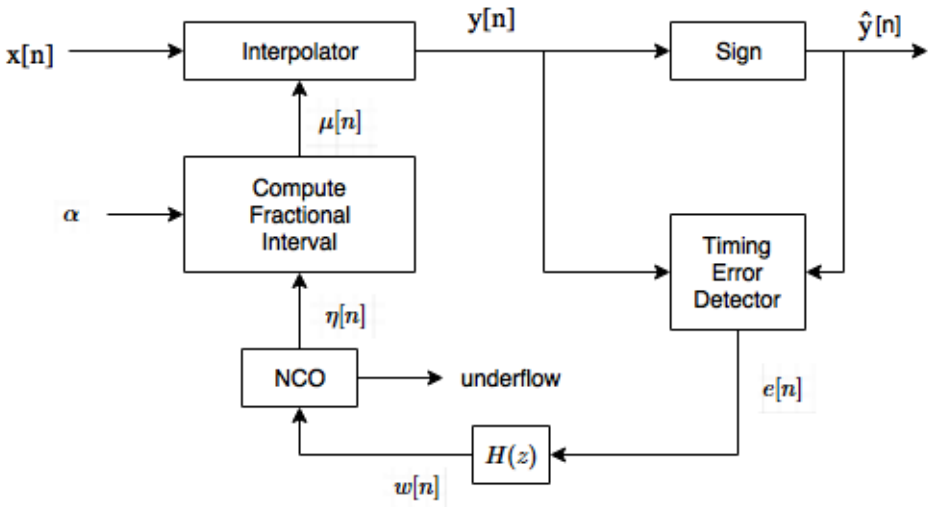

Figure 2.10: *A multirate time synchronization.*

The calculations for the multirate timing recovery is presented in [17]. The calculation for the NCO is

$$\eta[n] = (\eta[n-1] - w[n-1]) \mod 1 \tag{2.18}$$

The modulo 1 is because $\eta[n]$ shall be positive and only the fractional part shall remain. The NCO receives $w[n]$ from the filters which should be nearly constant. The NCO will on average underflow each $1/w[n]$ clock cycles. This gives the NCO a period of $T_i = T_s/w[n]$ which gives

$$w[n] \approx \frac{T_s}{T_i} \tag{2.19}$$

where T_s is the sample time before the interpolation and T_i is the sample time after. The value from the NCO is then used to calculate

$$\mu[n] = \frac{\eta[n]}{w[n]} \tag{2.20}$$

To avoid division it is possible to calculate

$$\alpha = \frac{T_s}{T_i} \approx 1/w[n] \tag{2.21}$$

Although the exact value of $1/w[n]$ is unknown, as seen in equation (2.19) it can be approximated to a know value [17]. With α, μ can be approximated by

$$\mu[n] \approx \alpha\eta[n] \tag{2.22}$$

To remove any timing jitter that can occur, the underflow of the NCO provides a timing clock. When the NCO underflows it indicates correct data clocking [17].

With a working interpolation it is the TED that will affect the result gained from the timing recovery. There are many different TEDs that uses different amounts of samples to estimate the error. A popular TED is Gardner which is presented below together with a similar TED.

Gardner

In [18] a TED algorithm is presented that requires double the symbol frequency to estimate the timing error. The algorithm given is

$$e[n] = y_I[n-1/2](\hat{y}_I[n] - y_I[n-1]) + y_Q[n-1/2](\hat{y}_Q[n] - y_Q[n-1]) \tag{2.23}$$

where

$$y_I[n] = \mathfrak{Re}(y[n]) \tag{2.24}$$

$$y_Q[n] = \mathfrak{Im}(y[n]) \tag{2.25}$$

The Gardner TED algorithm is a non-data aided (NDA) detector that can be transformed to a DD detector. NDA means that the detector does not use a pilot signal to train the detector in the beginning.

Linn

An NDA implementation of a TED comparable to Gardner has been derived in [19] by Linn. This implementation uses two lookup tables for each signal, for

QPSK it will be two tables for the I signal and two tables for the Q signal [19]. The calculation for the error is

$$e[n] = e_I[n] + e_Q[n] \tag{2.26}$$

where $e_I(n)$ and $e_Q(n)$ are defined as

$$e_I[n] = \frac{\hat{y}_I[n]y_I[n-1/2]}{\hat{y}_I^2[n] + y_I^2[n-1/2]} - \frac{\hat{y}_I[n-1]y_I[n-1/2]}{\hat{y}_I^2[n-1] + y_I^2[n-1/2]} \tag{2.27}$$

$$e_Q[n] = \frac{\hat{y}_Q[n]y_Q[n-1/2]}{\hat{y}_Q^2[n] + y_Q^2[n-1/2]} - \frac{\hat{y}_Q[n-1]y_Q[n-1/2]}{\hat{y}_Q^2[n-1] + y_Q^2[n-1/2]} \tag{2.28}$$

The result from the computations $e_I(n)$ and $e_Q(n)$ is limited to $[-0.5, 0.5]$ and can therefore easily be stored in lookup tables [19].

In reference [19] the performance of Gardner's algorithm and Linn's algorithm is compared. Linn's algorithm has a better performance than Gardner's algorithm but the power consumption of the two algorithms are not compared. By using lookup tables Linn's algorithm removes a lot of the heavy computations and therefore it could consume less power than Gardner's algorithm.

2.6.3 Packet Detection

There are two types of transmissions in wireless communications, broadcast and burst transmission. Broadcast transmission sends continuous signals and therefore does not require package detection. Burst transmission sends small packages with a preamble that the receiver has to detect to know when a package is sent.

The receiver compares the signal and frame header through correlation. If the correlation between the signal and the frame header is sufficiently large then the receiver assumes that it is a package that shall be received [20].

2.7 Modulation and Demodulation

Modulation is the process of mapping a symbol to a signal $x(t)$ that is suitable for transmission [7]. Demodulation is the process of retrieving the symbol from $x(t)$. There are many techniques for modulation. Three basic techniques are amplitude shift keying (ASK), frequency shift keying (FSK) and phase shift keying (PSK) [21].

2.7.1 Quadrate Phase Shift Keying

In QPSK the modulator will receive a sequence of binary bits that will be divided into pairs of two, i.e. the sequence 110100 will form the pairs 11, 01, and 00. The

pairs will then be converted to pulses, where a 1 is a positive pulse and 0 is a negative pulse. The first number of a pair will be multiplied by $\cos(2\pi f_c t)$ and second number with $\sin(2\pi f_c t)$, where f_c is the center frequency. Lastly these two pairs will be added together to form the signal that shall be sent. Figure 2.11 displays an overview of this process with one pair [21].

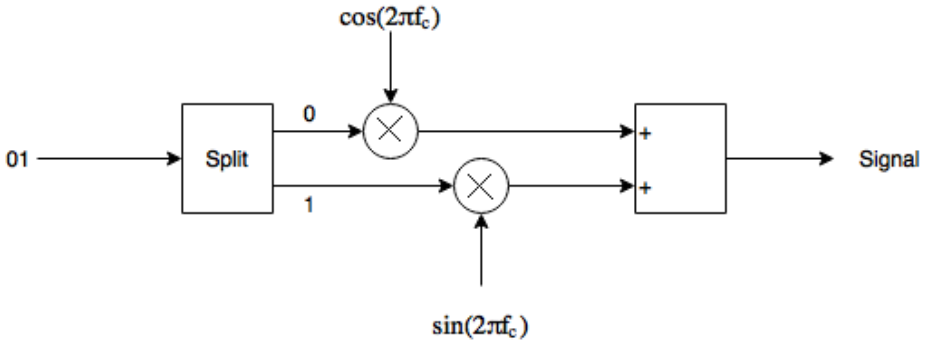

Figure 2.11: *QPSK modulation.*

For M-PSK it is possible to calculate the BER for a certain SNR. The calculation for the BER is [22]:

$$BER = \frac{2}{\max(\log_2 M, 2)} \sum_{i=1}^{\max(M/4,1)} Q\left(\sqrt{2\text{SNR}\log_2 M}\sin\left(\frac{(2i-1)\pi}{M}\right)\right) \tag{2.29}$$

where Q is defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2} dt \tag{2.30}$$

As can be seen in Fig. 2.12 a BER of 10^{-6} for QPSK gives an SNR of about 11 dB.

Figure 2.13 displays a constellation diagram of the four symbols represented by QPSK. Through the use of both the I and Q four symbols can be represented.

2.8 Power Optimization

The power consumption in the hardware is divided into two categories, static and dynamic power consumption. Static power consumption is the power the hardware consumes when only being switched on and not doing any operations. Dynamic power consumption is the power consumed when something in the hardware changes, i.e. switching bits or forwarding the signal. Because the platform

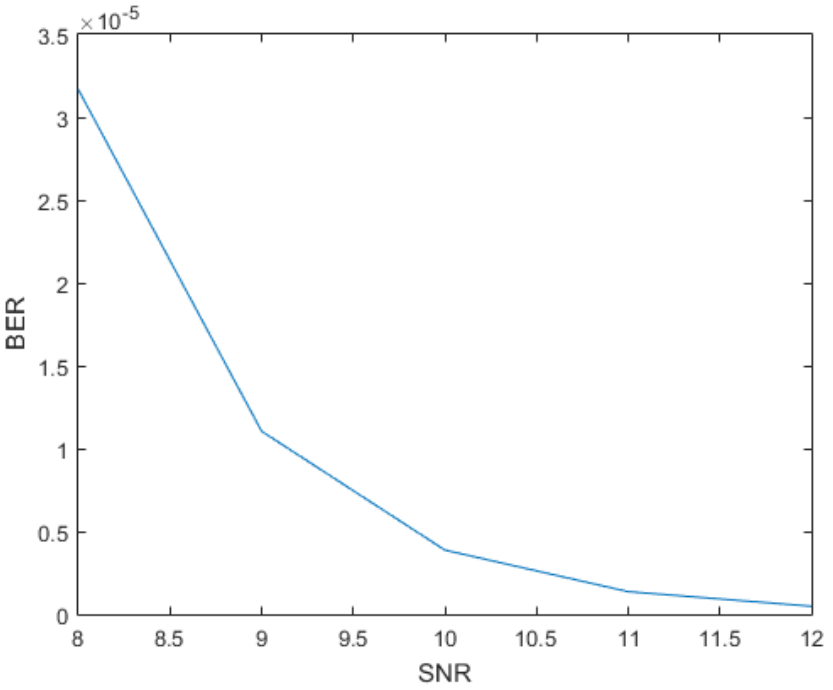

Figure 2.12: *BER versus SNR for QPSK.*

for the implementation is fixed the static power consumption cannot be reduced and therefore only the dynamic power consumption will be measured and evaluated.

