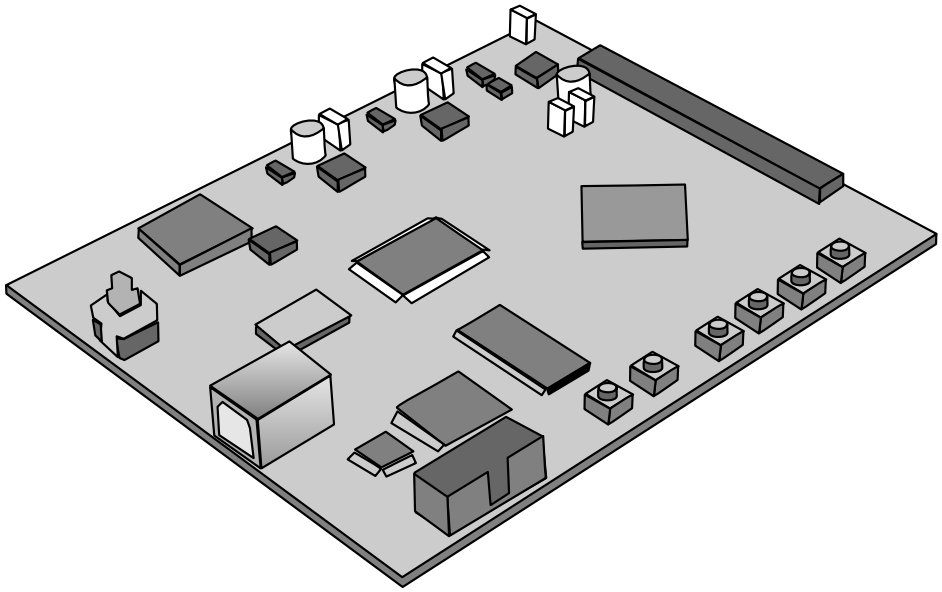
FEED-FORWARD QUADRATURE PHASE SHIFT KEYING FREQUENCY OFFSET CORRECTION

The development of a hardware-implementable phase error estimator algorithm for use in third generation mobile telephony systems



Master's Thesis

Authors:
KASPER LUND JAKOBSEN
KARL DAMKJÆR HANSEN

Supervisors:
YANNICK LE MOULLEC
JES TOFT KRISTENSEN

June 3, 2010

AALBORG UNIVERSITY

Aalborg University
Department of Electronic Systems
Frederik Bajers Vej 7
9220 Aalborg Ø
Telephone 96 35 86 00
http://es.aau.dk/

Title: Feed-Forward Quadrature Phase Shift Keying Frequency Offset Correction
The development of a hardware-implementable phase error estimator algorithm for use in third generation mobile telephony systems

Project term:
Master's Thesis, Fall 2009 and Spring 2010

Project group:
1043

Members of the group:
Kasper Lund Jakobsen
Karl Damkjær Hansen

Supervisors:
Yannick Le Moullec
Jes Toft Kristensen (Rohde & Schwarz)

Number of copies: 5

Number of pages: 72

Appendices and attachments:
6 and 1 CD-ROM

Completed 03/06 2010

Synopsis:

The objective of the project is to develop, analyze and implement a frequency drift compensation algorithm for use in 3G wireless transceiver systems.
It is shown that frequency drift severely degrades the performance of 3G wireless communication systems, thus making the compensation algorithm needed.
Three algorithms are analyzed for their individual properties and one of these are chosen for implementation. This algorithm consists of a phase estimator and a phase jump detector, where the phase estimator averages the phase error of previous input in a filter structure and the phase jump detector removes phase ambiguities created by the phase estimator. The original algorithm is developed for BSPK but as 3G utilizes QPSK, making an extension necessary. The extended algorithm is then simulated and analyzed with respect to fixed point and complex number representation. Based on these simulation it is concluded that the performance converges towards the theoretical performance of QPSK without frequency drift.
The Complexity of the algorithm is found to determine a suitable platform for the implementation. The algorithm is also pipelined and the inherent parallelism of the algorithm exploited to minimize the execution time. The platform chosen is the Altera Cyclone 3 FPGA. A test system for finding the performance of the algorithm is implemented onto the FPGA along with the algorithm. The performance of the implemented algorithm is shown to closely resemble the performance of the MATLAB simulations of the algorithm. The developed algorithm is therefore considered working and suitable for prototype use in 3G transceiver systems.

Preface

This project is conducted as a Master's Thesis project on the Applied Signal Processing and Implementation (ASPI) specialization at the Department of Electronic Systems at Aalborg University. The project is mainly targeted the people within the fields of signal processing, wireless communications and algorithm implementation.

The project concerns the development, analysis and implementation of a frequency drift compensating algorithm. Wireless communication systems performance is degraded because of frequency thereby making an algorithm which compensates for this problem of interest.

The project is based on a project proposal made by Jes Toft Kristensen at Rohde & Schwarz regarding "Digital Receivers and Carrier Frequency Drift". Initially the project concerned the 3G system, but the developed algorithm can be utilized in many other wireless communication applications.

It is interesting to note that this project, unlike most conducted at the university, produced positive results. In fact, we have worked our way through a full design process, right from the beginning with a specification over the initial information search and development of an algorithm to the end with the implementation and test of the system. We think that it has been exiting to see the intangible math written on the black boards the late evenings in the fall 2009 turn into a working implementation sitting here on the table blinking away with its "Ready-to-Run" LED today almost one year later.

Although we managed to complete the design process, it hasn't always gone as planed. We didn't really know the tools for synthesizing and simulating our VHDL code. (We didn't really know how to write VHDL either.) Also, we didn't have a project manager with the sole purpose of keeping us on track. But we managed to figure it out and finish on time. It gives us great pleasure to find out that we are indeed able to do things and get things done... On our own.

Of cause our supervisors have made sure that we didn't step too far of the path, and has always been keen on giving us feedback. Indeed, we haven't used your full potential. We owe you thanks for your support. Specifically a thanks is given to Jes for helping out with VHDL and ModelSim.

We would also like to thank our sister group during the fall 2009, Pradeep Silpakar and Boris Sala for shared work and discussions on the subject of frequency estimation.

In the end we would like to thank all of the students at ASPI 4 - 2010. Your have contributed to many pleasant and fun-filled days, both at the university and in private.

Kasper Lund Jakobsen

Karl Damkjær Hansen

Contents

Chapter 1

Introduction

Wireless communication has over the years been a very attractive choice when developing a communication system because of its wireless nature. The systems that utilizes the wireless communication technology are anything from GPS to mobile phone communication e.g. GSM or 3G. As the name indicates the main goal of wireless communication is to transfer data from one terminal to another without the use of wires between them. To be able to make this wireless link the data is first encoded which makes it more robust to transmission errors. The new data stream is then modulated to some waveform, which is then finally transmitted. In the receiver the inverse happens, which then should reconstruct the transmitted data. This is a short and simplified version of how a wireless link between two terminals work and in Figure 1.1 a block diagram of wireless transmission is shown.

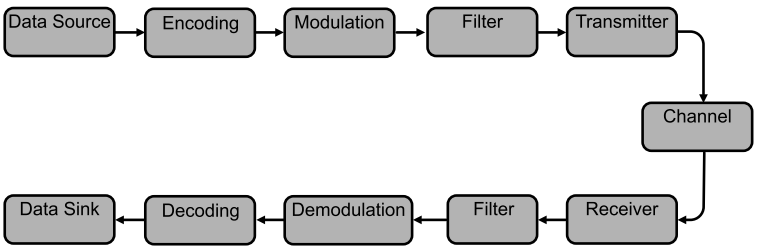


Figure 1.1: *Simplified wireless transmission model.*

Wireless communication is an interesting topic because it is so wide spred and widely used, which makes the technology interesting to improve, but also hard to improve because of the extensive research that has been going on in this technology.

Of the many different applications of wireless communication 3G is chosen for the topic of this project, because of its wide spred usage and because of the interest in this specific topic is high.

1.1 3G Mobile Communication

This section briefly describes 3G Mobile communication and gives a precise description of the scenario of focus for the project.

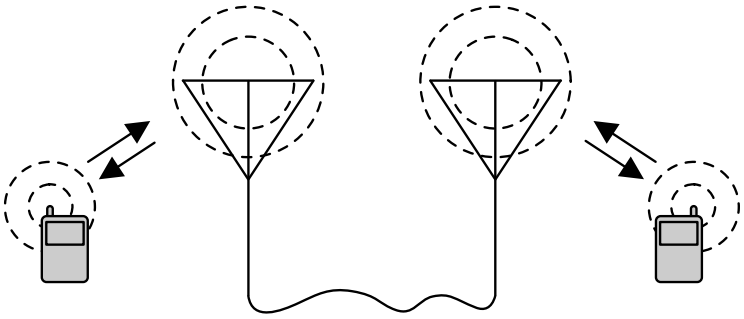


Figure 1.2: *3G phones communicating with use of basestations.*

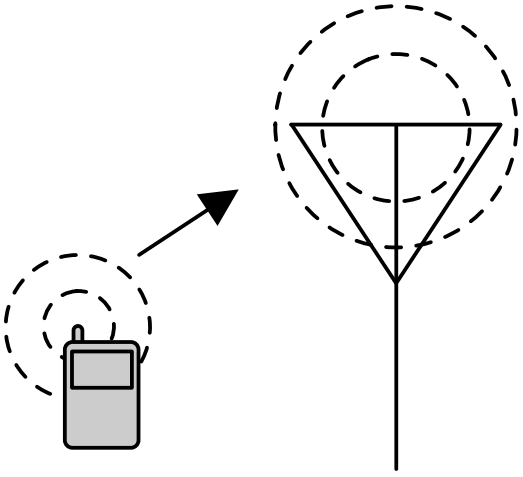


Figure 1.3: *The uplink from mobile to base station scenario.*

The 3G network uses base stations (BS) to communicate between two mobile phones. The setup can be seen in Figure 1.2. As seen the signal from one phone is first transmitted to the first base station (BS) which relays the signal to another which maintains the connection to the other mobile phone, and thereby the connection between the two phones is established. This also gives different scenarios to focus on, namely the connection between phone and BS and the connection between BS'. Not only are the scenario important, but also the direction of the communication. This is because the transmission differs for each scenario and direction. The direction and scheme for this project is the uplink scenario from the mobile phone to the BS which is shown in Figure 1.3.

To transmit data between two terminals in digital form, the data is encoded to a continuous waveform which is then transmitted, probably distorted by the channel and finally received and decoded. The encoding is done such that the data is represented with a sine wave with some carrier frequency, which is altered in phase to represent a binary zero or one. A detailed description of this is given in Section 3.1. The channel does, however, distorts the

signal. This distortion can have multiple effects on the sine signal; it can change the phase, frequency and amplitude of the signal. Of the three this project concerns the frequency distortion, also known as frequency drift. As shown in [7, p.424] this problem degrades the performance of PSK systems in terms of BER (Bit Error Rate), which is the case for 3G (see Section 3.1).

Because the 3G transmissions are well defined along with the structure of the front ends as little change in these transmissions would be preferred. The suggested solution to the problem should be implementable in as many already existing devices as possible without changing the structure of the system.

1.2 Problem statement

As mentioned in the previous sections the transceiver system which the project focuses on is 3G and the scenario is the uplink from mobile phone to base station. The frequency drift problem that was more specifically of interest, as it has been found to degrade the performance of transceiver systems. As the 3G system also has been found to be of interest due to its wide usage and therefore it is interesting to analyze how exactly frequency drift influences 3G signal. Given that the solution to this problem should be able to work within the structure and form of the 3G system this gives the restriction that the structure 3G system can not be changed. The structure restriction makes it suitable to develop a solution to the frequency drift problem in the digital domain such that the RF frontend used in the 3G network is not changed.

Given these choices the project system is shown in a typical transceiver system block diagram, which is seen in Figure 1.4.

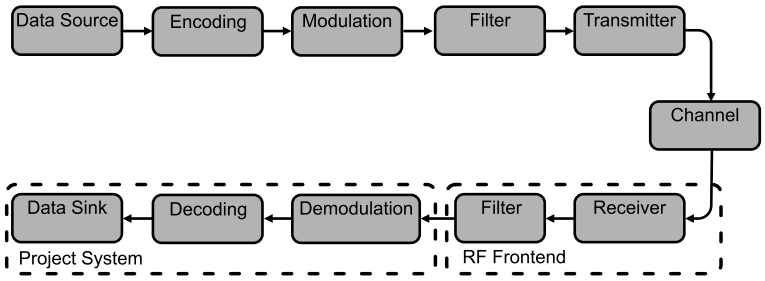


Figure 1.4: *The project focus within the simplified transmission model diagram shown in Figure 1.1.*

Combining frequency drift, 3G and digital signal processing into the hypothesis, which is the focus of the project, it becomes:

Hypothesis: *It is possible to analyze, design and implement an algorithm, which minimizes, with respect to constraints, the effects of frequency drift in 3G transceiver systems.*

Here minimizes refers to reduce the effects frequency drift has on the signals in 3G in terms of BER , such that the performance of the algorithm makes the transmission converges towards the performance of the system without frequency drift. This is done under the constraint that the before mentioned structure is not changed, thus making the algorithm easily implementable in existing transceiver systems.

This thesis involves analyzing the causes and effects of carrier drift, to be able to show where the main problems are and where the optimal place is to resolve the problems caused by frequency drift. Another aspect is the analysis of available algorithms to find measures for testing and evaluate the designed algorithm and to find ways of approaching the problem. With this knowledge the algorithm is designed, implemented and evaluated.

1.2.1 Sub Goals

The sub goals of the project are defined to keep the focus of the project. The goals are listed below and along with a brief discussion.

- Analysis of the frequency drift problem
  The analysis gives knowledge about the problem and which existing algorithms that can be used to solve the problem. The analysis i based on the results of the brief analysis of 3G communication such that the problem is analyzed with respect to the technologies used in 3G.
- Analysis of the interfaces to the 3G system
  This analysis gives knowledge about how to interact with the 3G system such that the developed algorithm can be implemented.
- Analysis of compensation algorithms
  An analysis of the known methods of compensating for the effects of frequency drift gives information about the common aspects that needs attention when designing and/or implementing an algorithm of this type.
- Development of an algorithm for implementation
  With the algorithms analyzed a new solution can be developed, simulated and analyzed such that it can be implemented.
- Implementation of the algorithm
  This refers to the implementation of the algorithm.
- Testing the algorithm
  Testing the implemented algorithm shows whether the implementation is in accordance with results from the analysis of the algorithm.

With the topics of interest for the project defined the topics which are not considered are described.

1.2.2 Limitations

The limitations explain what subjects that are not considered and the assumptions that are made in the project.

As an entire transceiver system for 3G communication would be too comprehensive, the project is limited. This is somewhat already done as the focus is the uplink scenario from the mobile phone to the BS. The other limitation which has been taken is that the system is only developed as a digital processing system to keep it in the scope of the learning goals for the project. Further the system only concerns the frequency drift compensation algorithm, such that other blocks in the transceiver system is not implemented.

A signification limitation or assumption for the project is that only an AWGN channel is considered during the implementation of the system. Normally wireless signals, especially in urban areas, would be subject to a multipath channel where replicas of the signal is shifted in time but also received. This can, however, be investigated when the system is designed such that the performance under these conditions can be documented.

The final limitation is regarding the platform. FPGAs are chosen as the platform of the project, because of the interests of the project group. Due to prior knowledge about Altera FPGAs and the availability of these FPGAs at the university the project only concerns these platforms.

Chapter 2

Methods

To prove or reject the hypothesis described in Section 1.2 the methods utilized is defined. This structures the project and keeps the focus of the project.

2.1 A^3

To structure the development of the system, the A^3 model is used. The model is not described in literature outside the university, but is very much used at Aalborg University. Because of this, the description of the model presented here is based on lectures given at Aalborg University.

The model, which is seen in Figure 2.1, divides the development of a system into three domains, namely the; Application, Algorithm and Architecture domains. First the application of the system that is to be developed is analyzed and specified, which results in choosing a suitable application in the application domain. With this choice of application, the possible algorithms which fulfills the requirements and functionality of the application is analyzed. With the analysis, choices and compromises can cause the application to change which then makes the mapping from application to algorithm iterative (Figure 2.2 line 1*).

The characteristics of the algorithm is then analyzed in terms of complexity and resources

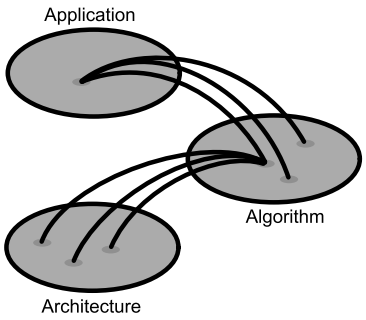


Figure 2.1: *The A^3 Model without iterations. Here the three domains are illustrated and the possible mappings.*

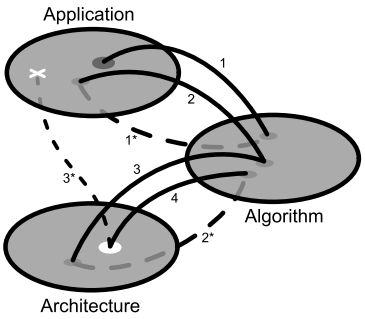


Figure 2.2: *The A^3 Model with the possible iterations marked as dashed lines.*

needed to find a suitable architecture. Here the choice of the architecture could also change the algorithm making this mapping iterative (Figure 2.2 line 2*).

The final thing that can happen after the choice of architecture is that the architecture would be suitable for a similar application or it could refine the application and algorithm resulting in a an all new iteration (Figure 2.2 line 3*).

The way this model is used in the project is depicted by the structure of the project, which is divided into the 3 domains of the model. The different domains are analyzed before the mapping from one domain to another is made. The application part (Chapter 3) describes the problem in detail and the system in which this problem arises. The algorithm part (Chapter 5) describes different algorithms which solves this problem, and an analysis of these aides the choice of architecture. Finally, the architecture part (Chapter 7) describes how the algorithm is implemented onto on the chosen platform forming an architecture.

2.2 Implementation Methodology

The A^3 model is used to structure the project, but it does not specify the way of mapping from one domain in the model to another. Because of this, a method of doing so, has to be defined. The method described here is used for the mapping from the algorithm domain to the architecture domain from a functional point of view. This means that the method describes how the functionality of the algorithm is mapped to the architecture.

The method used is based on the V-model, with its downward direction in design phase where it continuously refine the design and its upward direction where it integrates and validates. The model used here first defines the algorithm as a whole including how this algorithm should be tested as a whole. The algorithm is then divided into blocks, which is then again defined along with the tests of these.

After the algorithm has been chosen it is thoroughly analyzed thereby defining the algorithm. This analysis can lead to small changes, which can be thought of as being searching the neighbor algorithms in the algorithm domain in the A^3 model. After this the algorithm is analyzed for its feasibility to be implemented in terms of complexity and fixed point representation of numbers. This can again lead to a new point in the algorithm domain which is then finally chosen for implementation. With this choice the platform it is to be implemented on is found. The platform that is to be chosen must accommodate for the

complexity and number representation found in the analysis, which is needed in order to implement the algorithm.

With the platform chosen the interaction with the system is determined which then makes it possible to test the system. The test is used to determine if the performance of the system is as expected. Now the algorithm is divided into different blocks which each maintains a specific functionality. The division is based on the nature of the algorithm and not a predetermined method. With the division the interfaces from one block to another is also defined such that these is tested for their individual functionality. With all the blocks defined and testable the implementation of them begins. During the implementation process the blocks are continuously tested to remove unwanted behavior at an early stage. When the implementation of the blocks are done they are ready for integration where the blocks are combined back to form the algorithm as it was before the division. The implementation is now tested as a whole and with this test passed the implementation is considered done.

The method is depicted in Figure 2.3 and as seen there is a high similarity with the V-model.

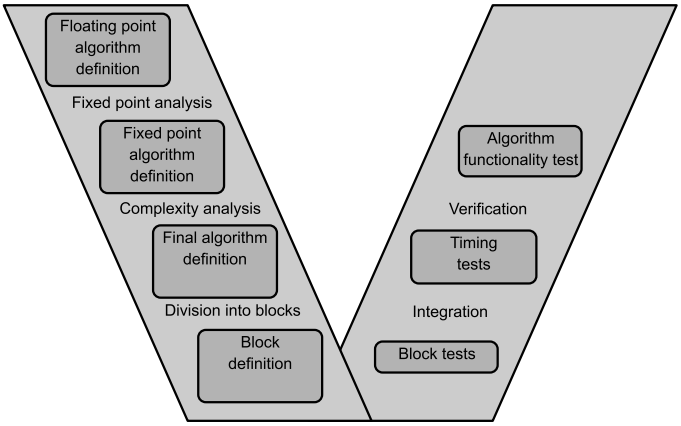


Figure 2.3: *The modified V-model which is used in the project.*

Chapter 3

Problem Analysis

To be able to find solutions to the problem described in 1.2 the problem has to be analyzed. This analysis focusses on the theory of wireless communication (applicable to the 3G system) and also includes an analysis of the mathematical implications of frequency drift, which derives the performance degration ferquency drift causes.

3.1 Frequency Drift

To determine the implications frequency drift has on wireless communication in general the modulation form has to be analyzed. The following descibes the modulation used in digital wireless communication which is a prerequisite for determining the BER for the modulation scheme used in 3G both with and without frequency drift.

Phase modulation (PM), which is used in 3G [11, p.33], can be considered as a special case of frequency modulation (FM), or it can be derived independently. Here the modulation is described as being an independent type of modulation, as going from FM will involve integration of the frequency components, most likely making the derivations more difficult to understand.

Just as FM is widely used for transmission of analog signals e.g. commercial radio, PM can also be used for analog signals. When using these modulations for transmission of digital signals they are called frequency- and phase-shift keying (FSK and PSK respectively). In essence PSK can be obtained by using a plain PM but letting the modulating signal take on only binary values.

The most simple form of PSK is the binary-PSK (BPSK). It will modulate the bits onto the carrier by changing the phase of the carrier 180°. All the receiver needs to do is to synchronize with the phase of the transmitter, and check if the phase is the same as the local oscillator or the opposite. When sampling the signal and plotting it in the complex plane, it will either fall in one of two message points, see Figure 3.2.

Quadrature-PSK (QPSK), which is the modulation in focus for the project [11, p.33], is essentially the same as BPSK. The difference is that QPSK has four message points instead of the two of BPSK. This is accomplished by varying the phase of the carrier in steps of 90° instead of 180°. Because of the double number of points, each point represents two bits. How to map the two bits from the bit stream to the message points differs between

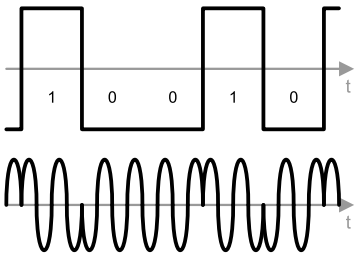


Figure 3.1: *An example of phase shift keying (specifically BPSK). Notice the 180° phase shifts when the data changes value.*

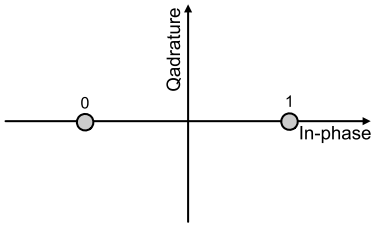


Figure 3.2: *The BPSK message points in the complex plane.*

implementations. Usually a Gray code mapping is employed to make sure that the points closest to each other only has one bit in difference[10, p.173], this minimizes the risk of bit errors due to noise. The signal constellation for QPSK is illustrated in Figure 3.3.

The points of the constellation does not need to lay exactly on the axes like in Figure 3.3. In Universal Mobile Telecommunications System (UMTS) standards [11] the imaginary and the real bits are treated individually. This is in essence two individual BPSK processes, one on a sine wave and one on a cosine wave. By doing this the message points are rotated 45°, as seen in Figure 3.4, which is the signal constellation used in 3G [11, p.25].

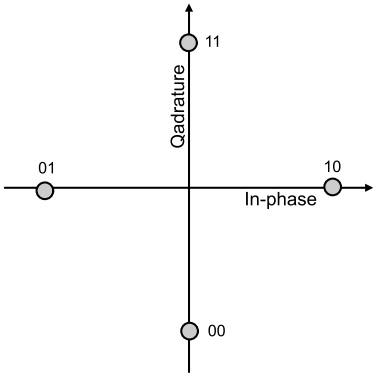


Figure 3.3: *The message points in the conventional QPSK constellation uses Gray coding to minimize BER due to noise.*

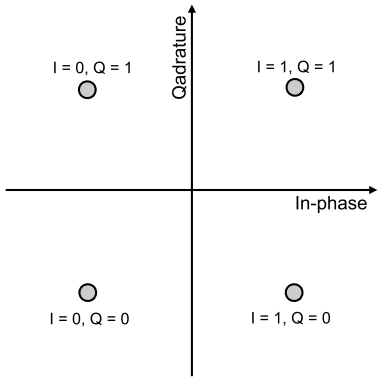


Figure 3.4: *Signal constellation of the QPSK used by 3G. The quadrature and the in-phase bits are treated individually, but in essence, it is the same as conventional QPSK.*

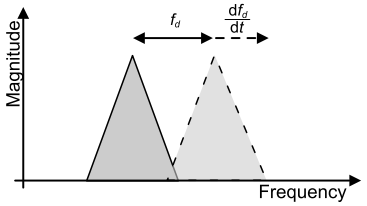


Figure 3.5: *The frequency offset between the transmitter and the receiver f_d can be caused by many things. The change in frequency $\frac{df_d}{dt}$ is called the drift.*

3.1.1 Drift Calculations

Assuming a frequency offset f_d between the transmitter and the receiver[1]. The entire offset is modeled as coming from the transmitter. This can be convenient, as the receiver will most likely also suffer from a degree of frequency offset which it cannot know of. But as it has knowledge of its own frequency, this can be regarded as the "ideal" frequency, and only the transmitter suffers from frequency offset. The frequency offset is illustrated in Figure 3.5. The frequency offset is not necessarily constant, the change of the offset is called the drift.

The transmitter modulates the information sequence $I(t)$ onto the offset carrier in (3.1) and

[1]The term f_d has been chosen for frequency offset, as f_o might be interpreted as some initial frequency. It does not refer to frequency drift as that refers to the change in f_d

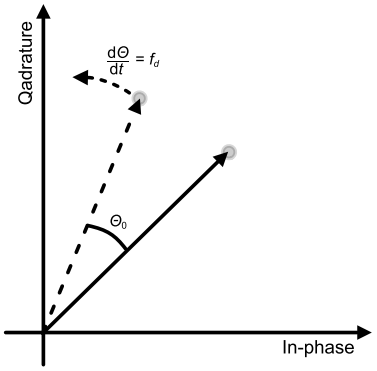


Figure 3.6: *The top right quarter plane of the QPSK signal constellation. The phase might be offset because of the distance between the transmitter and the receiver. An offset frequency f_d will move the phase over time.*

the receiver mixes the received signal with its local oscillator in (3.2).

$$\begin{aligned} k &= 2\pi (f_c + f_d) t + \pi I(t) \\ l &= 2\pi f_c t \\ s &= \cos(k) \qquad (3.1) \\ x &= \cos(k) \cdot \cos(l). \qquad (3.2) \\ &= \frac{1}{2}\left(e^{ik} + e^{-ik}\right) \cdot \frac{1}{2}\left(e^{il} + e^{-il}\right) \\ &= \frac{1}{2^2}\left[e^{i(k+l)} + e^{i(k-l)} + e^{-i(k+l)} + e^{-i(k-l)}\right] \\ &= \frac{1}{2}\left(\cos\left[2\pi (2f_c + f_d) t + \pi I(t)\right] + \cos\left[2\pi f_d t + \pi I(t)\right]\right) \qquad (3.3) \end{aligned}$$

The mixing process produces two new modulations of $I(t)$ as seen in (3.3), one at double the carrier frequency plus the offset frequency and another at the offset frequency. By low pass filtering the signal the high frequency component can be neglected. The resulting signal y in (3.4) would have been the pure information sequence, had the frequency offset been zero.

$$y = \frac{1}{2}\cos\left[2\pi f_d t + \pi I(t)\right] \qquad (3.4)$$

This offset turns the signal constellation at a speed of the offset frequency as illustrated in Figure 3.6.

The phase might initially be offset by the distance delay and and the fact that the receiver does not know of the state of the transmitter oscillator. This initial phase is denoted θ_0 in Figure 3.6.

3.1.2 Noisy Channel

As with all wireless communication noise will corrupt the signals. An example of a constellation plot of a signal corrupted by additive white Gaussian noise (AWGN) is illustrated in Figure 3.7.