The first step to reduce the power consumption is to choose algorithms that have few computations and give a result that complies with the requirements.

After efficient algorithms are chosen it is possible to reduce the number of bits used when receiving the signal. A lesser number of bits can decrease the SNR of the signal but it will also reduce the power consumption. By reducing the number of bits there will be fewer bits changing each clock pulse.

The amount of computations per time unit required by an algorithm can differ, e.g. the straightforward filter solution compared to the polyphase filter. By choosing algorithms that can use fewer samples it is possible to reduce the clock rate in certain areas of the FPGA.

It is also important to take the power consumption of different computations in to consideration when comparing algorithms. A multiplication often consumes more power than an addition.

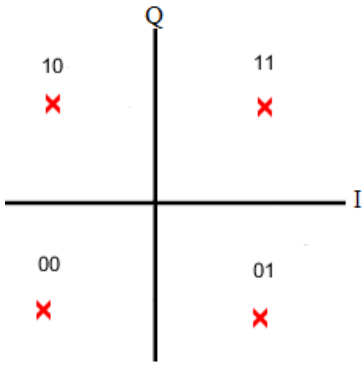

Figure 2.13: *QPSK constellation diagram.*

2.9 Orthogonal Frequency Division Multiplexing

As stated before the reason for OFDM is to split a high-rate datastream into a number of lower rate datastreams. A signal in OFDM consist of a sum of subcarriers that are modulated by either PSK or QAM. To generate the subcarriers used when transmitting the lower rate datastreams the inverse Fourier transform of N QAM or PSK symbols is used, where N is the number of subcarriers. The transform can be implemented by the inverse fast Fourier transform (IFFT) instead of the inverse discrete Fourier transform (IDFT) which reduces the number of multiplications [3].

To minimize the ISI a guard time is used before each OFDM symbol. To calculate the guard time the maximum time spread has to be known, then the guard time is chosen larger than the expected time spread. The guard time could introduce inter carrier interference (ICI) if the guard time consist of no signal at all. ICI is a distortion that occurs when the subcarriers are not orthogonal which is caused by the OFDM symbols not having an integer number of cycles in the IFFT interval. By cyclic extending the OFDM symbol in the guard time the ICI will be eliminated [3].

After choosing the guard time larger than the delay spread, the symbol duration has to be decided. It is desirable to have a large symbol duration because there is a loss in SNR caused by the guard time. But the symbol duration cannot be too large because it requires larger implementation complexity and is more sensitive to phase noise and frequency offset. Therefore the symbol duration is often chosen as five times larger than the guard time, which gives a 1 dB SNR loss because of the guard time [3].

2.10 Frequency Hopping Spread Spectrum

Frequency Hopping means that the carrier frequency of the signal changes periodically. The carrier frequency stays the same during the intervals T, then after the interval the carrier hops to another (could be the same) frequency. The available set of frequencies the carrier can hop to is called the hop-set and the hopping pattern is decided by a spreading code [3].

In FHSS there is a distinction made based on the hopping rate of the carrier. If the hopping rate is greater than the symbol rate it is called a fast frequency hopping (F-FH) and if it is smaller it is called a slow frequency hopping (S-FH). F-FH means that one symbol will be transmitted over multiple frequency whereas S-FH means that multiple symbols will be transmitted at the same frequency [3].

The bandwidth occupied by the signal varies depending on if S-FH or F-FH is used. The bandwidth occupied with S-FH depends mainly on the bandwidth of the transmitted signal. With F-FH the bandwidth occupied depends on the pulse shape of the hopping signal at one hopping frequency. The bandwidth occupied becomes very large if the frequency changes are very abrupt but if the frequency changes are smoother then the bandwidth decreases. To change frequency smoother the transmitted power can be lowered before a frequency hop and increased after the hop [3].

3
Method

This chapter goes through how the transmitter and receiver were implemented and which tools were used to implement and test the system.

3.1 Tools

This section will give an overview of the different tools that have been used when implementing and analyzing the system.

Simulink

Simulink is an add-on to Matlab that can be used to simulate hardware and has a set of predefined blocks. To Simulink there are a few extensions that contain more predefined blocks. DSP System Toolbox and DSP Builder are the only extensions to Simulink used in the thesis.

DSP Builder

DSP Builder is an extension to Simulink provided by Altera. DSP Builder contains a set of predefined blocks and the possibility to generate hardware description language (HDL) code and test benches for the HDL code.

Quartus

Quartus is used to analyze and compile the HDL description, and to fit the design to the FPGA. Quartus is also used to estimate the power consumption of the design. To get a better power estimation the design has to be simulated with an appropriate input.

ModelSim

ModelSim is used to run simulation of HDL code. In this thesis ModelSim is used to simulate the design and create data that is used to get a better power estimation.

3.2 Simulink Reference Model

Simulink was first used to implement the transmitter and receiver. The receiver was later implemented with DSP Builder blocks instead of Simulink blocks but the transmitter remained implemented with Simulink blocks because the power consumption of the transmitter did not have to be measured.

The implementation of the receiver was carried out in different stages. The first step was to create a signal for the receiver to retrieve. To create this signal a transmitter was implemented without any distortions added to the signal.

The receiver was then implemented and when the undistorted signal could be retrieved the transmitter was tweaked to add distortions to the transmitted signal. The receiver was then adapted to manage these distortions until the required BER was satisfied.

3.3 DSP Builder Reference Model

When the receiver was working with Simulink the blocks were replaced for those from DSP Builder. DSP Builder contained all the blocks used in Simulink except for the downsample block. The downsample block removes a number of samples and lowers the clock rate for the following part of the system.

Because DSP Builder did not contain this block it was not possible to lower the clock rate of the design in Matlab, it had to be performed in Quartus instead. It also meant small changes in the design where a valid signal had to be passed along with the received signal.

3.4 Transmitter

The base for the transmitter comes from Simulinks model "HDL Optimized QPSK Transmitter", this model produces bits and modulates them with QPSK. To create the required transmitter the sample time was changed to produce a 25 kHz signal and the interpolation filter was changed. After the signal had been interpolated the real part of the signal was multiplied by a 25 kHz cosine and the imaginary part was multiplied by a 25 kHz sine. An overview of the transmitter can be seen in Fig. 3.1.

The signal was then added together and multiplied by a 2 MHz sine, which puts the signal at 2.025 MHz. The signal was then exposed to AWGN, phase and fre-

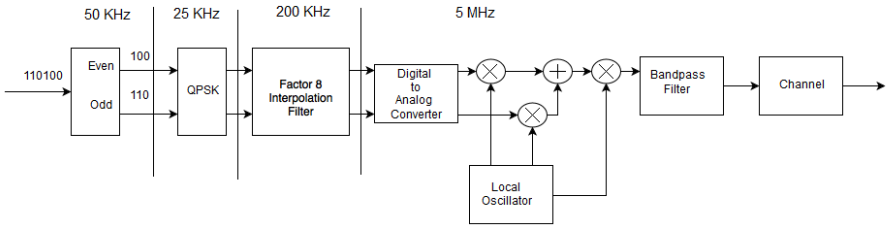

Figure 3.1: *Overview of the transmitter.*

quency shift, and time delay. As a final distortion a second 25 kHz QPSK randomly generated signal with the same amplitude was added to the transmission 75 kHz away from the first signal. Figure 3.2 displays the frequency spectrum of the two signals. The frequency spectrum is mirrored and is therefore identical at -2 MHz.

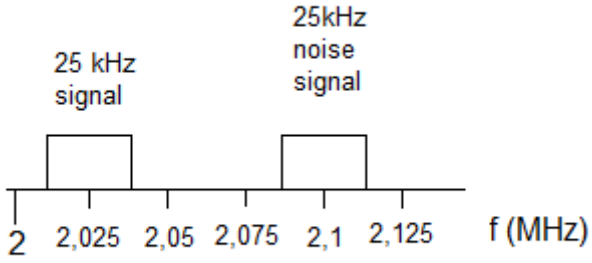

Figure 3.2: *The frequency spectrum around 2 MHz.*

3.5 Receiver

The receiver also started with a base from Simulink, the model "HDL Optimized QPSK Receiver With Captured Data". This model contains a decimation filter, carrier and time synchronization and package detection.

To create the receiver the first step was to create a matched SRRC filter. To create the filter the signal was sent undistorted through the transmitter and receiver filter and without being modulated onto the carrier frequency. The filters were tweaked until there was only a small amount of ISI in the transmitted signal. Because matched filters were used it was not possible to remove all of the ISI from the signal.