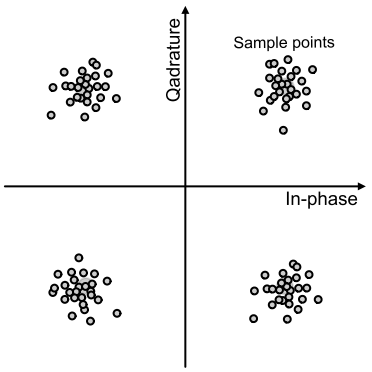


Figure 3.7: *When noise corrupts the signal the signal points will vary around the transmitted signals message points.*

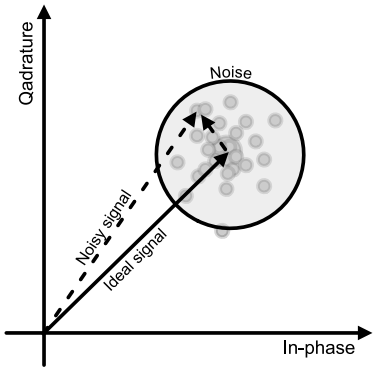


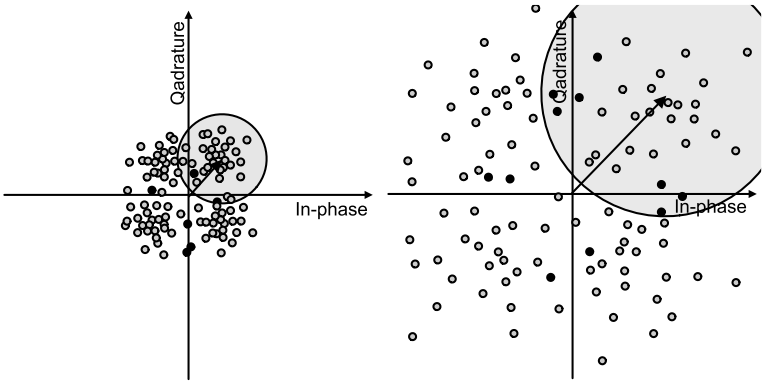
Figure 3.8: *When interpreting the signals as vectors, the noise is merely a vector being added to the ideal signal. The average noise energy is illustrated as the radius of the gray noise circle.*

The distance from the origin to a sample point equals the energy received at that point. Thus the signals can be interpreted as vectors as illustrated in Figure 3.8. This way the noise itself is a vector that is added to the signal.

Low signal-to-noise ratio (SNR) will lead to more bit errors. Two factors influence the SNR, namely the energy of the signal and the energy of the noise. Lowering the former or increasing the latter will lead to lower SNR, see Figure 3.9. When combining the phase offset with low SNR, the BER rises. This is illustrated in Figure 3.10

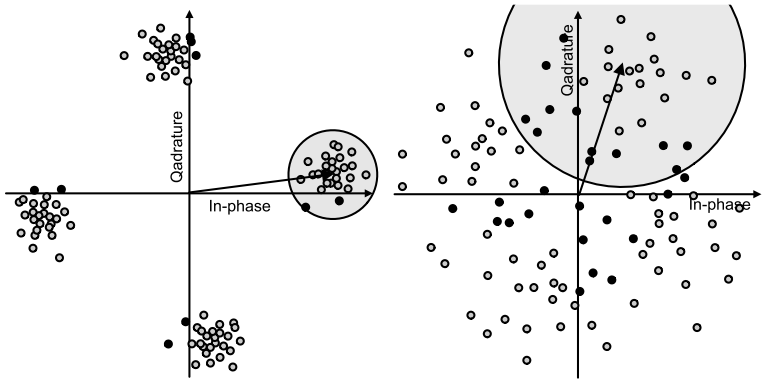
3.1.3 Bit Error Rate for QPSK Signals

In order to evaluate how a QPSK system performs when frequency drift is present the theoretical bit error rate (BER) is calculated both for the scenario without frequency offset and with frequency offset. With these calculations it is possible to show how the frequency drift degrades the performance of QPSK. The section is inspired from [6, p.84 - 102] and [5, p.9 - 23].



(a) *Low signal energy* **(b)** *High noise energy*

Figure 3.9: *Low SNR can occur when either the signal energy is low (a) or the noise energy is high (b). The signal points painted black are bit errors.*



(a) *High SNR, but severely offset phase.* **(b)** *Low SNR and phase offset.*

Figure 3.10: *Phase offset produces bit errors (a), but when combined with low SNR (b) the BER rises. The signal points painted black are bit errors.*

BER without Frequency Offset

To calculate the theoretical BER the channel over which the data is transmitted has to be specified and here the AWGN channel is assumed. Further, the signal constellation is assumed equal to be the one shown in Figure 3.4, grey encoding and the a priori probability of a 1 or 0 is assumed equal.

The probability of an error and wrong decoding is described as:

$$P(\hat{U} \neq U \mid X = s_1) \tag{3.5}$$

Which is the probility that estmated bit $\hat{U}$ is not equal to the transmitted bit U given that transmitted signal X was s_1.

As seen from the Figures of the signal constellation diagrams presented earlier (Figures 3.7, 3.9) the decision boundaries are the axis'. By using this and the fact that the distance from the signal point to origin is the symbol energy $\sqrt{E_s}$, the BER is found. It is only necessary to calculate the BER for one signal point due to symmetry in the constellation diagram and equal a priori probabilities.

It can be shown using Pythagoras that the message point in first quadrant has the coordinates $\left(\sqrt{\frac{E_s}{2}}, \sqrt{\frac{E_s}{2}}\right)$. This means that in order for one bit to be decoded wrong ($\hat{U} \neq U$) the noise components n_1 and n_2 in at least one dimension has to be bigger than $\sqrt{\frac{E_s}{2}}$. Because of the AWGN channel, the noise is normally distributed with $\mathcal{N}(0, \frac{N_0}{2})$. Using this, the the probability of error is found as:

$$P(\hat{U} \neq U \mid X = s_1) = \frac{1}{2} P\left(n_1 > \sqrt{\frac{E_s}{2}} \text{ or } n_2 > \sqrt{\frac{E_s}{2}}\right) \tag{3.6}$$

Assuming that n_1 and n_2 are independent and they have zero mean, this gives:

$$P(\hat{U} \neq U \mid X = s_1) \quad = \quad \frac{1}{2}\left(P\left(n_1 > \sqrt{\frac{E_s}{2}}\right) + P\left(n_2 > \sqrt{\frac{E_s}{2}}\right)\right) \tag{3.7}$$

Normalizing this with the variance of the noise this gives:

$$P(\hat{U} \neq U \mid X = s_1) \quad = \quad \frac{1}{2}\left(P\left(\frac{n_1}{\sqrt{\frac{N_0}{2}}} > \sqrt{\frac{E_s}{N_0}}\right) + P\left(\frac{n_2}{\sqrt{\frac{N_0}{2}}} > \sqrt{\frac{E_s}{N_0}}\right)\right) \tag{3.8}$$

This is the same as the cummalative distribution function ($\mathcal{Q}$) for one random variable with $\mathcal{N}(0,1)$ and thereby the error probability is found as:

$$P(\hat{U} \neq U \mid X = s_1) \quad = \quad \mathcal{Q}\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{3.9}$$

Here because 2 bits are transmitted per symbol $E_s = 2E_b$ where E_b is the energy transmitted per bit. Then finally the BER is given by:

$$\text{BER} = \mathcal{Q}\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{3.10}$$

$$\text{BER} = \mathcal{Q}\left(\sqrt{2\gamma}\right) \tag{3.11}$$

Where γ is the average transmitted signal-to-noise ratio (SNR) per bit.

Then applying the frequency drift to the calculations the degration should be detectable which is calculated in the following section.

BER with Frequency Offset

To calculate the BER for this case the same assumptions are made, as for the calculations without frequency offset namely; an AWGN channel, equal a priori probabilities of a 1 and a 0 and the signal constellation shown in Figure 3.4.

It has previously been shown that the signal output from the mixer for the in phase channel when frequency offset is present is as follows (Section 3.1.1) (here the symbol sent is 00):

$$z_i = \sqrt{\frac{E_s}{2}}\cos(2\pi f_d t) - j\sqrt{\frac{E_s}{2}}\sin(2\pi f_d t) + n_1(t) \tag{3.12}$$

The same is the case for the quadrature channel:

$$z_q = \sqrt{\frac{E_s}{2}}\sin(2\pi f_d t) - j\sqrt{\frac{E_s}{2}}\cos(2\pi f_d t) + n_2(t) \tag{3.13}$$

As mentioned, to decode these signals, this result is integrated over the bit duration (T) which then is compared to zero, which are the decision boundaries. After the integration the signals become estimates of the I and Q values for the given signal as follows:

$$\hat{s}_i = \sqrt{\frac{E_s}{2}}\int_0^T \cos(2\pi f_d t)\,dt - j\sqrt{\frac{E_s}{2}}\int_0^T \sin(2\pi f_d t)\,dt + n_1 \tag{3.14}$$

$$= \sqrt{\frac{E_s}{2}}k_1(f_d) - j\sqrt{\frac{E_s}{2}}k_2(f_d) + n_1 \tag{3.15}$$

Where: $k_1(f_d) = \int_0^T \cos(2\pi f_d t)\,dt$ and $k_2(f_d) = \int_0^T \sin(2\pi f_d t)\,dt$.

The result for the quadrature channel is:

$$\hat{s}_q = \sqrt{\frac{E_s}{2}}\int_0^T \sin(2\pi f_d t)\,dt - j\sqrt{\frac{E_s}{2}}\int_0^T \cos(2\pi f_d t)\,dt + n_2 \tag{3.16}$$

$$= \sqrt{\frac{E_s}{2}}k_2(f_d) - j\sqrt{\frac{E_s}{2}}k_1(f_d) + n_2 \tag{3.17}$$

As mentioned, the real parts of the two values $\hat{s}_i$ and $\hat{s}_q$ i.e. $\Re(\hat{s}_i)$ and $\Re(\hat{s}_q)$, are compared to zero to determine which symbol was most probably sent. The real parts are used because the imaginary part does not move the point in the real direction (see [7, p.424]). Therefore the probability of an error $P(\hat{U} \neq U \mid X = s_1)$ is found as:

$$\begin{aligned} P(\hat{U} \neq U \mid X = s_1) &= \frac{1}{2} P\left(n_1 < -\Re(\hat{s}_i) \text{ or } n_2 < -\Re(\hat{s}_q)\right) && (3.18) \\ &= \frac{1}{2} P\left(n_1 < -\Re\left(\sqrt{\frac{E_s}{2}}\left(k_1(f_d) - jk_2(f_d)\right)\right) \text{ or } \right. \\ &\qquad \left. n_2 < \Re\left(-\sqrt{\frac{E_s}{2}}\left(k_2(f_d) - jk_1(f_d)\right)\right)\right) && (3.19) \\ &= \frac{1}{2} P\left(n_1 > \sqrt{\frac{E_s}{2}}\left(k_1(f_d)\right)\right) \\ &+ \frac{1}{2} P\left(n_2 > \sqrt{\frac{E_s}{2}}\left(k_2(f_d)\right)\right) && (3.20) \end{aligned}$$

As the noise (n_1 and n_2) are normally distributed with $\mathcal{N}(0, \frac{N_0}{2})$ and normalizing the above equations with variance of the noise this gives:

$$\begin{aligned} P(\hat{U} \neq U \mid X = s_1) &= \frac{1}{2} P\left(\frac{n_1}{\sqrt{\frac{N_0}{2}}} > \frac{\sqrt{\frac{E_s}{2}} k_1(f_d)}{\sqrt{\frac{N_0}{2}}}\right) \\ &+ \frac{1}{2} P\left(\frac{n_2}{\sqrt{\frac{N_0}{2}}} > \frac{\sqrt{\frac{E_s}{2}} k_2(f_d)}{\sqrt{\frac{N_0}{2}}}\right) && (3.21) \\ &= \frac{1}{2} P\left(\frac{n_1}{\sqrt{\frac{N_0}{2}}} > \sqrt{\frac{E_s}{N_0}} k_1(f_d)\right) \\ &+ \frac{1}{2} P\left(\frac{n_2}{\sqrt{\frac{N_0}{2}}} > \sqrt{\frac{E_s}{N_0}} k_2(f_d)\right) && (3.22) \end{aligned}$$

With this rewriting the BER is found using the cummulative distrubution function (the $\mathcal{Q}$ function) and the fact that $E_s = 2E_b$ which gives:

$$\begin{aligned} \text{BER} &= \frac{1}{2} \mathcal{Q}\left(\sqrt{\frac{2E_b}{N_0}} k_1(f_d)\right) \\ &+ \frac{1}{2} \mathcal{Q}\left(\sqrt{\frac{2E_s}{N_0}} k_2(f_d)\right) && (3.23) \end{aligned}$$

Again using $\gamma = \frac{E_b}{N_0}$ is the SNR per bit.

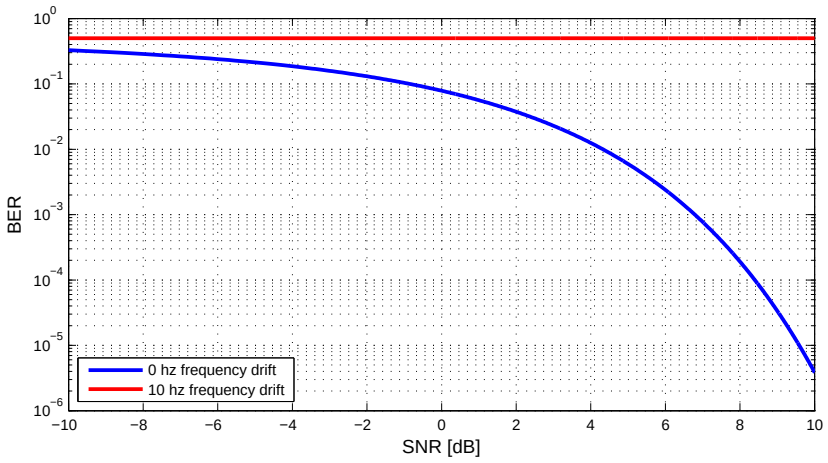


Figure 3.11: *BER plot for 0 hz frequency drift and 10 hz frequency dift.*

$$\begin{aligned} \text{BER} &= \frac{1}{2}\mathcal{Q}\left(\sqrt{2\gamma}k_1(f_d)\right) \\ &+ \frac{1}{2}\mathcal{Q}\left(\sqrt{2\gamma}k_2(f_d)\right) \end{aligned} \tag{3.24}$$

The plot of these BER curves is shown in Figure 3.11.

It is seen that because of the drift the message point in the constellation diagram will rotate and therefore the BER will on average be 0.5 no matter how big the frequency drift is. This is implies that even small frequency drifts will accumulate the phase error and give the system a BER of 0.5 even for a good SNR. The reason that the result is 0.5 and not 0.25 although there are 4 message points is that the result BER is calculated per bit and two message point will give a correct decoding of one bit. This means that the algorithm which should handle this problem is very important.

3.2 Maximum Frequency Offset

To simulate and design a system that compensates for frequency drift an estimate of the maximum frequency offset needs to be found.

The frequency offset can be caused by many different effects, therefore a rough estimate of the interval it can be in has to be found. This determines which frequency offsets the algorithm has to handle.

One of the causes for a frequency drift is the Doppler shift, that will arrise when the transmitter is moving towards or away from the receiver. As such the frequency shift is a function of the velocity difference between the transmitter and the receiver and the carrier frequency. The Doppler shift can be calculated as follows[13, p.525]:

$$f' = \left(\frac{v + v_t}{v - v_r}\right) f_c \tag{3.25}$$

where v is the speed of light, v_t is the speed of the transmitter, v_r is the speed of the receiver and f_c is the carrier frequency. Given, as an example, that a carrier frequency in the 3G communication system is 1937.6 MHz [12, p.17] and that the speed of light is approximately $3 \cdot 10^8$ m/s the frequency error in PPM (Parts Per Million) in shown in Figure 3.12.

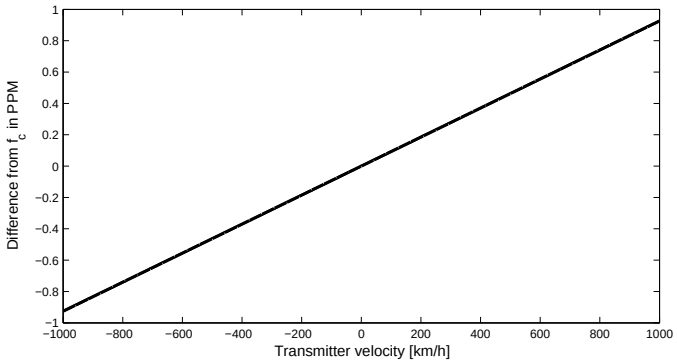


Figure 3.12: *Example of carrier frequency errors due to Doppler shift. Here the velocity of the receiver is 0.*

As it is unlikely that the velocity difference between the transmitter and the receiver exceeds 1000 km/h one PPM is chosen as the upper bound for the frequency Offset. The offset can however as shown also be negative and therefore the frequency offset range becomes $[-1\text{PPM} + f_c, 1\text{PPM} + f_c]$ with regard to the Doppler shift. This interval is in the case of the example above $[f_c - 1937.6, f_c + 1937.6]$ where $f_c = 1937.6$ MHz. The highest possible carrier frequency in 3G is found to be [12, p.17] 2567.6 MHz. Therefore the range maximally becomes $[2567597432.4;\ 2567602567.6]$ Hz which is chosen as the design parameter. This means that the algorithm is designed to compensate for frequency offsets of maximally 2500 Hz.

3.3 Interfaces

As previously mentioned in Section 1.2 the structure, of which the project system is to be implemented in, is of improtance. Therefore the place where the system can be implemented has to be defined. Defining this gives the interfaces to the algorithm, such that it is clearly defined under which prerequisites the algorithm has to work.

It is chosen not to implement an algorithm which alters the front-end of the transceiver system. The reason for this is that it would mean redesigning the the front-end to include another block, which as most frontends are single chips, would be difficult. Because of this the system designed in the project focus on the signal processing which takes place after the Analog to Digital Converter (ADC). This gives the benefits that the system can be easily implemented as a digital signal processing algorithm in near or at baseband and that it can

be implemented easily in existing systems, making it an atractive improvement possibility. By this choice and the context of 3G the interfaces for the project system is shown in Figure 3.13, with the project system being implemented in the "Baseband Detection" block.

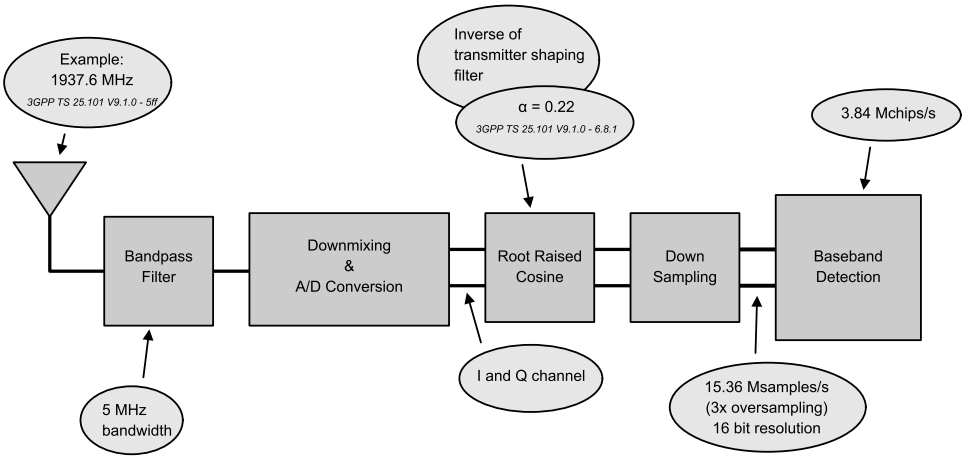


Figure 3.13: *Block diagram showing the 3G frontend with interfaces for the different blocks including the block showing the interface to the project system [12].*

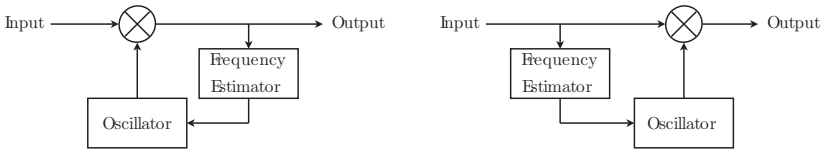
As seen by the figure the project system should handle 3.84 Mchips/s, meaning the input to the system is 3.84 Msamples/s each 16 bit in two channels.

Chapter 4

Algorithms

4.1 Feedback and Feed-forward Algorithms

Algorithms concerning this topic can be divided into two groups, namely feedback and feed-forward algorithms. The feedback algorithms uses iterations to compensate for the error in frequency, whereas the feed-forward algorithms simply compensates for the error based on the present and previous observations. Block diagrams of both feedback and -forward are shown in Figure 4.1.



(a) *Block diagram of feedback algorithms.*

(b) *Block diagram of feedforward algorithms.*

Figure 4.1: *Block diagrams of the possible classes of algorithms.*

As seen in Figure 4.1, the blocks used are the same in both cases. Because of this, the algorithms are different ways of performing the functionality needed in these blocks.

4.1.1 Frequency Estimator

This block is the most crucial part of the algorithms. The functionality that this block has is to find the frequency of the input signal and output this frequency. Estimating the frequency can be divided into two methods. These are data aided and non data aided estimators.

The data aided methods use training sequences in the input signal to be able to lock on to the phase and frequency of the signal. An example of such an algorithm is presented in [7, p.311]. These methods are, however, not of interest for the 3G scenario because there is only training sequences present when setting up the connection and as such the algorithms cannot track the frequency through an entire transmission.

The non data aided algorithms, which are of interest, include both feed-forward and feedback algorithms ([1], [2], [4]). Some estimate the phase error of the signal, while others

estimate the frequency error. Because of this, the algorithm which is considered here must either be an algorithm which estimates the phase varying over time i.e. the frequency.

4.1.2 Sine Oscillator

This block is essential for estimating the phase or frequency to be used for compensation. The block can, however, also be included in the estimation block, like done by Viterbi in [15]. This way the frequency estimator outputs an oscillating signal that is directly mixed with the input. If the block is not included, implementing the functionality of the oscillator is the same as implementing a sine and cosine function. These functionalities are usually implemented by means of the CORDIC algorithm [16] or look up tables, as these are efficient and easy to use.

Because the need for this block depends on whether the chosen estimation algorithm needs an external oscillator or not, other algorithms are not discussed before the estimation algorithm is chosen. The choice of sine oscillator algorithm can also depend on the platform and/or architecture chosen for implementation.

4.2 Algorithm Choice

To choose an algorithm for the estimation block, the different algorithms are compared based on their performance properties with regards to the range of the offset and noise level. Based on this, three algorithms are singled out, which are presented by Costas [1], Divsalar and Simon [2], and Yu, Shi, and Su[18]. The first algorithm is known as the Costas loop, which uses a multiplication of the I- and Q-channel to output the instantaneous phase of the input signal. The second algorithm removes the data dependency of the input signal from the signal and then estimates the instantaneous phase based on multiple samples. The third uses oversampling to determine the instantaneous frequency of the input signal by finding the difference between the samples within on bit period. Of these the second one is chosen because the first one has very poor performance for higher frequency offset (larger than 1 kHz) and the third has very poor performance in low SNR which is present when spread spectrum is used, as it is in 3G. The second, however, performs very good under low SNR conditions and for larger frequency offsets compared to the other algorithms. This is based on initial investigations of the BER and the performance for different frequency offsets done by Nicoloso [8, chap.6] and Divsalar and Simon [2].

The chosen algorithm is, as mentioned, presented in [2] and is a maximum likelihood phase estimator which can be modified to comprehend frequency offsets. The algorithm is a feed-forward algorithm and the output does not need a separate local oscillator as the output of the algorithm is the oscillating signal. In [2] three similar algorithms are found and the one chosen is scheme number two which in [8, chap.6] is found to perform good compared to the others presented. The algorithm is, however, only defined for BPSK, which means that it has to be modified to work in the 3G scenario using QPSK, which is the focus of the project.

Chapter 5

Rewriting the Algorithm for QPSK

As described in Chapter 4 the algorithm chosen for analysis and implementation is developed by Divsalar and Simon [2]. They describe three variations of the algorithm and the second one is chosen. The three algorithms is based on each other and therefore this chapter first modifies the first variation of the algorithm to QPSK, and the results from here are then used to modify the second variation. This algorithm is then simulated to determine its performance, in terms of BER, to verify that its good properties from BPSK are maintained for QPSK operation.

The algorithm can be seen in Figure 5.1 and the two main parts of the algorithm are shown in the dashed boxes. These parts are derived separately for QPSK based on the findings in [2].

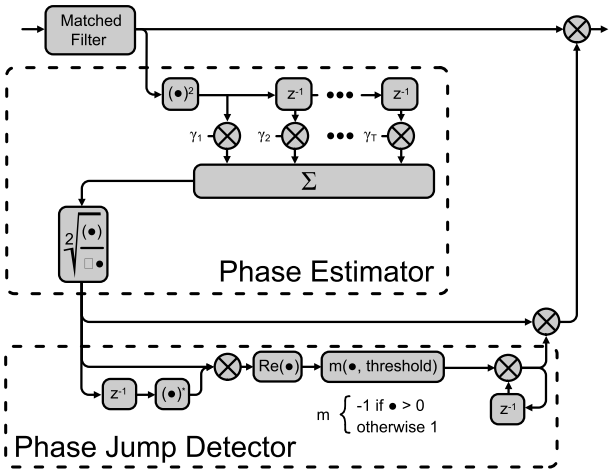


Figure 5.1: *Block diagram of the algorithm with the phase estimator and phase jump detector illustrated by the dashed boxes. γ represents the sinc function used in (5.24).*

5.1 Phase Estimator

The main functionality of the phase estimator is to estimate the phase error of the carrier signal based on samples obtained from the output of the matched filter. The estimator explained in [2] is for BPSK, while the derivation here is made for QPSK (for the 3G system), but is based on the same methodology.

The first step is to look at the estimator dealing with only static phase offset. The output of the matched filter can be represented as:

$$r_k = \sqrt{2P}e^{j(\phi_k+\theta)} + n_k \tag{5.1}$$

where n_k is a sample of white Gaussian noise with zero mean and variance $\sigma_n^2 = \frac{N_0}{T}$ in both directions of the complex plane ($\mathcal{N}(0, \frac{N_0}{T}) + j\mathcal{N}(0, \frac{N_0}{T})$), P is the transmit power and ϕ_k is the phase of the transmitted signal. Now, the received sequence $\bar{r}$ becomes $\bar{r} = r_{k-1}, r_{k-2}, \cdots, r_{k-N}$ which has the length N and it is assumed that the carrier phase θ is constant over this length (the case with frequency offset is considered in Section 5.1.1). With this, the likelihood function for receiving this sequence becomes:

$$p(\bar{r} \mid \phi, \theta) = \left(\frac{1}{2\pi\sigma_n^2}\right)^N e^{-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}\left|r_{k-i} - \sqrt{2P}e^{j(\phi_{k-i}+\theta)}\right|^2} \tag{5.2}$$

This is rewritten to the following:

$$p(\bar{r} \mid \phi, \theta) = Fe^{\alpha\sum_{i=1}^{N}\Re\left[r_{k-i}e^{-j(\phi_{k-i}+\theta)}\right]} \tag{5.3}$$

where:

$$F = \left(\frac{1}{2\pi\sigma_n^2}\right)^N e^{-\frac{NP}{\sigma_n^2}+\sum_{i=1}^{N}|r_{k-i}|^2} \quad \text{and} \quad \alpha = \frac{\sqrt{2P}}{\sigma_n^2}$$

The rewriting is found in Appendix A.