The Matlab implementation was analyzed to understand the different subsys-

tems. Some of the systems were only implemented with code and not Simulink blocks, to understand how these system worked the code was replaced with Simulink blocks instead. When the whole system was implemented with Simulink blocks the transmitter was tweaked to add distortions to the signal. Then the subsystems at the receiver were changed to newer implementations that might have a lower power consumption. An overview of the receiver implemented in DSP Builder and an overview of how the different subsystems are connected is displayed in Appendix A.

3.5.1 Decimation Filter

The filter is implemented with a predefined block in both the Simulink and DSP Builder version of the receiver. The block creates the architecture for the filter based on the coefficients of the zeros and poles of the desired filter.

Both Simulink and DSP Builder have a predefined block for decimation filters and these blocks are used in both versions of the receiver. The blocks require the coefficients of the zeros of the required filter, then the blocks construct a filter based on the coefficients. The only information about the implementation of the decimating filter in DSP Builder available is that the filter decimates at the input before the multiplications. Therefore the conclusion can be made that the filter is implemented as a polyphase filter. It would have been possible to split up the filter to create both a polyphase filter and a multistage filter but there was not enough time.

Because of the time constraint a decision had to be made about focusing on the filter part of the receiver or the synchronization and package detection. Both parts are important in the receiver but the decision was made to focus on the latter part.

To decide the zeros of the filter and get the coefficients, Matlabs filter tool *fdatool* was used. With *fdatool* it is possible to choose what kind of filter it should create, which was set to raised-cosine. It is then possible to choose different parameters to create the required filter. The values chosen for the filter used in this thesis are:

- Units: Normalized
- Order: 80
- w_c: 0.125
- Rolloff: 0.75

To increase the amplitude of the signal the filter gain was changed to 0.000907043. The reason that the amplitude was changed to the specific value is because the amplitude of the signal shall be close to 0.6. The filter uses symmetrical coefficients and uses a total of 22 multipliers. Because it is a polyphase filter this means that it requires 22 multiplications for every output sample.

The order and rolloff for the filter was chosen larger than required to easily find

a filter that did not introduce ISI to the signal. Both the order and rolloff could probably be decreased, the affect of decreasing the order should be a reduction of the power consumption. The reduction in power consumption should be close to linear because it reduces the number of computations the filter calculates.

The signal was interpolated by a factor of 8 at the transmitter. At the receiver the ADC then samples at 5 MHz which gives 200 samples per symbol. After the 25 kHz signal has been extracted by multiplying the received signal with a 2 MHz signal the received signal is decimated by a factor of 25. This decimation is accomplished by throwing away the samples though a decimation filter would increase the performance of the receiver. Unfortunately there was not enough time to change this part, and it might not be possible to keep this decimation with FHSS. The reason that the decimation can not be accomplished with FHSS is because the frequency carrier is not known and therefore it can not be determined how many samples that are required. Then a decimation by a factor of 4 was accomplished by the decimation filter at the receiver. This gives one extra sample per symbol to get a better synchronization and for the chosen time synchronization algorithms it is required to have two samples when synchronizing the timing error. The signal will be decimated by a factor of 2 when performing the time synchronization.

3.5.2 Carrier Synchronization

The carrier synchronization consists of two parts, a coarse synchronization and then a fine synchronization. The coarse synchronization was implemented as an power of M carrier recovery with Luise algorithm. First the signal was increased by a power of 4 because QPSK is a 4-phase modulation. By raising the power the information bearing signal is removed and the unmodulated carrier can be retrieved [9]. With this implementation the requirements of the system were met.

Luise algorithm was chosen for the coarse frequency synchronization because it had less computations compared to the algorithms presented in [14]. The value of N was set to 512 because then it was possible to use a bit shift for the division. The value of M was chosen to 3 to lower the number of multiplications. If a better approximation would be required then M can be increased. If a coarse synchronization has to be performed more often then the value of N can be decrease but then the division cannot be performed by a bit shift.

To calculate the angle, the CORDIC block was used in both the Simulink and DSP Builder implementations. A small change was made in the Luise algorithm where the divisions were moved to after the addition, which reduced the number of divisions to one instead of three. The signal is divided by 512 after the addition which can be implemented with a bitshift of nine bits. This should reduce the power consumption because the number of calculations is decreased and a bit shift is simpler arithmetic operation than a multiplication.

This could affect the error estimation slightly. If the value one is be passed through the three divisions in the original implementation the result is

$$\frac{1}{511} + \frac{1}{510} + \frac{1}{509} = 0,005882 \tag{3.1}$$

By passing the value one through the implementation with one division the result is

$$\frac{3}{512} = 0,0058593 \tag{3.2}$$

This shows that there is a small difference in the result between the two computations but it is not a large difference. Although a more detailed study would be required to be certain how the change affect the synchronization.

The fine carrier synchronization was tested with three different algorithms, Costas loop, DD QPSK Costas loop and QPSK Costas loop. The different versions of Costas loop were chosen because Costas loop has a very simple design with few computations, and the different versions also have very few computations. If the system meets the requirements with one of these algorithms it would be hard to decrease the number of computations in the fine carrier synchronization without removing it.

When the fine carrier synchronization was implemented as Costas loop it did not give a sufficiently good BER, 0.1769%, early in the simulation.

The DD QPSK Costas loop was implemented without a timing synchronization after it because the loop does both the carrier synchronization and timing synchronization. It did fairly well when there was a small time delay. When the time delay increased, the synchronization could not perfectly synchronize and the BER increased. After 6300 symbols the BER was 0.0006349%. Although more symbols would be required to ascertain the BER it was possible to notice during the simulation that the algorithm could not synchronize large time delays. The BER could be sufficient depending on the application but for this thesis the BER shall be below 0.0001%.

The DD QPSK Costas loop was then changed to only use the signed signal for the error estimation and a timing recovery was added after the carrier synchronization. This implementation fulfilled the required BER.

The QPSK Costas loop was implemented in two different ways. The first implementation was according to Fig. 2.9. The other implementation was according to Fig. 3.3. The difference in these two implementations are the computations required while the result still remains the same. Table 3.1 shows the difference in calculations. The table shows the signed I and Q value, the most significant bit (MSB) of I and Q, what the multiplexer will output, and what the summation with multiplier will output.

As can be seen in Fig. 3.4 the error estimations from the coarse and fine carrier synchronization are added together and then forwarded to an NCO. The phase

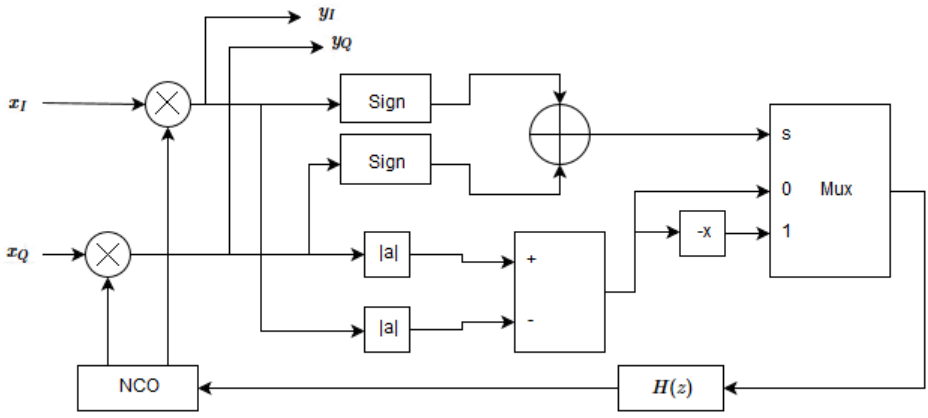

Figure 3.3: *QPSK Costas implemented with XOR and multiplexer.*

I	Q	MSB(I)	MSB(Q)	XOR	MUX ($\|Q\| - \|I\|$)	SUM ($\hat{I} \cdot Q - \hat{Q} \cdot I$)
1	1	0	0	0	Q-I	Q-I
-1	1	1	0	1	I-Q	I-Q
1	-1	0	1	1	I-Q	I-Q
-1	-1	1	1	0	Q-I	Q-I

Table 3.1: *Comparing QPSK implemented with multipliers or multiplexers.*

error detector contains the computations in the fine carrier synchronization. The NCO calculates a sine and cosine wave with a frequency based on the received value. The waves are then multiplied by the signal to remove any phase and frequency errors. A more detailed overview of how the coarse and fine frequency carrier synchronizations are connected is shown in Appendix A.

3.5.3 Time Synchronization

The time synchronization was implemented as a multirate time synchronization, see Fig. 3.5 for an overview, because the signal received from the ADC is oversampled and there is no feedback to the ADC to change the sample time.

When implementing the time synchronization only the TED was changed from the implementation already existing in Matlab. The Matlab implementation already used a multirate solution with interpolation and the NCO described earlier. Although it would have been valuable to test different interpolation algorithms there was unfortunately not enough time.