As F and α are independent of the data phase ϕ_k, they are unimportant for the Maximum Likelihood (ML) function. The ML estimator ($\hat{\theta}_{ML,k}$) of the phase error θ is then found as:

$$\ln[p(\bar{r} \mid \hat{\theta}_{ML,k})] = \max_{\theta}\left[\ln p(\bar{r} \mid \theta)\right] \tag{5.4}$$

As seen, the above equations define $p(\bar{r} \mid \phi, \theta)$. The one needed is $p(\bar{r} \mid \theta)$, but this value can be found be taking the expectation of $p(\bar{r} \mid \phi, \theta)$ with respect to ϕ, therefore:

$$p(\bar{r} \mid \theta) = E\left[p(\bar{r} \mid \phi, \theta)\right] \tag{5.5}$$

The possible values of ϕ_{k-1} in the QPSK case are $\left[0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right]$, the case for 3G is actually $\left[\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\right]$ (see constellation diagram in Figure 3.4), but this is makes no difference for the derivations. Therefore:

$$p(\bar{r} \mid \theta) = F \prod_{i=1}^{N} \left[\frac{1}{4} \sum_{\phi_{k-i}} e^{\alpha \Re\left[r_{k-i} e^{-j(\phi_{k-i}+\theta)}\right]} \right] \tag{5.6}$$

This is rewritten as:

$$\begin{aligned} p(\bar{r} \mid \theta) = F \prod_{i=1}^{N} \Bigg[\frac{1}{4} \Bigg(& e^{\alpha \Re\left[r_{k-i} e^{-j(0+\theta)}\right]} + e^{\alpha \Re\left[r_{k-i} e^{-j(\frac{\pi}{2}+\theta)}\right]} \\ & + e^{\alpha \Re\left[r_{k-i} e^{-j(\pi+\theta)}\right]} + e^{\alpha \Re\left[r_{k-i} e^{-j(\frac{3\pi}{2}+\theta)}\right]} \Bigg) \Bigg] \end{aligned} \tag{5.7}$$

Now using that $\Re\left[e^{-jx+\frac{\pi}{2}}\right] = \Im\left[e^{-jx}\right]$ and $\Re\left[e^{-jx+\pi}\right] = -\Re\left[e^{-jx}\right]$, this gives:

$$\begin{aligned} p(\bar{r} \mid \theta) = F \prod_{i=1}^{N} \Bigg[\frac{1}{4} \Big(& e^{\alpha \Re\left[r_{k-i} e^{-j\theta}\right]} + e^{\alpha \Im\left[r_{k-i} e^{-j\theta}\right]} \\ & + e^{-\alpha \Re\left[r_{k-i} e^{-j\theta}\right]} + e^{-\alpha \Im\left[r_{k-i} e^{-j\theta}\right]} \Big) \Bigg] \end{aligned} \tag{5.8}$$

As the cosine hyperbolic function is defined as $\cosh(x) = \frac{1}{2}\left(e^{x} + e^{-x}\right)$ this gives the following:

$$\begin{aligned} p(\bar{r} \mid \theta) &= F \prod_{i=1}^{N} \left[\frac{1}{4} \left(2\cosh\left(\alpha \Re\left[r_{k-i} e^{-j\theta} \right] \right) + 2\cosh\left(\alpha \Im\left[r_{k-i} e^{-j\theta} \right] \right) \right) \right] \\ &= F \frac{1}{2} \prod_{i=1}^{N} \left[\cosh\left(\alpha \Re\left[r_{k-i} e^{-j\theta} \right] \right) + \cosh\left(\alpha \Im\left[r_{k-i} e^{-j\theta} \right] \right) \right] \end{aligned} \tag{5.9}$$

And then $\ln[p(\bar{r} \mid \hat{\theta}_{ML,k})]$ is found, to ease the calculations, as:

$$\begin{aligned} \ln[p(\bar{r} \mid \hat{\theta}_{ML,k})] = \ln(F) + \sum_{i=1}^{N} \ln \Bigg(& \frac{1}{2} \Big(\cosh\left(\alpha \Re\left[r_{k-i} e^{-j\theta} \right] \right) \\ & + \cosh\left(\alpha \Im\left[r_{k-i} e^{-j\theta} \right] \right) \Big) \Bigg) \end{aligned} \tag{5.10}$$

To find this maximum value, the derivative is found and set to zero such that the value of θ which maximizes (see [7, p.318]) becomes $\hat{\theta}_{ML,k}$. This means that:

$$\frac{\partial \ln\left(p(\bar{r} \mid \theta)\right)}{\partial \theta} = 0 = \frac{\partial \ln(F) + \sum_{i=1}^{N} \ln\left(\frac{1}{2}\left(\cosh\left(\alpha \Re\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]\right) + \cosh\left(\alpha \Im\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]\right)\right)\right)}{\partial \theta} \tag{5.11}$$

Using the chain rule and that the real and imaginary operators can be moved outside derivatives this becomes:

$$\frac{\partial \ln\left(p(\bar{r} \mid \theta)\right)}{\partial \theta} = \sum_{i=1}^{N} \frac{2\left(-\sinh\left(\alpha \Re\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]\right)\right) \alpha \Re\left[j r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]}{\cosh\left(\alpha \Re\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]\right)} \cdots \cdots \frac{-\sinh\left(\alpha \Im\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]\right) \alpha \Im\left[j r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]}{+\cosh\left(\alpha \Im\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]\right)} \tag{5.12}$$

Now for a low SNR (the case for spread spectrum transmission), meaning that $\sigma_n^2 \to \infty$ and thereby $\alpha \to 0$ then $\sinh(\alpha x) \approx \alpha\left(\alpha x + \frac{\alpha^3 x^3}{6}\right)$ (this is the second order Taylor expansion around 0) and $\cosh(\alpha x) \approx 1$ this gives:

$$\begin{aligned}\frac{\partial \ln\left(p(\bar{r} \mid \theta)\right)}{\partial \theta} = \sum_{i=1}^{N} \Bigg(& \alpha \Re\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] \alpha \Re\left[j r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] \\ & - \alpha \frac{1}{6} \alpha^3 \Re\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]^3 \alpha \Re\left[j r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] \\ & - \alpha \Im\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] \alpha \Im\left[j r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] \\ & + \alpha \frac{1}{6} \alpha^3 \Im\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]^3 \alpha \Im\left[j r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] \Bigg)\end{aligned} \tag{5.13}$$

This is reduced to:

$$\begin{aligned}\frac{\partial \ln\left(p(\bar{r} \mid \theta)\right)}{\partial \theta} = \sum_{i=1}^{N} \alpha^4 \frac{1}{6} \Bigg(& -\Re\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]^3 \Im\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] + \\ & \cdots \Im\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right]^3 \Re\left[r_{k-i} e^{-j\hat{\theta}_{ML,k}}\right] \Bigg)\end{aligned} \tag{5.14}$$

Now, knowing that $\frac{\partial \ln(p(\bar{r}|\theta))}{\partial \theta} = 0$ in the maximum point, α^4 and the $\frac{1}{6}$ are neglected, the equation is also multiplied with minus one to ease calculations.

$$\sum_{i=1}^{N} \Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^3 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right] - \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^3 \Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right] = 0 \quad (5.15)$$

Solving this for $e^{j\hat{\theta}_{ML,k}}$ the estimator of the instantaneous phase becomes (The derivation are shown in Appendix B):

$$e^{j\hat{\theta}_{ML,k}} = \sqrt[4]{\frac{\sum_{i=1}^{N} r_{k-i}^4}{\left|\sum_{i=1}^{N} r_{k-i}^4\right|}} \quad (5.16)$$

When (5.16) is compared to the result of the first algorithm in [2] it is seen that these are very similar. The only difference is the exponents and the root, which is very intuitive taking the number of message points into account. This previous derivations therefore proves the intuition for the estimator.

5.1.1 Frequency Modification

To accommodate that the algorithm has to work with frequency offsets and not just phase errors, the algorithm has to be modified to handle this. This is also explained in [2] for BPSK and is derived here for QPSK.

First equation (5.9) is modified to include the frequency dependency of the data, which gives the following [2]:

$$p(\bar{r} \mid \theta, f) = F\frac{1}{2}\prod_{i=1}^{N}\left[\cosh\left(\alpha\Re\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right) + \cosh\left(\alpha\Im\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right] \quad (5.17)$$

Where T is the period time for one bit and f is the frequency error.

In order to find $p(\bar{r} \mid \theta)$ the expected value of the frequency is found by averaging over the frequency by integrating. Here the probability of the frequency is assumed to be uniformly distributed (see Figure 5.2) such that $p(\bar{r} \mid \theta)$ becomes:

$$p(\bar{r} \mid \theta) = \frac{F}{4f_{\max}}\int_{-f_{\max}}^{f_{\max}}\prod_{i=1}^{N}\left[\cosh\left(\alpha\Re\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right) + \cosh\left(\alpha\Im\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right]df \quad (5.18)$$

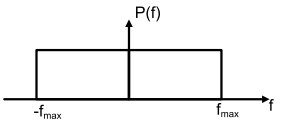


Figure 5.2: *The assumed probability distribution of the frequency offset.*

The value of $e^{j\theta_{ML,k}}$ which maximizes the above equation is found by differentiating the above equation. This is done by using the following equation:

$$\frac{\partial}{\partial x}\left[\prod_{i=1}^{k} f_i(x)\right] = \sum_{i=1}^{k}\left(\frac{\partial}{\partial x} f_i(x) \prod_{j\neq i} f_j(x)\right) \tag{5.19}$$

This results in the following derivative:

$$\begin{aligned}
\frac{\partial p(\bar{r} \mid \theta)}{\partial \theta} &= \frac{F}{2f_{\max}} \int_{-f_{\max}}^{f_{\max}} \frac{\partial \prod_{i=1}^{N} \frac{1}{2}\left[\cosh\left(\alpha\Re\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right.}{\partial\theta} \cdots \\
&\quad \cdots \frac{\left.+\cosh\left(\alpha\Im\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right] df}{\partial\theta} \\
&= \frac{F}{4f_{\max}} \int_{-f_{\max}}^{f_{\max}} \sum_{i=1}^{N}\left(\frac{\partial}{\partial\theta}\frac{1}{2}\cosh\left(\alpha\Re\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right. \\
&\quad +\frac{1}{2}\cosh\left(\alpha\Im\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right) \\
&\quad \cdot\prod_{j\neq i}\frac{1}{2}\cosh\left(\alpha\Re\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right) \\
&\quad \left.+\frac{1}{2}\cosh\left(\alpha\Im\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right) df
\end{aligned} \tag{5.20}$$

which is rewritten to:

$$\begin{aligned}
\frac{\partial p(\bar{r} \mid \theta)}{\partial \theta} &= \frac{F}{4f_{\max}} \int_{-f_{\max}}^{f_{\max}} \sum_{i=1}^{N}\left(\frac{1}{2}\sinh\left(\alpha\Re\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right. \\
&\quad \cdot\alpha\Re\left[jr_{k-i}e^{-j\theta}e^{j2\pi fiT}\right] \\
&\quad +\frac{1}{2}\sinh\left(\alpha\Im\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\alpha\Im j\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right] \\
&\quad \cdot\prod_{j\neq i}\frac{1}{2}\cosh\left(\alpha\Re\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right) \\
&\quad \left.+\frac{1}{2}\cosh\left(\alpha\Im\left[r_{k-i}e^{-j\theta}e^{j2\pi fiT}\right]\right)\right) df
\end{aligned} \tag{5.21}$$

Now using the low SNR assumption the sine and cosine hyperbolic functions are approximated with $\sinh(x) \approx x + \frac{1}{6}x^3$ and $\cosh(x) \approx 1$ this gives:

$$\begin{aligned}\frac{\partial p(\bar{r} \mid \theta)}{\partial \theta} &= \frac{F}{4f_{\max}} \int_{-f_{\max}}^{f_{\max}} \sum_{i=1}^{N} \left(-\frac{1}{2}\alpha^2 \Re\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \Im\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \right. \\ &\quad - \frac{1}{8}\frac{1}{6}\alpha^4 \Re\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right]^3 \Im\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \\ &\quad + \frac{1}{2}\alpha^2 \Re\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \Im\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \\ &\quad \left. + \frac{1}{8}\frac{1}{6}\alpha^4 \Im\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right]^3 \Re\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \prod_{j\neq i} 1 \right) df \\ &= \frac{F}{4f_{\max}} \frac{1}{48}\alpha^4 \int_{-f_{\max}}^{f_{\max}} \sum_{i=1}^{N} \left(-\Re\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right]^3 \Im\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \right. \\ &\quad \left. + \Im\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right]^3 \Re\left[r_{k-i} e^{-j\theta} e^{j2\pi fiT}\right] \right) df\end{aligned} \tag{5.22}$$

Setting the equation to equal zero, and using the derivation from Appendix B and neglecting F and $\frac{1}{192}$ results in the following, which is then integrated over f:

$$\begin{aligned}&\int_{-f_{\max}}^{f_{\max}} \sum_{i=1}^{N} \left(r_{k-i} e^{-j\hat{\theta}_{ML,k}} e^{j2\pi fiT} \right)^4 - \left| r_{k-i} e^{j2\pi fiT} \right|^4 df = 0 \\ &\left[\sum_{i=1}^{N} \left(r_{k-i} e^{-j\hat{\theta}_{ML,k}} \right)^4 e^{j8\pi fiT} - |r_{k-i}|^4 \left| e^{j8\pi fiT} \right| \right]_{-f_{\max}}^{f_{\max}} = \\ &\sum_{i=1}^{N} \left(r_{k-i} e^{-j\hat{\theta}_{ML,k}} \right)^4 \frac{e^{j8\pi f_{\max} iT}}{j8\pi f_{\max} iT} - |r_{k-i}|^4 \left| \frac{e^{j8\pi f_{\max} iT}}{j8\pi f_{\max} iT} \right| \\ &\quad - \left(r_{k-i} e^{-j\hat{\theta}_{ML,k}} \right)^4 \frac{e^{-j8\pi f_{\max} iT}}{j8\pi f_{\max} iT} - |r_{k-i}|^4 \left| \frac{e^{-j8\pi f_{\max} iT}}{j8\pi f_{\max} iT} \right| = 0 \\ &\sum_{i=1}^{N} 2 \left(r_{k-i} e^{-j\hat{\theta}_{ML,k}} \right)^4 \operatorname{sinc}(8 f_{\max} iT) - 2 \left| r_{k-i}^4 \right| \left| \operatorname{sinc}(8 f_{\max} iT) \right| = 0\end{aligned} \tag{5.23}$$

Solving this for $e^{j\theta_{ML,k}}$ the estimator becomes:

$$e^{j\hat{\theta}_{ML,k}} = \sqrt[4]{\frac{\sum_{i=1}^{N} r_{k-i}^4 \operatorname{sinc}(8 f_{\max} iT)}{\left| \sum_{i=1}^{N} r_{k-i}^4 \operatorname{sinc}(8 f_{\max} iT) \right|}} \tag{5.24}$$

Here $f_{\max}$ denotes the maximum possible frequency offset expected for the algorithm to handle. This value is as such the maximum value specified in Section 3.2.

5.2 Phase Jump Detector

Because the estimator defined in Eq. (5.16) is a result of the fourth root, this causes a 4 fold ambiguity in the estimator [7, p.317]. This means that the estimator only tracks the phase 90 degrees at a time. The implication of this is that the estimator only outputs a signal with a phase between $\frac{-\pi}{4}$ and $\frac{\pi}{4}$. This also means that when the output reaches $\pm\frac{\pi}{4}$ it jumps to $\mp\frac{\pi}{4}$. Because this is where the estimator should proceed to the next quadrant of the complex plane, these jumps are important for moving the estimator to the correct position.

To detect these jumps, a modified version of the phase jump detector presented in [2] is created. A diagram of it is shown in Figure 5.3. As seen, the present estimate is multiplied with the previous complex conjugated estimate. The imaginary part of this multiplication is then compared to a threshold value and if the value exceeds that a jump has taken place between the two estimates, which has to be handled.

When the jump is detected and the estimator has jumped from $\frac{-\pi}{4}$ to $\frac{\pi}{4}$, the estimator is multiplied with $-j$ and j when it has jumped from $\frac{\pi}{4}$ to $\frac{-\pi}{4}$. To continue the multiplication with j or $-j$ when a jump has not been present for the given sample, the last delay element and multiplier of the figure are introduced. If the output of the comparator is 1 when a jump is not present, the compensation is maintained for all samples.

Simulations of the algorithm is shown in Figure 5.5. Here it is shown that the jump detector can change due to noise even for a very high filter order. This can make the estimator end up in a wrong quadrant. This problem is not addressed in this project, but it is a possible point for improvement.

The above definition of the Phase Estimator and the Phase Jump Detector is shown in the block diagram in Figure 5.3. Comparing this to the block diagram in Figure 5.1 it is seen that there are small differences in the Phase Jump Detector and the Phase Estimator, but the overall structure of the algorithm is the same.

The following section describes the simulations of the algorithm which should clarify if the algorithm performs as good for QPSK as for BPSK and it should also determine the best suited number of inputs to be averaged over in the estimator.

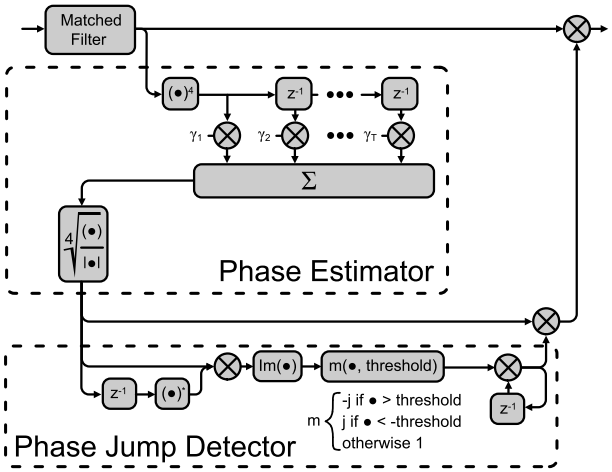


Figure 5.3: *Block diagram of the algorithm with the phase estimator and phase jump detector illustrated by the dashed boxes. γ represents the sinc function used in (5.24).*

5.3 Simulations

The functionality of the algorithm is simulated using MATLAB and Simulink. The model of the algorithm is built in Simulink, and MATLAB is used to initialize the constants of the simulation, run the Simulink model and collect and present the resulting data. The description of this model is found in Appendix C.

5.3.1 Results

The SNR/BER plots was done for a range of filter orders from 16 to 512 within the ordinary operating frequency offset of 0–2500 Hz. The simulation was also run for frequencies outside the normal operating interval with delay line lengths 128, 256 and 512. The results shown are only for filter lengths of 128, 256 and 512 as it is found in [8] that these are the ones which performs the best and are therefore the most interesting.

The BER plots are made from -4 to 10 dB SNR with a step size of 2 dB. Further the relative phase error for the 128 taps configuration is plotted in a 3D plot to see the phase error of the output of the system.

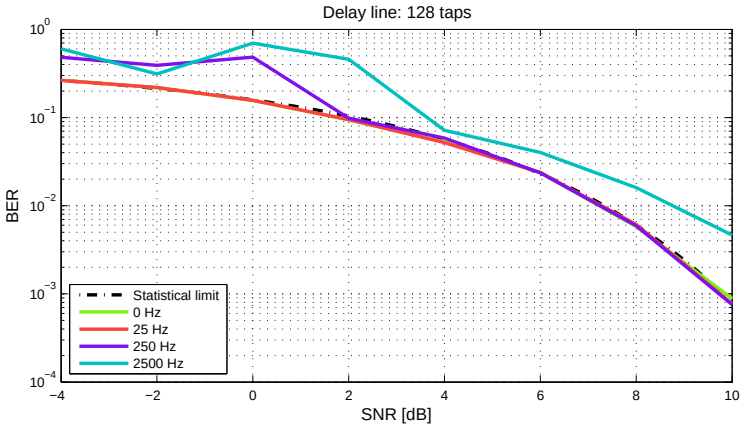


Figure 5.4: *SNR/BER plot of the algorithm employing a delay line length of 128 for offsets of 0, 25, 250 and 2500 Hz.*

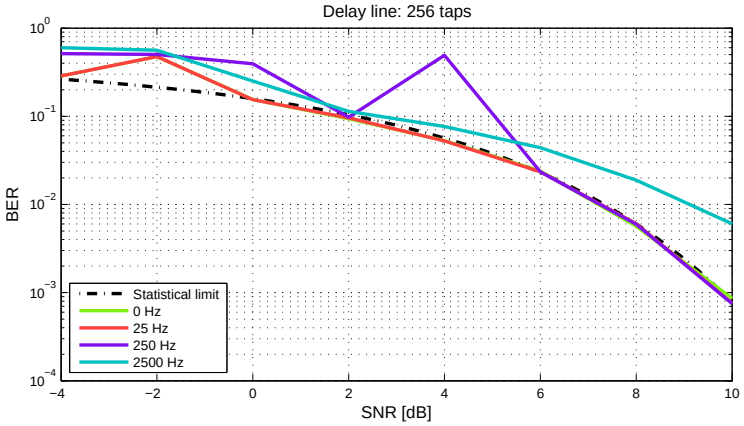


Figure 5.5: *SNR/BER plot of the algorithm employing a delay line length of 256 for offsets of 0, 25, 250 and 2500 Hz.*

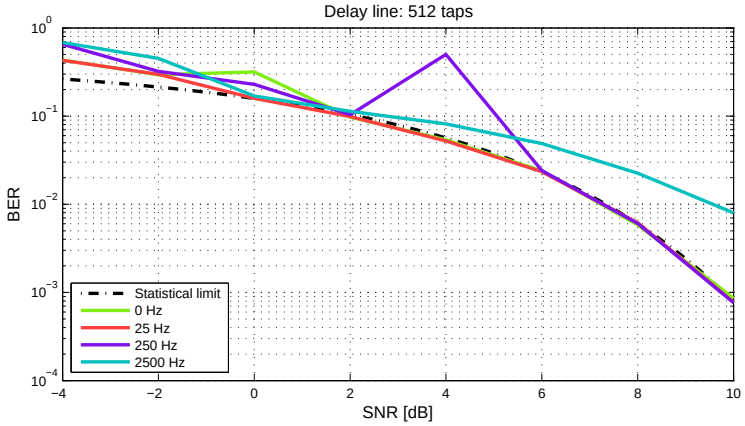


Figure 5.6: *SNR/BER plot of the algorithm employing a delay line length of 512 for offsets of 0, 25, 250 and 2500 Hz.*

The relative phase error of the system is shown in 5.7.

5.3.2 Discussion

The plots show a trade-off between the ability to "lock onto a signal" and the length of the delay line. The relative phase error plots indicate that the angle estimator works rather well for SNR above 0 dB as the estimates are located around either the correct angle or quarter circle intervals away. This is because of the phase jump detector not being able to "lock" onto a specific quadrant. But as the SNR decreases, the phase jump detector is less likely to be "fooled" into making a jump correction.

As the delay line length is increased, the noise is filtered away, and the phase jump detector begins to work properly. But a longer delay line means that the offset angle between the oldest and the youngest samples in the line getting more significant, thus affecting the performance of the filter, because it is "trying" to estimate one single angle, but is getting different angles as input.

The drawbacks of a short filter is that it produces quarter circle disambiguations thus making the system useless at low SNR. This can be seen in the relative phase error plot. The relative errors group around either the correct angle or at 90° clockwise, 90° counterclockwise or 180° relative to the correct angle. This happens when the noise admitted though the filter "fools" the phase jump detector. The advantages of the short filter is, however, that it gives a more precise angle estimate, as the real angles of the inputs are not too far apart; furthermore, it is tractable when considering implementation.

5.3.3 Conclusion

Based on the simulations it is decided that the filter order used in the rest of the project is 128. This choice is based on the lack of visible differences in the BER plots for 0, 25 and 250 Hz frequency offsets and the small increase in BER for the higher filter orders for 2500 Hz.

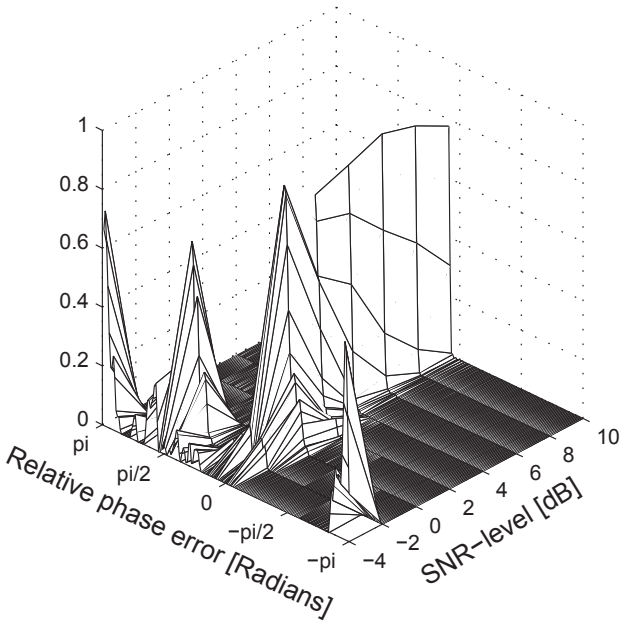


Figure 5.7: *3D plot of the relative phase error for 512 filter taps and 2500 Hz frequency offset. Notice the Phase Jump Detector fails to lock onto the correct quadrant for -4 and -2 dB SNR.*

Chapter 6

Implementation Analysis

This chapter serves two purposes. The main purpose is to aid the choice of hardware platform. Secondly, the choice of complex number representation is considered.

To aid the choice of the platform, some key figures for the algorithm are calculated. These are the estimated execution time of the algorithm and the multiplier count. To estimate the execution time, the critical path in the algorithm must be found. To do this, the maximum number of operations are found for fully sequential operation, and then by considering the inherent parallelism in the algorithm, the number of sequential operations in the critical path are found. The execution time of the operations are then estimated and combined with the number of operations used, to give a rough estimate of the total execution time of the system. This estimate is refereed to as the complexity of the algorithm.

Because the algorithm use complex numbers, the complexity analysis also investigates which is the best way to represent these complex numbers. The complex numbers can be represented in either Cartesian or polar format. The differences between these formats and their impact on the execution of the algorithm are investigated. The algorithm is divided into three main blocks; the filter, the estimator, and the phase jump detector. The number of operations needed for executing each block for both of the formats are then evaluated, making it possible to evaluate the difference between the complex number formats.

Some of the operations which are defined for the individual blocks are more complex than simple additions or multiplications. These operations are defined in terms of simpler operations. This results in the algorithm being defined in terms of simple operations and using the least complex type of the calculations with respect to this particular algorithm.

6.1 Complexity

The section finds the complexity of the algorithm in terms of the number of operations used. To find the complexity of the algorithm it is first divided into blocks. The individual blocks are then analyzed for their complexity individually before these results are added to get the complexity of the entire algorithm.

The main blocks of the algorithm are the filter, the estimator, and the phase jump detector, shown in Figure 6.1. This division was chosen, as the interfaces between the blocks could

be easily defined. Also, as will be described later, the interfaces between the blocks are suitable points in which to introduce complex format conversions.

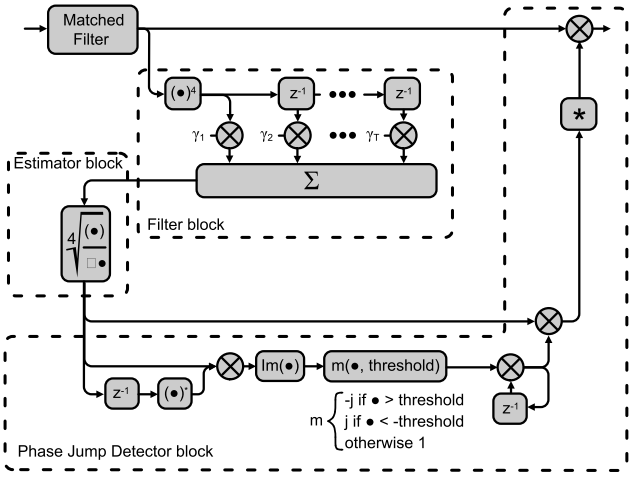


Figure 6.1: *Block diagram of the algorithm using the cartesian complex format. In this figure, the three functional blocks of the algorithm are framed.*

6.1.1 Complex Number Format

There are two ways of representing complex numbers, in Cartesian form with a real and an imaginary part, and polar form with magnitude and angle. Both of these formats can be used for the algorithm but the best suited has to be chosen.

The input to the algorithm is in Cartesian form, because the I and Q channels are sampled individually. Thus, a conversion has to be introduced in order to use the polar format. A representation of the algorithm in the Cartesian format results in the block diagram shown in Figure 6.1. The points where the conversion blocks can be inserted are either in front of or after the blocks. The placement of a conversion in front of a block depends on the type of operations that are performed in the block, and whether it is an advantage to do them in polar format.

Performing the conversion before the filter would result in the filter being a matter of summing complex numbers in polar format, which would not be efficient because this operation would include a conversion back and forth between the polar and the Cartesian format.

After the Filter block a conversion would, however, be very suitable. The operations taking place after the filter block only concerns the phase of the complex number. As seen, the output of the filter is normalized in the estimator block which then means that the length of the complex number is irrelevant for the further computations. Effectively, the fourth root divides the phase of the number by four, which is very easily computed in the fixed point polar format, because this would mean just two shift operations on the phase of the complex number.

The phase jump detector in the polar would also be simpler than in the Cartesian format.

Only the phase of the current and previous estimates have to be compared and if they are different by more than 45°, then 90° are added to or subtracted from the current phase estimate.

Taking these observations into consideration a conversion to polar format is placed in front of the estimator block and a reconversion block is placed after the phase jump detector. This gives the modified algorithm shown in the block diagram shown in Figure 6.2.

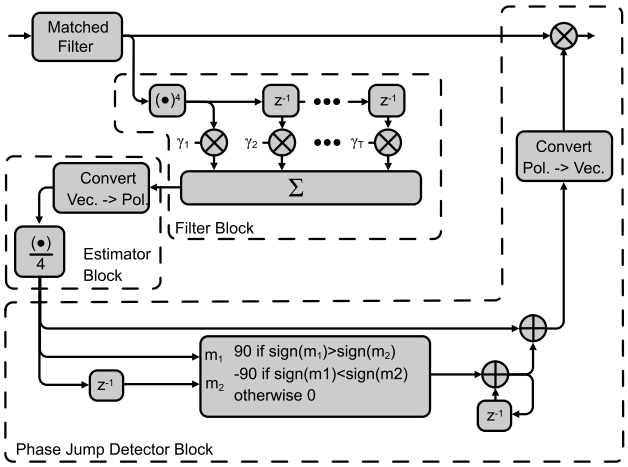


Figure 6.2: *Block diagram of the algorithm using the polar complex format.*

Based on these setups the complexity of the algorithm is found for both formats, which is then used to choose the best suited format for implementation.