Because the signal is interpolated by a factor of two it gives $\alpha = \frac{T_i}{T_s} = 2$. The interpolation is implemented as a fractional delay filter. The signal is interpolated by a factor of two and then the original two samples are discarded and therefore

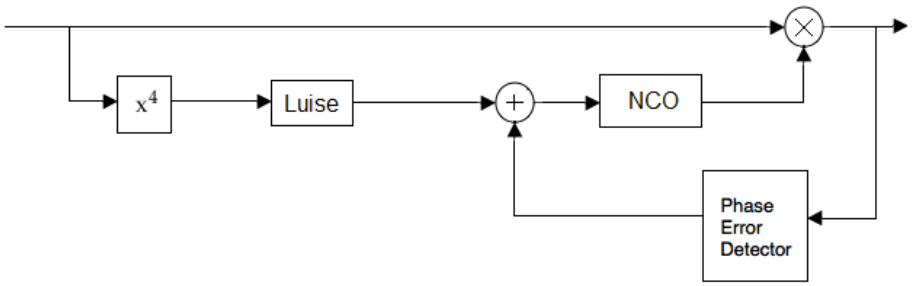

Figure 3.4: *Simple overview of the carrier synchronization.*

the sample rate is not changed due to the interpolation. With $w[n] \approx \frac{1}{\alpha}$, the NCO will underflow on average every second clock cycle. The underflow will be used to signal that the current sample is valid, therefore the signal will be downsampled by a factor of two.

Two algorithms were tested for the TED, Gardner and Linn. Gardner is a popular detector because it only requires two samples per symbol and has a good performance [19]. In [19] Gardner's and Linn's performance is compared and Linn's has a better performance in general. Therefore it was off interest to know if Linn's implementation also had a lower power consumption.

Figure 3.6 displays the implementation of Gardner. This implementation was very easy to implement and it also met the BER requirements.

When implementing Linn's algorithm it had to be changed slightly. In DSP Builder the lookup tables do not support two inputs and therefore a slightly different implementation than the one proposed in [19] was implemented, see Fig. 3.7 for the implemented TED. In order to find a matching value in the lookup table, the eight least significant bits (LSB) are used because the lookup table only contains 64 values. Because the signal has been synchronized with coarse and fine carrier synchronization the signal does not contain a large amount of error. The largest amount of error is contained in the LSBs and therefore they are used for the lookup table.

An overview of the timing synchronization is displayed in Appendix A. To get a better value at the correlation the sign of the samples were extracted and sent to the package detection, this means that the sample value only could be 1 or -1. The reason this gives a better result is because if there are fluctuations in the sample values, around ± 0.1, then the correlation is reduces and there is a higher chance to miss a package. The downside with extracting the sign is that it implies a higher chance to have false detection because the correlation increases for patterns similar to the frame header.

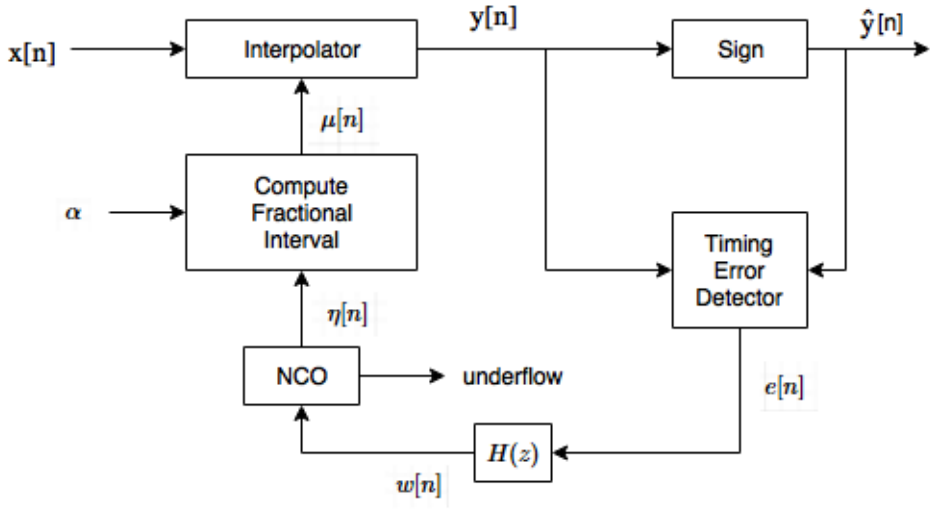

Figure 3.5: *A multirate time synchronizaiton.*

3.5.4 Package Detection

For the receiver to be able to detect when a package is sent two matched filters are used. By using two matched filters, a package can be detected even if the carrier synchronization locks on to the wrong phase. Figure 3.8 displays the two summations of the I and Q signals before the two filters.

This setup is used to detect packages even if the synchronization does not lock on to the correct phase and the signal is therefore rotated.

The matched filter then compares the received symbols with the symbols in the preamble. If a large amount of the symbols match then a high number will be received from the matched filter. The magnitude is then calculated from the values received from the filters and is then compared to a threshold to determine if the correlation was large enough to decide that a package was sent. The calculation for the magnitude is:

$$|V| = \sqrt{C_1^2 + C_2^2} \tag{3.3}$$

This calculation requires many multiplications and the square root is a complex and expensive calculation. An algorithm for estimating the magnitude is presented in [23] called "alpha max plus beta min"

$$|V| \approx \alpha Max + \beta Min \tag{3.4}$$

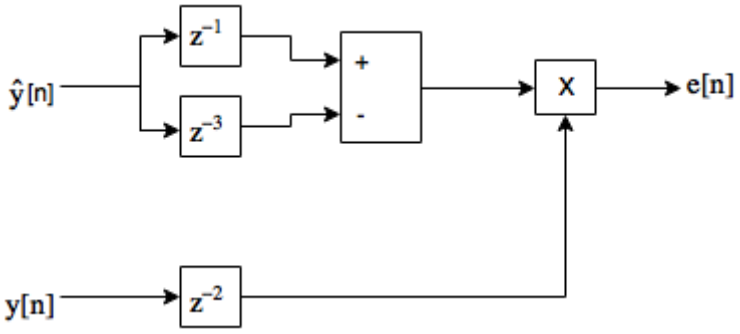

Figure 3.6: *Implementation of Gardner's algorithm.*

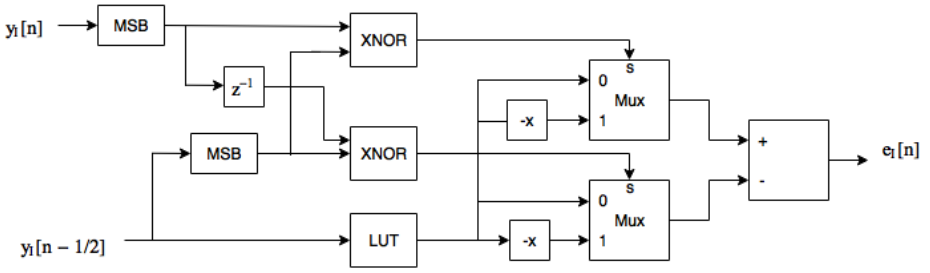

Figure 3.7: *Linn's algorithm with one input lookup table.*

The accuracy of this estimation varies depending on the chosen values for α and β. The largest value of the C_1 and C_2 is the max value and the other is the min value of C_1 and C_2 [23]. Because the magnitude is only used when comparing with the threshold it could be possible to use $(I)^2 + (Q)^2$ but to compare the estimations of $\alpha Max + \beta Min$ the calculation for the magnitude is used. Comparing $(I)^2 + (Q)^2$ to $\alpha Max + \beta Min$ a conclusion can be made that $\alpha Max + \beta Min$ should consume less power. This is because the number of bits required to represent the result is less for $\alpha Max + \beta Min$. Depending on the choice of α and β it is also possible to implement the multiplications with bit shifts in $\alpha Max + \beta Min$.

For the first implementation of the αMax+βMin, the values $\alpha = 1$ and $\beta = 0.4$ were used, which will be called Max+0.4Min. This gave a good enough estimation of the magnitude to decide if a package was detected or not.

The values $\alpha = 1$ and $\beta = 0.5$ were also tested, which will be called Max+0.5Min. With these values it is possible to use bit shift instead of multiplication which could require less power.

The values for αMax+βMin were chosen from [23] to give good results. There are other values as well for both α and β but because these two gave a good result no other values were required. The reason the αMax+βMin algorithm were chosen

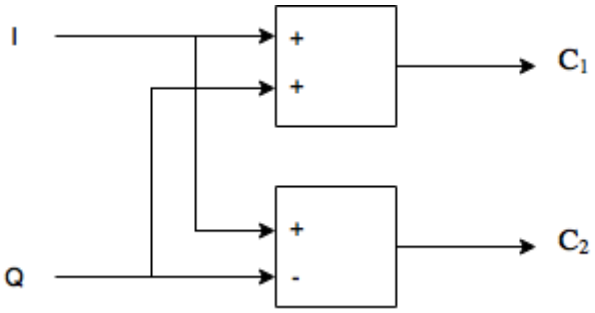

Figure 3.8: *Calculation before the matched filter.*

to calculate the magnitude was because it gives an accurate enough result for the package detection and reduces the number of calculations required in the package detection.

By observing the correlation value an appropriate value for the threshold could be decided. The threshold value was set to 21 and the largest correlation value is 25. If more knowledge of the signal would be collected, then [20] presents a calculation for the threshold value based on the requirements on detecting false packages and missing packages.

After a package has been detected the conjugated value of the filter output is stored and multiplied by the samples for that package. This is performed to rotate the samples correctly if the carrier synchronization has locked on to the wrong phase. To demodulate the signal and get the correct bit the MSB is taken from each sample. The MSB is the correct demodulated bit because of the chosen constellation points. An overview of the package detection and demodulation block can be seen in Appendix A, there it is also possible to see an overview of the package detection.

3.6 Power Optimization

After the model was transferred to DSP Builder blocks the power, BER, and SNR were measured.

First, the number of bits used to represent the signal was changed to see how the different measurements were affected. The signal is represented using fixed point with a number of bits. One bit is used to represent a positive or negative value and one is used to represent the integer part, the remaining bits are used for the fractional part. This means that when 20 bits are used 18 bits are used for the fractional part.

When the number of bits could not be decreased any more the different algorithms for each subsystem were tested against each other. The BER and power consumption were always measured and the SNR was measured when applica-

ble.

To measure the SNR the modulus of the expected value, $|x_e[n]|$, of the signal was calculated by sending the signal through the transmitter and receiver filter without any distortions. Then the distorted signal was transmitted and the absolute value of the signal, $|x_r[n]|$ was calculated. With these two values the noise could be calculated by

$$w[n] = |x_r[n]| - |x_e[n]| \tag{3.5}$$

With the calculated noise it was possible to estimate the SNR by

$$SNR = 10\log_{10}\left(\frac{\sum_0^N (x_e[n])^2}{\sum_0^N (w[n])^2}\right)[dB] \tag{3.6}$$

After comparing the power consumption of different combinations of the subsystems, the combination that had the lowest power consumption was chosen. Afterward the clock rate was reduced after the decimation filter. After the decimation filter the clock rate could be reduced 100 times to 50kHz which reduces the amount of times every component has to forward the signal. If the synchronization was in the analog part then the clock rate could be reduced to 25 kHz because it is split up in two parts, I and Q, but because the synchronization requires two sampels the clock rate has to be twice as high.

4

Result

In this chapter the results from the different evaluations and measurements will be presented. The chapter starts with comparing the power consumption of the system with different algorithms implemented and comparing the SNR when applicable. The power consumption of the whole system is measured, not only a certain component, except for the oscillator and filter. There was not enough time to lower the power consumption in these subsystems and therefore it can be valuable to know how much power these consume. In Table 4.1 the power consumption of the filter and oscillator with 10 bits is displayed.

	Power Consumption (mW)
Oscillator	0.44
Filter	0.22

Table 4.1: *The power consumption of the oscillator and filter.*

The final design has a power consumption of 1.05mW and a BER of 0 after 37440 bits had been retrieved. To be certain that the BER is 10^{-6} more bits would have to be received, at least more than 10^6 bits but it took too long to simulate, the simulation had been runing over 24 hours for the 37440 bits. With 10 bits the SNR was around 14 dB which according to (2.29) yields a BER of 1.5×10^{-7}.

Table 4.2 displays the subsystems implemented in the final system, for the most part the signal was represented using 10 bits.

When comparing the algorithms all parts except one was the same as the final system, Max+0.4Min was used instead of Max+0.5Min. When comparing the different algorithms only one part is replaced, i.e. when comparing Gardner and Linn the rest of the system will stay the same.

Subsystems
Oscillator
Digital decimation filter
Luise algorithm with one division
QPSK Costas loop with multiplexer
Linns TED
Matched filter
Max+0.5Min

Table 4.2: *The subsystems implemented in the final system.*

The transmitter adds a random signal as a distortion and therefore the received signal can differ slightly which can affect the estimated power consumption. Therefore the same signal is used when comparing algorithms in the same subsystem. It is also important to note that it is only an estimation of the power consumption and therefore the power consumption could differ when implementing the actual system.

4.1 Carrier Synchronization

Only the Luise algorithm was implemented as the coarse carrier synchronization but in two different versions. Table 4.3 compares the power consumption of the system implemented with Luise algorithm with three divisions and with one division.

	Power Consumption (mW)
Three divisions	2.12
One division by bitshift	1.90

Table 4.3: *Comparison in power consumption between Luise implementations.*

As stated earlier two implementations of the fine carrier synchronization did not meet the required BER. Although the DD QPSK Costas loop did not meet the requirements the power consumption of the system was measured because it could be used if a slightly higher BER was acceptable. Table 4.4 compares the DD QPSK Costas loop, QPSK Costas loop with multipliers, and QPSK Costas loop with multiplexer.

The result of the carrier synchronization can be seen in the two eye diagrams in Fig. 4.1. The first eye diagram is taken before the synchronization and the second eye diagram is taken after the carrier synchronization. The carrier synchronizations used in the second eye diagram are the Luise algorithm and QPSK Costas loop with multiplexer.

The constellation plots before and after the carrier synchronization were also

	Power Consumption (mW)
DD QPSK Costas loop	1.87
QPSK Costas loop multiplier	1.91
QPSK Costas loop multiplexer	1.90

Table 4.4: *Comparison between fine carrier synchronization power consumption.*

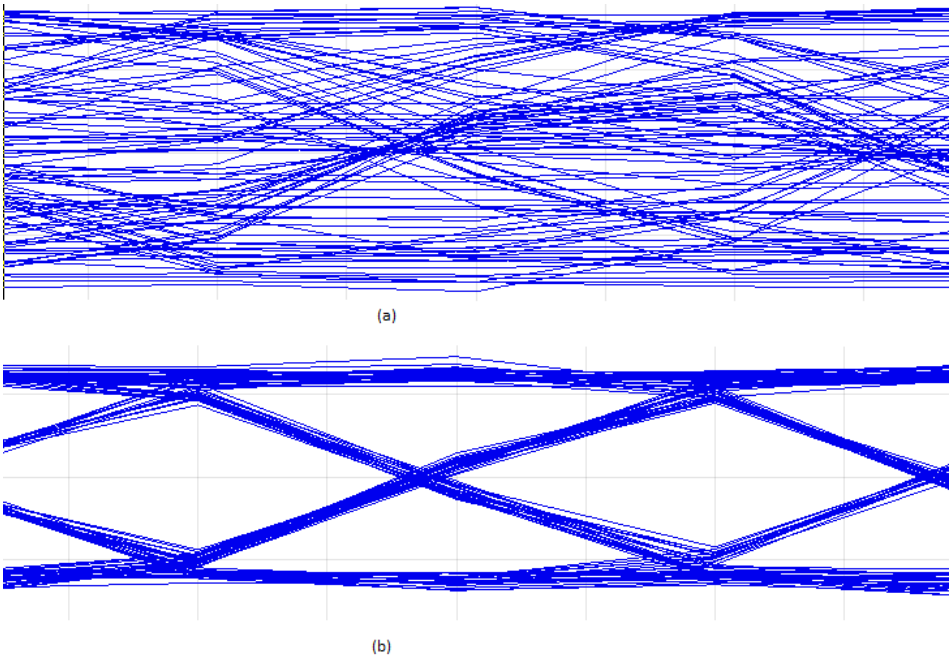

Figure 4.1: *(a) Eye diagram before synchronization. (b) Eye diagram after synchronization.*

compared to further verify the accuracy of the synchronization. Figure 4.2(a) shows the constellation diagram before the synchronization and Fig. 4.2(b) shows the constellation diagram after synchronization with Luise algorithm and QPSK Costas loop.

4.2 Timing Synchronization

The two TEDs that were implemented were Gardner and Linn. Table 4.5 compares the power consumption of the system with these two TEDs implemented.

Figure 4.3 displays two constellation diagrams after the time synchronization.

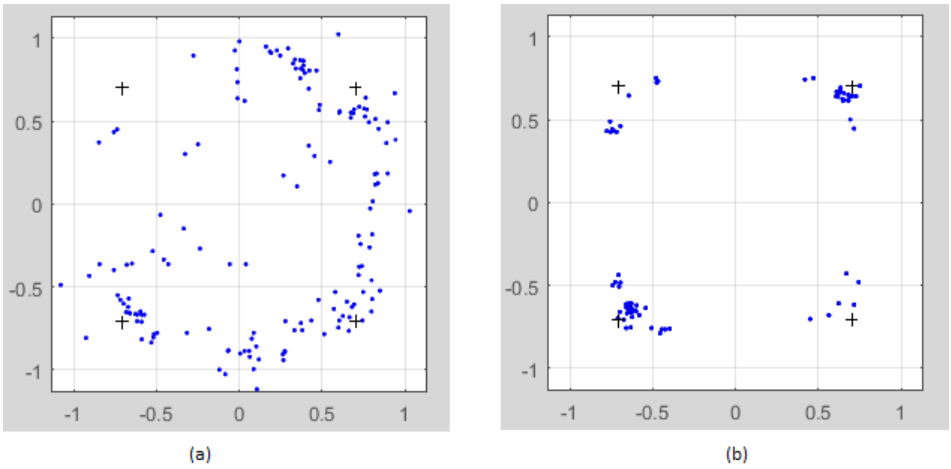

Figure 4.2: *Constellation plot comparing (a) before carrier synchronization (b) after synchronization.*

	Power Consumption (mW)
Gardner	1.95
Linn	1.90

Table 4.5: *Comparison between TEDs power consumption.*

Figure 4.3(a) is after Gardner and Fig. 4.3(b) is after Linn. The constellation diagrams are after the time synchronization but before the sign is extracted from the sample.

4.3 Package Detection

The αMax+βMin algorithm with two different beta values were compared to the calculation for the magnitude. Figure 4.4 displays the estimated magnitude of the three implementations with the magnitude being rounded to the nearest integer.

Because the magnitude is being rounded to the nearest integer the calculations have a small quantization error.

Table 4.6 compares the power consumption of these three algorithms. Both of the αMax+βMin implementations use nine bits to represent the value while the magnitude calculation requires more bits to represent the square value.