6.1.2 Number of Operations

As described previously the complexity of the algorithm is based on the number of operations that needs to be executed for the algorithm. Because of the different complex number formats, the number of operations are found for each block for both formats and for both sequential and parallel operation. The number of operations in parallel operation is found by utilizing all inherent parallelism in the algorithm.

As the algorithm is divided into the blocks shown in Figure 6.1 and Figure 6.2, the complexity of the algorithm is found by adding the complexities of the individual blocks for both the complex number formats. The count of the simple operations are found by use of data flow graphs derived from the two algorithm diagrams in Figure 6.1 and Figure 6.2.

Filter Block

The number of operations needed for the filter block is found by means of the data flow graph shown in Figure 6.3. The two top left multiplications are the power of four operation. The number of operations for utilizing the parallelism is directly the number of operations in the critical path in the data flow graph.

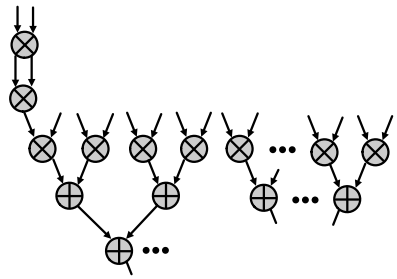


Figure 6.3: *Partial data flow graph for the filter block. The ellipses indicate that the amount of multiplications and additions varies depending on the desired filter length. Note that all operations in the data flow graph are complex. The critical path of additions in case of parallel processing is found to be* $\log_2(n)$ *where n is the filter order. The filter order throughout has been set to* $n = 128$ *thus the critical path is 7.*

The number of operations are found by examination of the data flow graph. The filter length is as mentioned in Section 5.3.3 set to 128 taps, which is therefore used in this analysis. Note that the operations are complex, meaning that each operation is an operation with complex numbers as input and output. As the conversion to polar format happens after the filter, the number of operations are the same for both complex formats.

The number and types of operations needed for this block both with and without utilizing parallelism are shown in Table 6.1.

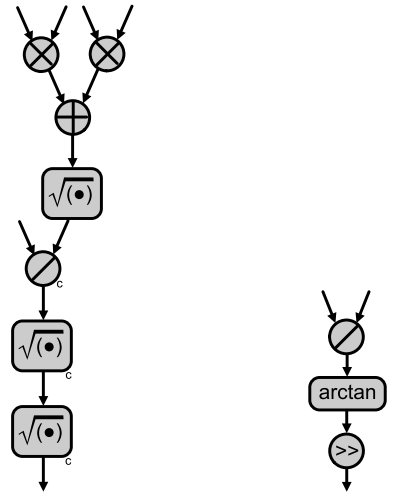
	Sequential	**Parallel**
Multiplications	130	3
Additions	127	7

Table 6.1: *The critical paths for the filter block in sequential and parallel operations.*

Estimator Block Complexity

The number of operations for the estimator block are based on the data flow graphs shown in Figure 6.4.

The number of operations are shown in Table 6.2. As seen by the figures in Table 6.2; there is a big difference between the two number formats, but only a small reduction in the number of operations in the critical path when utilizing the inherent parallelism. It is also seen that if the division has a longer execution time than the trigonometric function this block is preferred to be executed in the polar format.



(a) *Cartesian format.*

(b) *Polar format.*

Figure 6.4: *Data flow graphs for the estimator block with the different complex number formats.*

Sequential Operation

	Cartesian	**Polar**
Division	2	1
Trigonometric	0	1
Shift	0	1
Multiplication	2	0
Addition	1	0
Square Root	1	0
C Square Root	2	0

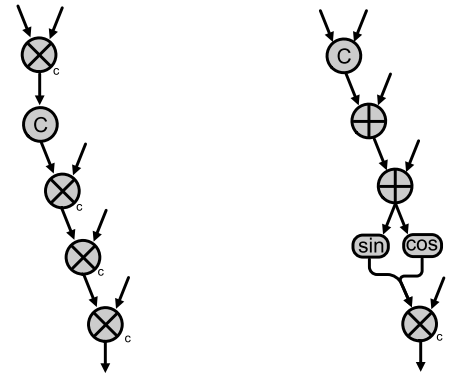
Parallel Operation

	Cartesian	**Polar**
Division	1	1
Trigonometric	0	1
Shift	0	1
Multiplication	1	0
Addition	1	0
Square Root	1	0
C Square Root	2	0

Table 6.2: *The critical paths for the filter block in sequential and parallel operations.*

Phase Jump Detector Block Complexity

The data flow graphs for the phase jump detector blocks are shown in Figure 6.5 for the Cartesian and polar format. Based on these, the number of operations are shown in Table 6.4.



(a) *Cartesian format.*

(b) *Polar format.*

Figure 6.5: *Data flow graphs for the phase jump detector block with the different complex number formats.*

The different complex formats makes that the number of operations for the block are different. The number of operations for the block in cartesian and polar format is shown in Table 6.3. Again only a small reduction is seen between sequential and parallel operation, which is because of the small amount of inherent parallelism for this block in both formats.

Sequential Operation

	Cartesian	**Polar**
C Multiplication	4	1
Compare	1	1
Addition	0	2
Trigonometric	0	2

Parallel Operation

	Cartesian	**Polar**
C Multiplication	4	1
Compare	1	1
Addition	0	2
Trigonometric	0	1

Table 6.3: *Number of complex operations needed for the filter block.*

Trigonometric Operation Because the Trigonometric operations can be implemented in many different ways the best suited for the algorithm needs to be found. With the method specified the operation execution time can be estimated, and then the algorithm execution time can be estimated.

The two conversion blocks shown in Figure 6.2 do not have the same functionality, as one

converts from Cartesian to polar format and the other does the opposite. The first has to find the angle of a complex number. The other and opposite convert operation finds the real and imaginary values from the angle of a complex number. Note that some operations can be left out as this implementation has no need for the magnitude when it has been converted to polar format.

The phase v of a complex number A can be found as $v = \arctan\left(\frac{\Im(A)}{\Re(A)}\right)$.

The real and imaginary values of the number A can be found from the phase v as: $\Re(A) = \cos(v)$ and $\Im(A) = \sin(v)$.

This means that the convert operations combined uses 3 real trigonometric functions and 1 real division.

The trigonometric is chosen to be implemented as a look up table with a linear approximation between the points in the table. This is done because of the results of [14] where the CORDIC, look up table and piece wise linear approximation methods are compared. They found that the piecewise linear approximation is less power- and area-consuming and by nature it is also less memory consuming, than the look up table. This means that the trigonometric functions are substituted with one compare, one multiplication and one addition.

By means of this the number of operations of the algorithm is found for sequential and parallel operations. The resulting number of operations are shown in Table 6.4 and 6.5 respectively.

Operation	**Filter**	**Estimator Cartesian**	**Estimator Polar**	**Phase Jump Detector Cartesian**	**Phase Jump Detector Polar**	**Cartesian Total**	**Polar Total**
Multiplication		2	1		2	2	3
- complex	130			4	1	134	131
Addition		1	1	(1)	4	1+(1)	5
- complex	127					127	127
Division		2	1			2	1
Square root		1				1	
- complex		2				2	
Comparison			1	1	3	1	4
Shift			2				1

Table 6.4: *An overview of the operations count for the different blocks for sequential operation, in the end summarized for Cartesian and polar solutions. The (1) in the Cartesian phase jump detector signifies a complex conjugation.*

Operation	Filter	Estimator Cartesian	Estimator Polar	Phase Jump Detector Cartesian	Phase Jump Detector Polar	Cartesian Total	Polar Total
Multiplication		1	1		1	1	2
- complex	3			4	1	7	4
Addition		1	1	(1)	3	1+(1)	4
- complex	7					7	7
Division		1	1			2	1
Square root		1				1	
- complex		2				2	
Comparison			1	1	2	1	3
Shift			1				1

Table 6.5: *An overview of the operations count for the different blocks for parallel operation, in the end summarized for Cartesian and polar solutions. The 1 in parenthesis in the Cartesian phase jump detector signifies a complex conjugation.*

It is easy to see from Table 6.4 and 6.5 that using parallel processing the algorithm has a much shorter critical path. This is especially due to the filter block which has a high degree of inherent parallelism.

6.2 Pipelining

To further decrease the execution time of the algorithm, pipelining is utilized [9, p.66-67]. It decreases the time between outputs of the algorithm, but increases the latency and the number of operations executed in parallel.

To utilize pipelining, the algorithm is split into nodes each maintaining a specific functionality. The connections between nodes (arcs) denotes the data dependencies between the nodes. Pipelining is utilized in order to reduce the critical path of the algorithm in exchange for a longer latency and control of the data. The control of the pipeline is not considered here, but in the implementation of the algorithm in Section 7.3. The neglection of the control of the pipeline gives a small overhead in the execution time which is not taken into account, but this is chosen for simplicity. Two new blocks are introduced here, which are the "Power Four" and "Complex Mult", which until this point have been considered part of the "Filter" and "Phase Jump Detector" blocks respectively. This is done to further decrease the critical path of the algorithm compared to a pipeline with the three stages equal to the blocks defined in the previous section.

The signal-flow graph can be seen in Figure 6.6. This leads to a coarse grain pipelining, but it still significantly reduces the critical path of the algorithm.

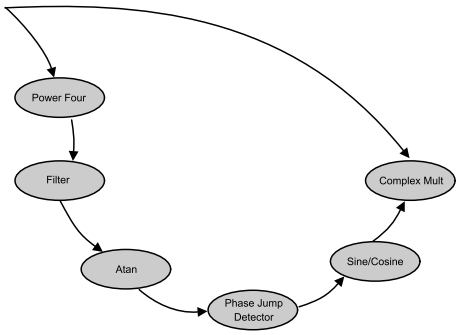


Figure 6.6: *Signal flow graph of the algorithm containing the two extra blocks that are introduced for pipelining purpose. This SFG is used for both complex formats.*

To implement pipelining feed-forward cutsets are introduced in the signal flow graph as shown in [9, p.66-67], and delays are introduced where the arcs and the cutsets intersect. This means that the signal flow graph is divided into several independent signal flow graphs. The signal-flow graph with the cutsets is shown in Figure 6.7. With the graph the pipeline is shown in Table 6.6. The input to the algorithm is denoted $r(k)$.

It is seen that the critical path now has been reduced from $t_{\text{crit}} = t_{\text{Power four}} + t_{\text{Filter}} + t_{\text{Estimator}} + t_{\text{PJD}} + t_{\text{Complex mult}}$ to $t_{\text{crit}} = \max\left(t_{\text{Power four}}, t_{\text{Filter}}, t_{\text{Estimator}}, t_{\text{PJD}}, t_{\text{Complex mult}}\right)$ which is a significant reduction. However, the number of operations remains the same in the same

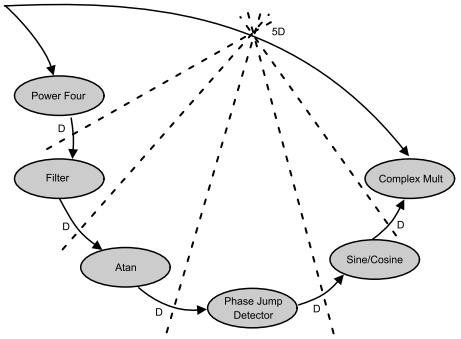


Figure 6.7: *Block diagram of the algorithm using the cartesian complex format.*

Clock 1	Clock 2	Clock 3	Clock 4	Clock 5
Power four($r(k)$)	Power four($r(k+1)$)	$\cdots$	$\cdots$	$\cdots$
	Filter($r(k)$)	$\cdots$	$\cdots$	$\cdots$
		Estimator($r(k)$)	$\cdots$	$\cdots$
			PJD($r(k)$)	$\cdots$
				Complex Mult($r(k)$)

Table 6.6: *The table shows when the different inputs to the algorithm are processed in each node.*

amount of time, but they can be executed in parallel as they do not have any data dependencies.

In Table 6.7 the number of operations in each of the 5 blocks are shown

With these numbers the estimated execution time of the algorithm is found.

Operation	Power Four	Filter	Estimator Cartesian	Estimator Polar	Phase Jump Detector Cartesian	Phase Jump Detector Polar	Complex Multiplication
Sequential							
Multiplication			2	1		2	
- complex	2	128			3	0	1
Addition			1	1	(1)	4	
- complex		127					
Division			2	1			
Square root			1				
- complex			2				
Comparison				1	1	3	
Shift				2			
Parallel							
Multiplication			1	1		1	
- complex	2	1			3	0	1
Addition			1	1	(1)	3	
- complex		7					
Division			1	1			
Square root			1				
- complex			2				
Comparison				1	1	2	
Shift				1			

Table 6.7: *An overview of the operations count for the different blocks in the pipeline.*

6.3 Execution Time

To estimate the time for execution of the algorithm, the operations in Table 6.7 are analyzed for their individual execution time. To find the execution time, experiments are conducted. This is due to the fact that it has not been possible to find reasonable estimates for the execution time for the different operators. As mentioned in Chapter 1, the platform is chosen as an Altera FPGA, which means that the execution of the different operators also depends on their individual implementation. To avoid this, the functions are implemented with the IP cores available in the development tool (Quartus II v. 9). This means that the time needed to implement the algorithms for e.g. the square root and divide operations are reduced to a minimum. The execution times that have been found through experiments are shown in Table 6.8 with their respective execution time. The FPGA's considered are the Cyclone 2, Cyclone 3 and Stratix 2.

	Execution time [ns]		
Operation	**Cyclone 2**	**Cyclone 3**	**Stratix 2**
Addition	11.2	12.5	11.74
- complex	11.2	12.5	11.74
Multiplication	15.5	15.3	15.3
- complex	18.2	15.22	17.56
Square root	15.5	13.4	14.13
- complex	137.8	97.8	93.44
Division	48	46.0	41.7
Compare	10	10	13.19

Table 6.8: *The operations and their execution times in [ns] derived from experiments on the different FPGAs.*

Using these results the execution time of the algorithm is estimated for both sequential and parallel operation. This is done by summing the number of operations multiplied with the execution times for the operations.