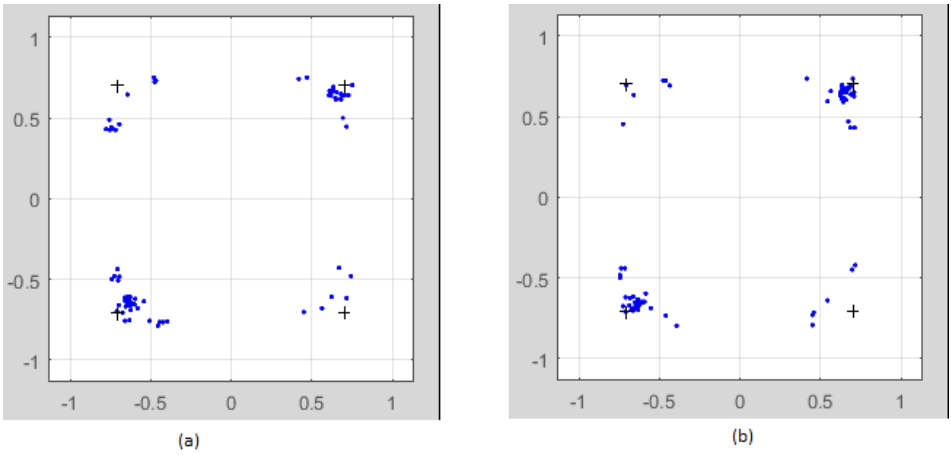

Figure 4.3: *Constellation plot after time synchronization (a) Gardner (b) Linn.*

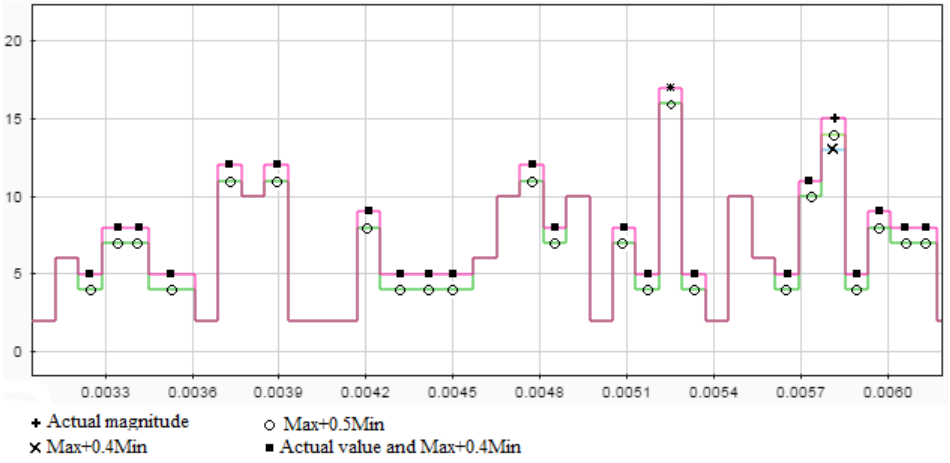

Figure 4.4: *Magnitude comparison.*

	Power Consumption (mW)
Magnitude	2.01
Max+0.4*Min	1.90
Max+0.5*Min	1.84

Table 4.6: *Comparison between magnitude calculation power consumption.*

4.4 Bit Reduction

The number of bits was reduced until the BER became too high or could not be lowered any more because of the algorithm. A few algorithms have limitations on the number of bits required because of divisions, i.e in Luise algorithm the bits are shifted by nine bits and therefore it would be impractical to reduce the number of bits below ten.

Figure 4.5 displays a graph comparing the bits used and the power consumption. When 20 bits are used the whole system uses the same bit length but when the number of bits is lowered some computations require more bits and can therefore not be lowered.

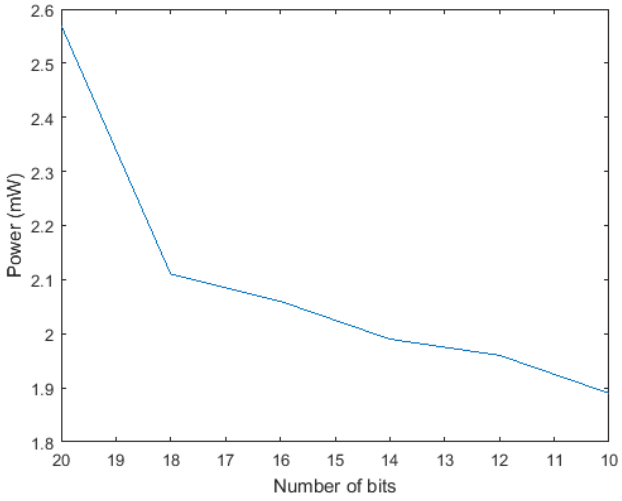

Figure 4.5: *Graph over power consumption decrease with bit reduction.*

It is also important to compare the number of bits with SNR. The SNR is measured after the timing synchronization but before it is signed. Figure 4.6 displays a graph with the number of bits and the SNR measured with 500 symbols.

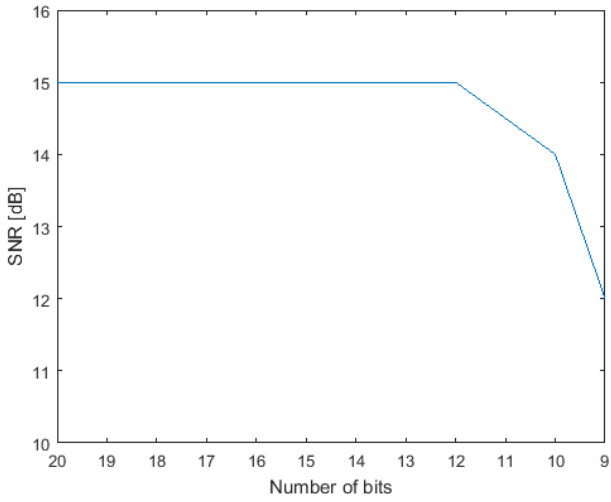

Figure 4.6: *Graph over SNR when reducing the number of bits.*

4.5 Clock Rate Reduction

Table 4.7 compares the power consumption of the receiver with a single clock at 5MHz and with two clocks at 5MHz and 50kHz. The 50kHz clock is used after the decimation filter and has to run at twice the speed of the signal, 25kHz, because two samples are used to synchronize. There was not enough time to set up a testbench for the system with a lower clock rate which means it is untested. The system could require small changes to work correctly with the lower clock rate. The result in Table 4.7 should therefore be seen as an estimation of what the power could be with a lower clock rate.

Both of the clock rates are measured with the final design. This was the last power reduction made to the system and therefore the final design could be used.

	Power Consumption (mW)
5MHz	1.86
50kHz	1.05

Table 4.7: *Comparison between power consumption at different clock rates.*

4.6 Resource Allocation

Table 4.8 gives an overview of the resources used by the receiver. The digital signal processing (DSP) blocks are used for the computational heavy parts of the system. For the simpler computations the adaptive lookup tables (ALUT) and registers are used.

	DSP Blocks	Block Memory	ALUTs	Registers	Clock
Oscillators	7	10240	284	588	5 MHz
Decimation Filter	12	0	329	292	5 MHz
Carrier Synchronization	16	23070	1812	447	50 kHz
Timing Synchronization	4	0	321	298	50 kHz
Package Detection	16	0	264	129	50 kHz
Total	55	33310	3012	1755	

Table 4.8: *Resource allocation of the different subsystems.*

5
Discussion

This chapter starts with an evaluation and discussion of the results gained from the measurements. Then the methods used when implementing the system and measuring the results are evaluated.

5.1 Result

The power consumption of the filter and oscillator together is 0.66mW which is about a third of the power consumption of the whole system when the clock rate is not reduced, 1.86mW. The power consumption of the oscillator and filter is not reduced when the clock rate is reduced. Therefore it becomes an even larger percentage of the power consumption when the clock rate is reduced. The filter and oscillator consumes 63% of the total power consumption of the system with a reduced clock after the filter.

When comparing the fine carrier synchronization it was expected that the QPSK Costas loop with multiplexer would have a lower power consumption compared to the implementation with the multiplier. The result from the two measurements were 1.91mW for the multiplier and 1.90mW for the multiplexer which gives a 0.01mW difference in power consumption. But because it is only a power estimation this could be in the error margin. Therefore both implementations could be thought of having the same power consumption.

When comparing the TEDs it was interesting that Linn's implementation with the lookup tables had a lower power consumption than Gardner even though Linn's required a few calculations before the lookup tables.

Comparing the two αMax+βMin algorithms to the magnitude calculation it can

be seen that both αMax+βMin implementations reduce the power consumption. The two implementations of αMax+βMin also have a difference in power consumption which is because Max+0.4Min uses multiplication and Max+0.5Min uses bit shift.

Comparing the calculated value in Fig. 4.4 it can be seen that Max+0.4Min is very close to the actual value while Max+0.5Min is slightly smaller all the time. Because of this there is a greater chance of false detection with Max+0.5Min but if the threshold is high enough this should not be a problem. Max+0.4Min would probably differ more if the magnitude was not rounded to the closest integer. There are also other values that could be used as α and β to get a more accurate result but where the magnitude is calculated it is not necessary with a more accurate result.

It was expected that the power consumption would go down as the bits were reduced. As can be seen in Fig. 4.5 the power consumption does not have a constant decline, instead it starts to level out as the bits are reduced. This is to be expected because with higher precision the lower bits will shift more often and therefore require more power.

When comparing the SNR at the different bit lengths it was expected that the SNR would be affected more by decreasing the number of bits. As can be seen in Fig. 4.6 the SNR does not start to decrease until only 10 bits are used. When 9 bits are used the SNR is 12 dB which is enough to get a BER at 10^{-6} but the synchronization algorithms can not synchronize correctly.

Although it was expected that the power consumption would be reduced by reducing the number of bits, it was unexpected that the power consumption was reduced by such a large amount. The power consumption when comparing 20 bits to 16 bits was 23% and when comparing 20 bits to 10 bits it was 28%. Even though halving the number of bits did not give a power reduction of 50% it is a large power reduction. The reason that the power consumption was not reduced by 50% can be by a number of reasons. The package detection used a lesser number of bits to represent the signal. After the matched filter the signal was rounded to an integer number between -25 and 25, therefore only 6 bits were required to represent the signal. There were some parts in the system were the number of bits could not be lowered. One of these parts is the filters because they were implemented with predefined blocks which used predefined number of bits.

The final result of a 44% reduction of power consumption was unexpected. With the new clock frequency of 50kHz, the estimated reduction in power consumption was somewhere around 25-30%. It was expected to be around 25-30% because the clock rate was reduced only for a part of the system. The signal was also passed forward with a valid signal and therefore kept static between each valid pulse. This means that only the forwarding of the signal in each component was reduced when the clock rate was reduced. Although as can be seen in Table 4.8 the last three subsystem contains a large amount of the system.

When comparing the amount of resources allocated by the different subsystems

an estimation of which subsystem consumes most power is possible. As can be seen in Table 4.8 the carrier synchronization allocates more than half of the ALUTs, 1812 of 3012, and the largest amount of DSP blocks, 16 of 55. Through these values it can be assumed that the carrier synchronization consumes the largest amount of power of the subsystems with a 50 kHz clock rate. As can be seen in Table 4.1 the oscillator and decimation filter consumes 0.66 mW together which is 63% of the power consumption of the final system.

5.2 Method

When implementing both the transmitter and receiver a model from Matlab was used as base. When implementing the transmitter this was the best choice because the focus for the thesis is the receiver but to test the receiver a transmitter is required. By using the transmitter from Matlab as a base and changing the necessary parts it took less time setting up the transmitter compared to implementing the transmitter from scratch.

When implementing the receiver it could have been implemented from scratch which might have yielded a different design. Using the Matlabs receiver as a base gave a better and quicker understanding of the system and which subsystems that were necessary. It was also good to have a base when testing new algorithms because the whole system could be simulated and the retrieved signal could be compared to the transmitted signal. If the receiver was implemented from scratch this would only have been possible after the design was working correctly.

Changing the design first in Simulink and then switching over to DSP Builder might have been an unnecessary step to take. It could have been better to use DSP Builder directly from the start but because the base for the receiver was in Simulink it was easier to continue using Simulink until a final design had been implemented.

The choice to measure the power consumption of the whole system with a fix setup and then change a certain algorithm makes it a bit harder to estimate how much the power consumption has been lowered by using different algorithms.

It would have been possible to implement the receiver with what could have been assumed to be the most power consuming algorithms and then lower the power consumption from this setup. This would of course give the final design a much lower power consumption when comparing the final design to the first design. But to intentionally create a bad design first does not give an accurate result of what is actually accomplished.

Another way to measure the difference in power consumption between the algorithms would have been to isolate each subsystem and only measure the power of that subsystem. This would be good if only the algorithms were the focus and not the whole system. Changing one part of the whole system will probably affect the following parts of the system and could slightly change the power consumption

of these system. The power consumption could be affected because it is affected by the input to the system.

Therefore the choice to implement a good system first and then evaluating the whole system was made. This makes it possible to see how the different changes affect the whole system and that the chosen algorithms were correctly chosen to reduce the power consumption.

6
Conclusion

This chapter discusses the conclusions that can be drawn from the results gained. After the discussion future work for the system is presented.

6.1 Discussion

Most of the results were as expected although the gain from some changes were larger than expected. As can be seen when comparing the bit reduction it reduces the power consumption with 28% which is a large reduction. When comparing the power consumption before and after the clock reduction it also provides a large power reduction of 44%. Therefore it seems important to choose algorithms that allows a lower clock rate to be used and a smaller amount of bits.

When comparing the gain from the different algorithms the power was often reduced about 5-15 mW which adds up if multiple power efficient algorithms are used. Therefore the conclusion can be made that reducing the clock and number of bits is the first important step to take when reducing the power consumption. When the clock rate and number of bits cannot be reduced any more then it is important to compare different algorithms.

As can be seen when comparing the two implementations of Luise algorithm the power consumption can be reduced by reducing the number of arithmetic operations. Three divisions are exchanged for one division through bit shifting which yields a reduction in the power consumption from 2.12 mW to 1.91 mW. A similar effect can be seen when comparing the different values for αMax+βMin. When switching out a division to a bit shift the power consumption is reduced from 1.90 mW to 1.84 mW. Through this it can be seen that the power consumption can be reduced by reducing the number of arithmetic operations and through

choosing appropriate values that can be implemented with bit shifting instead of multiplication and division.

6.2 Future Work

This section presents what work there is left to be implemented and tested in the radio receiver. The section starts with going through unimplemented work that there was not enough time to implement. The section then goes through different methods that could lower the power consumption but there was not enough time to test.

6.2.1 Unimplemented work

There are still some subsystems that shall be implemented in the receiver. The first two are OFDM and FHSS. Some of the theoretical background for these have been covered in the report but they are not yet implemented.

6.2.2 Lower Power Consumption

In the final receiver there are still some predefined blocks that could be implemented with a different design which could lower the power consumption.

There was not much time spent on designing the filter for a lower power consumption and a predefined block was used to create the filter. By implementing the filter from scratch and testing different designs, the power consumption would probably be lowered.

With the final design the clock was only lowered after the decimation filter. But before the decimation filter the samples are reduced to four samples and therefore the clock rate at the decimation filter could be reduced. As seen in the result reducing the clock is an efficient method to reducing the power consumption.

Another possible way to lower the power consumption would be to test different implementations to sum and multiply values. In reference [23] an implementation of multiplication with complex numbers is presented. Even if only a small gain per computation is gained through this there are many computations which would yield an overall larger decrease in power consumption.

Bibliography

[1] J. Mitola, "Software radios: Survey, critical evaluation and future directions," *IEEE Aerospace and Electronic Systems Mag.*, vol. 8, no. 4, pp. 25–36, 1993.

[2] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Commun. Surveys & Tutorials*, vol. 12, no. 4, pp. 531–550, 2010.

[3] R. V. Nee and R. Prasad, *OFDM For Wireless Multimedia Communications*. Artech House Publishers, 2000.

[4] E. Grayver, *Implementing Software Defined Radio*. Springer, 2013.

[5] SDRF, "SDRF cognitive radio definitions working document, SDRF –06-R-001-V 1.0.0,"

[6] H. Zimmermann, "OSI reference model–the ISO model of architecture for open systems interconnection," *IEEE Trans. on Commun.*, vol. 28, no. 4, pp. 425–432, 1980.

[7] E. G. Larsson, *Signals, Information and Communications*. Liu Press, 2012.

[8] I. A. Glover and P. M. Grant, *Digital Communications*. Prentice Hall Europe, 1998.

[9] G. J. Miao, *Signal Processing in Digital Communications*. Artech House, 2007.

[10] L. Wanhammar and H. Johansson, *Digital Filter*. Linkoping University, 2000.

[11] M. Renfors and T. Saramaki, "Recurise Nth-band digital filters-Part II: Design of multistage decimators and interpolators," *IEEE Trans. on Circuits and Systems*, 1987.

[12] J. G. Proakis and M. Salehi, *Digital Communications*. McGraw-Hill, 2008.

[13] M. Luise and R. Reggiannini, "Carrier frequency recovery in all-digital

modems for burst-mode transmissions," *IEEE Trans. on Commun.*, vol. 43, no. 2/3/4, pp. 1169–1178, 1995.

[14] P. Calvo, J. Sevillano, I. Velez, and A. Irizar, "Enhanced implementation of blind carrier frequency estimators for QPSK satellite receivers at low SNR," *IEEE Trans. on Consumer Electronics*, vol. 51, no. 2, pp. 442–448, 2007.

[15] Y. Linn, "Robust M-PSK phase detector for carrier synchronization PLLs in coherent receivers: theory and simulations," *IEEE Trans. on Commun.*, vol. 57, no. 6, pp. 1794–1805, 2009.

[16] C. Dick, F. Harris, and M. Rice, "Synchronization in software radios. carrier and timing recovery using FPGAs," in *2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 195–204, IEEE, 2000.

[17] F. Gardner, "Interpolation in digital modems - part i: Fundamentals," *IEEE Trans. on Commun.*, vol. 41, no. 3, pp. 501–507, 1993.

[18] F. Gardner, "A BPSK/QPSK timing-error detector for sampled receivers," *IEEE Communications Society*, pp. 423–129, May 1986.

[19] Y. Linn, "A new NDA timing error detector for BPSK and QPSK with an efficient hardware implementation for ASIC-based and FPGA-based wireless receivers," in *ISCAS '04. Proceedings of the 2004 International Symposium on Circuits and Systems*, vol. 4, pp. IV–465, IEEE, 2004.

[20] S. Cui, J. An, F. Zhang, X. Xu, B. Cai, and Z. Wang, "Burst frame synchronization in low SNR," *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, 2013.

[21] F. Xiong, *Digital Modulation Techinques*. Arthech House, 2006.

[22] J. Lu, K. Letaief, J. Chuang, and M. L. Liou, "M-PSK and M-QAM BER computation using signal-space concepts," *IEEE Trans. on Commun.*, vol. 47, no. 2, pp. 181–184, 2002.

[23] R. G. Lyons, *Understanding Digital Signal Processing*. Prentice Hall, 2004.

Appendix

A

Detailed Descriptions of the systems

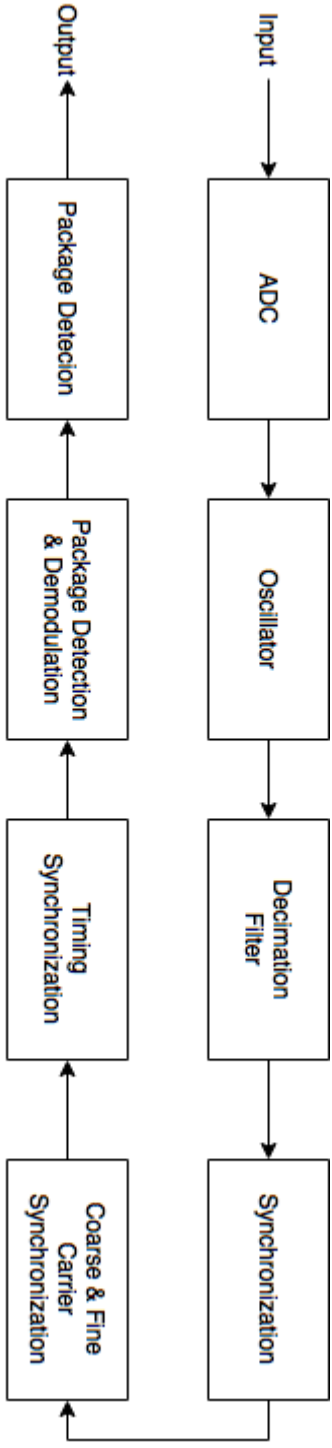

Figure A.1: *An overview how the subsystems are connected.*

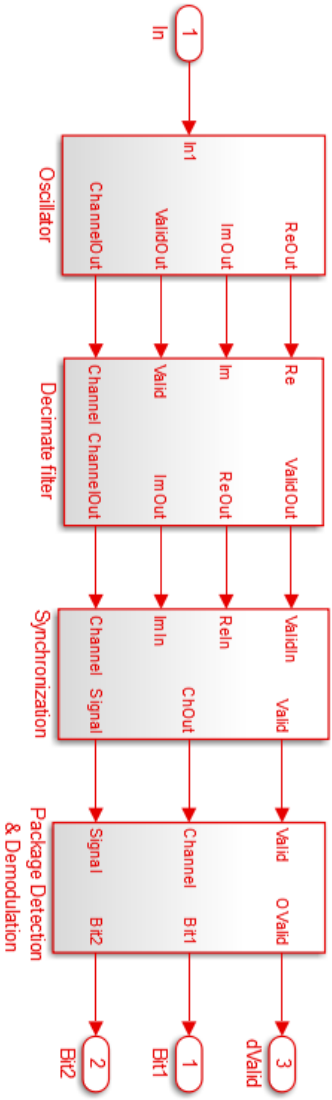

Figure A.2: *Overview of the whole receiver.*

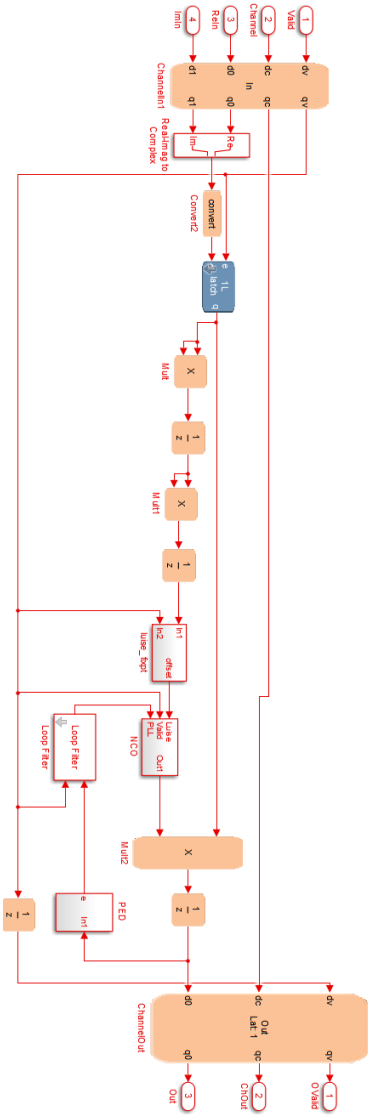

Figure A.3: *Overview of carrier synchronization.*

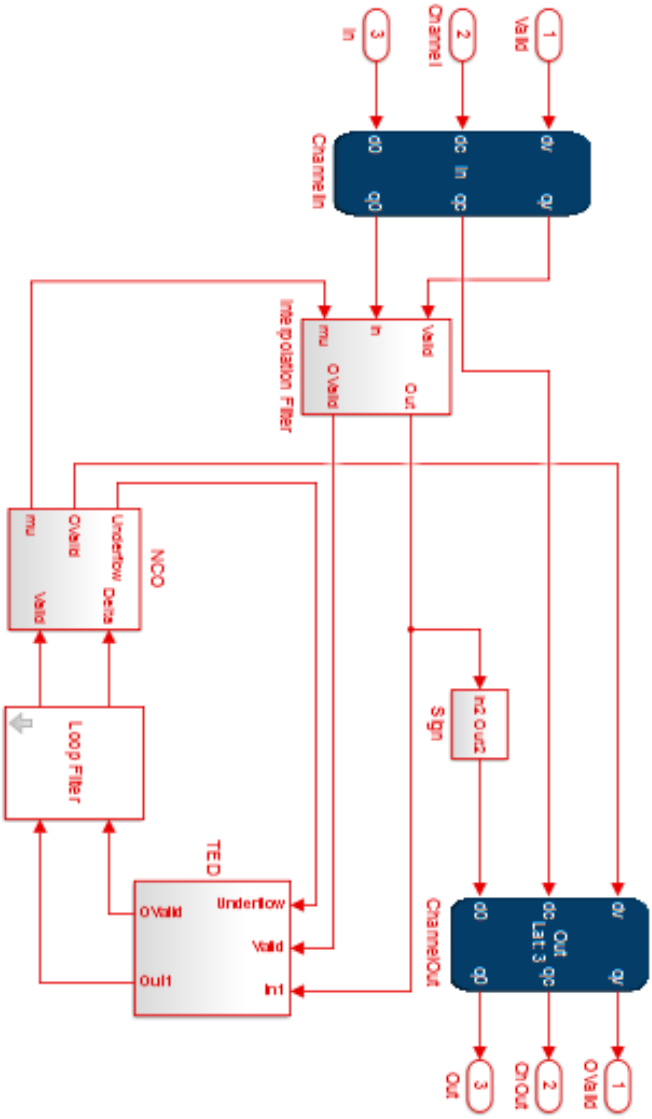

Figure A.4: *The time synchronization.*

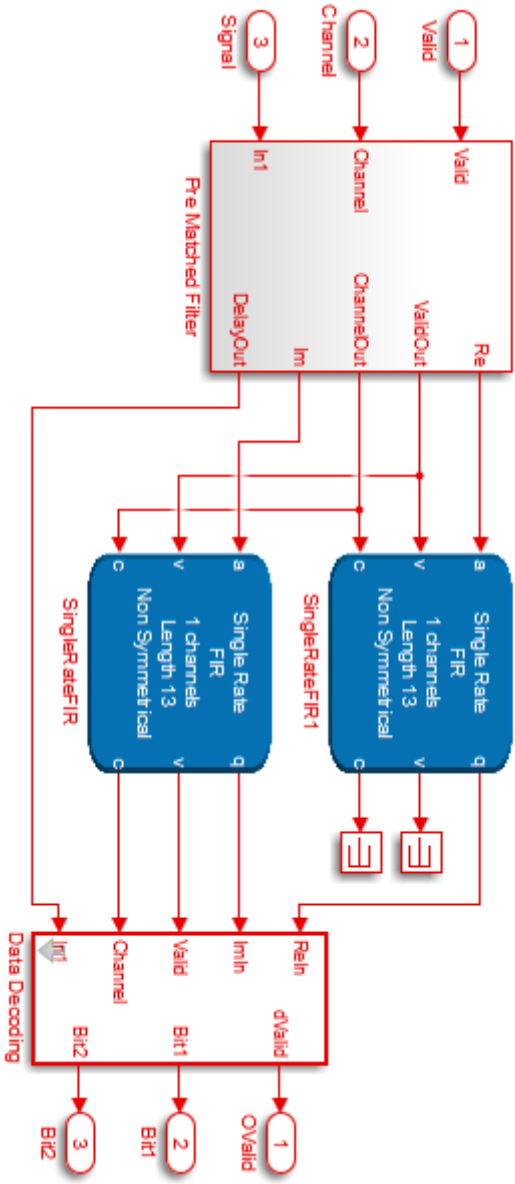

Figure A.5: *Overview of the matched filter and package detection.*

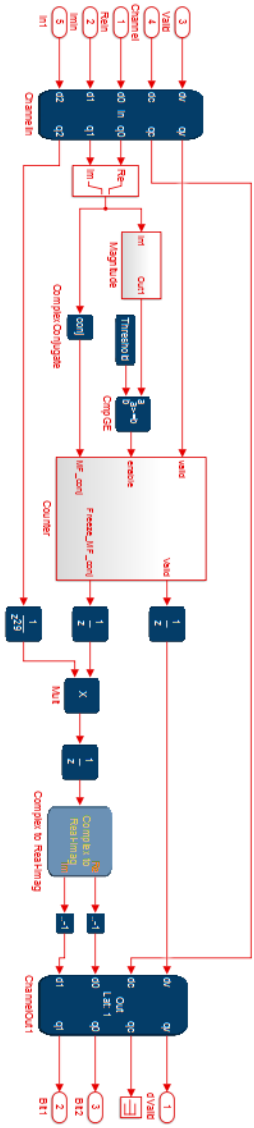

Figure A.6: *The package detection.*