In Table 6.9 it is shown that both parallelism and pipelining should be utilized to be able to deal with the real time requirement to the algorithm. As described in Section 3.3 the sample frequency is 3.84 MSamples/s which gives approximately 260 ns for the algorithm. As the estimated execution time shown in Table 6.9 is without any kind of control the fastest of the complex formats is chosen, which is the polar format, when both parallelism and pipelining are utilized.

Another aspect of the platform choice is the number of embedded multipliers on the FPGA's. The parallel configuration considered in the previous sections has up to 270 multipliers in parallel but the analyzed FPGA's has only up to 132 (the Cyclone 3). Because of this and the similar results presented in Table 6.9, the FPGA with the highest number of embedded multipliers is chosen as platform for the implementation (the Cyclone 3). How the multipliers are utilized is described in Chapter 7.

Device	Power Four	Filter	Estimator Cartesian	Estimator Polar	Phase Jump Detector Cartesian	Phase Jump Detector Polar	Complex Multiplication
Sequential							
Cyclone 2	36.4	3752	429.3	84.7	64.6	105.8	18.2
Cyclone 3	30.44	3535.7	344.1	83.8	55.66	110.6	15.22
Stratix 2	35.12	3738.7	326.75	81.93	65.87	117.13	17.56
Parallel							
Cyclone 2	36.4	96.6	365.8	84.7	64.6	69.1	18.2
Cyclone 3	36.4	96.6	282.8	83.8	55.66	72.8	15.22
Stratix 2	35.12	95.96	269.75	81.93	65.87	76.9	17.56

Table 6.9: *Estimated execution time in [ns] for all the blocks defined in Section 6.2 on the different FPGA's.*

6.4 Determining Fixed-Point Format

The issue when moving to fixed point is: How many bits of word length are needed for the algorithm to work properly? Fewer bits reduces the amount of possible values of the numbers thus reducing the accuracy of the output, whereas more bits demands more resources from the system. In determining the fixed-point format for representing the values in the system, two choices must be taken. The word-length and the fraction length. The word-length determines the amount of possible values that a variable can represent, and the fractional length how accurately a number can be represented, see Figure 6.8 and Example 6.1. The choices depend on the possible values of the variable to be represented can take on.

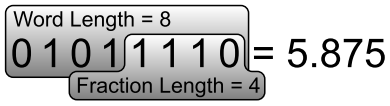


Figure 6.8: *An example of the value 5.875 being represented by an 8-bit word with a fraction length of 4. The stored integer "01011110" in decimal is 94. The fraction length of four effectively divides the stored integer by* 2^4 *giving the value 5.875. The accuracy is* $2^{-4} = 0.0625$ *meaning that any value in [5.8438; 5.9063[is rounded to 5.875.*

Example 6.1

Given a fixed-point number with a word length of 8 bits and a fraction length of 4 bits. The amount of different values it can represent is $2^8 = 256$ and the smallest number it can represent (next to 0) is $2^{-4} = 0.0625$ thus the largest representable number is 15.9375.

6.4.1 Dynamic Range

Both the word length and the fractional length determine the maximum (and in case of signed values the minimum) representable value of the number. Choosing the word length is fairly easy for band-limited signals, as the number must be able to represent both the highest and the lowest number of the signal. If the signal on the other hand has infinite bandwidth, like white Gaussian noise, the signal will not be fully representable, and some limit must be decided. The error function of the signal's probability density function might be used to find the limits which includes statistically e.g. 95 % of the expected values.

Furthermore it must be decided what will happen with the values that are unrepresentable by the number, this is called overflow. One possibility is for the number to saturate and just represent the maximum. Another approach is to make the number wrap, i.e. to just start from the other end of the scale, see example 6.2. The saturation is preferred for inputs like the Cartesian samples for the phase estimator, because it does not make sense to make them wrap and represent some extremely large positive number with a large negative value. The wrapping method, however, works great for representing angles. where incrementing π radians will make it wrap to $-\pi$, which is the intended operation.

Example 6.2

A fixed-point number with a word length of 8 bits and a fractional length of 0 will be able to represent integer numbers 0–255. If the number saturates when overflow occurs, the numbers: 255, 256 and 257 will be represented as [255 255 255]. If, on the other hand the number wraps the overflow the resulting numbers will be [255 0 1].

6.4.2 Precision

The fractional length will determine the resolution of the number. But deciding on the this length might not be as straightforward as deciding on the word length. The longer the fractional length the finner grained the number representation, but then more computing resources are needed. So on one hand the designer has the desire to represent numbers as close to their original values as possible, and in the other the need to reduce the word length (and thereby the fractional length) to a degree that is realizable, using the resources available.

Every value not representable by the fixed-point number will be rounded to a representable value in the vicinity of the original value. Whether the value is rounded up or down is a matter of the rounding method employed, see table 6.10.

All the rounding methods will add noise to the signal. The mean of this noise will differ according to the rounding method, but the probability density function (pdf) of the noise will be uniformly distributed with a width equal to the quantization step. Widrow et al. show in [17] that quantization of the value is in many respects the same as discretization of

Rounding Method	Behavior
ceil	Round toward positive infinity.
convergent	Round toward nearest. Ties round to the nearest even stored integer.
fix	Round toward zero.
floor	Round toward negative infinity.
nearest	Round toward nearest. Ties round toward positive infinity.
round	Round toward nearest. Ties round toward negative infinity for negative numbers, and toward positive infinity for positive numbers.

Table 6.10: *The different rounding methods included in the Fixed Point Toolbox in MATLAB*

the time, and essentially the same approach can be taken in analyzing the impact it has on the system.

To obtain the pdf of the signal after quantization the pdf of the original input must be convolved with the pdf of the quantization noise. By Fourier transforming the pdfs, they can be multiplied instead. The Fourier transform of the pdf is called the characteristic function (CF). This allows for constructing a model of the quantization processes of the entire system. The paper "Modeling Quantization Noise in Finite Impulse Response Filters" in Appendix F discusses the work done in this project on how this approach can be used to model the filter part of the phase estimator. In essence it shows that when enough uniformly distributed pdfs are convolved, they approximate a Gaussian distribution with the variance of the sum of the variances of the uniform pdfs. Thus the resulting simplified noise model of the filter is an additive white Gaussian noise (AWGN) source with a variance equal to the sum of the variances of all the filter taps. This noise source can then be plugged into a floating point simulation, in stead of running an actual fixed point simulation. Alternatively, the variance of the quantization noise can be added to the variance of the noise source in a calculation of the BER for an ideal QPSK system. This was done for a 128 tap filter with fraction lengths between 3–10 and inputs between -1–1. The results can be seen in Figure 6.9.

The plot shows that any word length above 6 bits exhibits about the same performance as the theoretical limit. This suggests using a word length of 7 bits.This analysis only deals with the filter part of the system, but the rest of the system will certainly also add some quantization noise. However, the filter adds noise in each tap. Thus the quantization noise of the many taps dwarfs the noise from the few following blocks. A design choice of 9 bits word length was done in order to be on the "safe" side of the 7–8 boundary and including one bits for sign, in all 9 bits. The fraction length is set to 8 so that the resulting dynamic range becomes [-1; 0.9961], *almost* encompassing the interval -1–1 saturating only very little. This is done in stead of *almost* wasting a bit by setting the fraction length to 7 and encompassing [-2; 1.922].

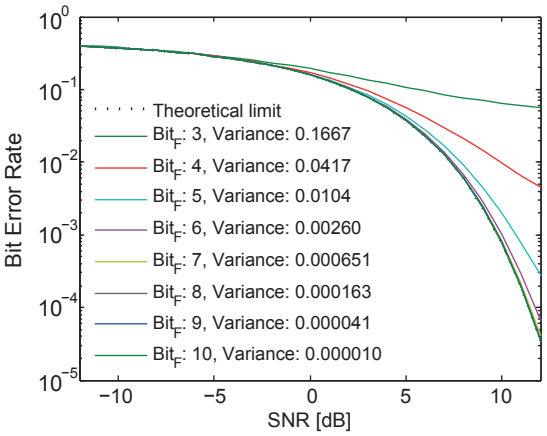


Figure 6.9: *The BER vs. SNR plot when augmenting the calculation with the variance of word lengths between 3–10. When the word length rises beyond 6, the performance is very close to the theoretical limit.*

Chapter 7

Implementation

This chapter describes the implementation of the algorithm on to the chosen Cyclone III FPGA. An overall overview and block diagram of the implementation is presented along with the controlling state machine of the system. Following this the individual blocks are described and tested and finally integrated

First a block diagram for the implementation is developed. It is based on the blocks which are also used in Section 6.2 which is done to have easier control of the scheduling of the blocks. The diagram is shown in Figure 7.1.

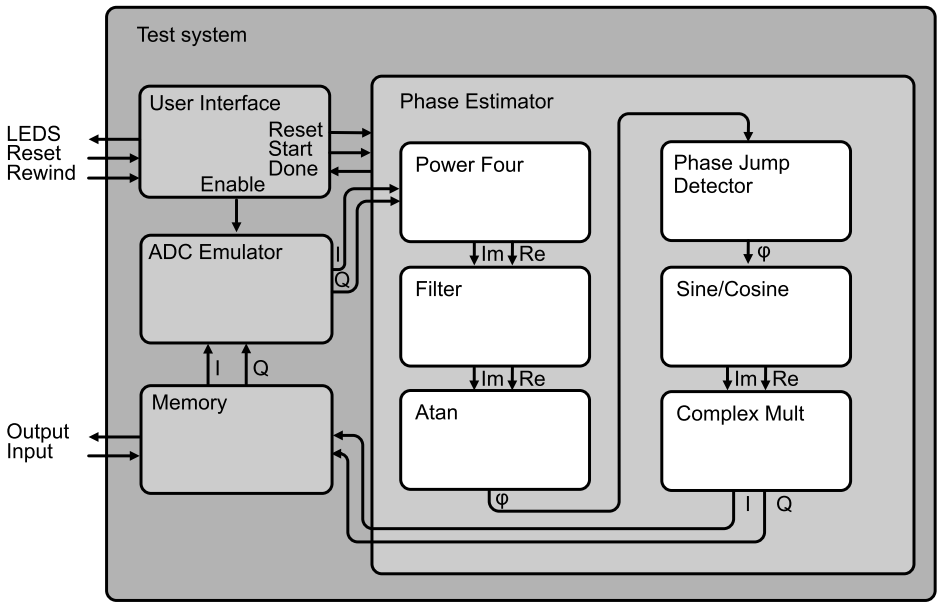


Figure 7.1: *Block diagram of the VHDL implementation of the algorithm. The control signals of the low hierachy blocks is omitted.*

As seen by the diagram the test system encapsulates the algorithm. This is done to emulate

the interfaces the algorithm would have if it was implemented in a 3G base station. When a block surround another block this means that the surrounding block controls the surrounded block, and therefore also the hierarchy is illustrated by the figure.

The different blocks and their functionality is shortly described below.

- **Test System**
  Emulates the interfaces to the algorithm and maintains the test facilities for the system, mainly in- and outputs.
- **User Interface**
  Controls the entire system and indicates to the user the state of the system and allows interaction to e.g. start the algorithm.
- **ADC Emulator**
  Emulates the behavior of an ADC.
- **Memory**
  Stores the in- and outputs of the algorithm.
- **Phase Estimator**
  Includes the overall structure of the algorithm. The block therefore also takes care of pipelining the algorithm.
- **Power Four**
  Outputs the 4^{th} complex power of the complex input.
- **Filter**
  Filters the input with the sinc function derived in Section 5.1.1.
- **Atan**
  Outputs the atan (angle) of the division of the imaginary input part with the real part.
- **Phase Jump detector**
  Handles the ambiguity of the estimator and thereby ensure the estimator is defined from 0 to 360 degrees.
- **Sine/Cosine**
  Outputs the sine and cosine of the input angle, thus the output is a complex numbers real and imaginary part with angle equal to the input of the block.
- **Complex Mult**
  Multiplies the estimator with the input to the algorithm and thereby compensates the input for the frequency offset.

In order to control the blocks they have common interfaces. These can be seen in Figure 7.2. Other than the in- and outputs, the system clock and reset there are done and strobe signals. These are used to control when the block should run and when it is done. The blocks internal behavior is determined by a finite state machine.

The in- and output signals are as the name indicates the in- and outputs of the block. Other than these signal the strobe signal is used to start a block while the input to the block is set and the done signal then indicates when a block has set the output and is therefore also

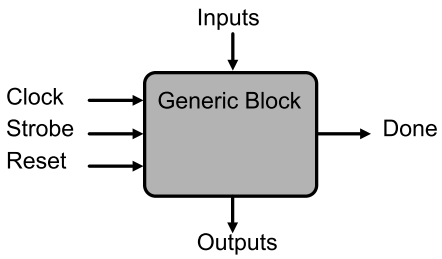


Figure 7.2: *Diagram of a generic block with specific common interfaces which is used throughout the implementation.*

ready to begin a new calculation. The reset signal is used to set all registers which are used by the block to its initial value. The clock signal is used by the block to clock all actions within the block. With these common interfaces it is easy to design, interconnect and control the blocks.

Before the implementation of the algorithm is described the tests which are conducted during the design of the blocks are designed.

7.1 Block Tests

To design the blocks, tests have to be conducted in order to verify, that the blocks fulfill their functionality. These tests eases the design of the blocks such that they do not have to be implemented in the entire implementation before they can be tested. This Section describes the design of these tests and how they are conducted.

To test the blocks ModelSim is enployed. This program enables the user to simulated VHDL code and does so in a quick and easy way. Then to test VHDL code in the program test benches is written, which is also done in VHDL code, but these files has other properties than the design VHDL files as they are created only for simulation purposes. This means that these files can use timing commands which can not be used in design files as they are not synthesisable to logic gates. Therefore, in these test benches all the in- and outputs to a specific design, which is simulated/tested, are created. The control signals are created to fit the implementation of the block. Therefore the test does not test every possible input control input values but only the values which are expected for normal use. The input data however is made using random inputs. Because every input cannot be tested e.g. with complex input, the inputs to the blocks are made as random values. These values are created using MATLAB and written to a file which can be loaded using the test benches in ModelSim. As the inputs are made using MATLAB where the fixed point simulations are also made the expected output from the block under test is also calculated here. The output from the simulation in ModelSim can then be compared to the fixed point simulation to determine if the block performes as expected.

The tests are run throughout the block design process where errors and bugs are corrected. This ensures that the bugs and errors are quickly found and corrected such that errors does not propergate to other parts of the block, making it harder to identify the error.

The following describes the blocks individually along with their state machines, when their

behavior is described they are tested according to the tests described in Section 7.1. Only the Complex Mult block is not described as it is implemented as an IP-core and it is tested with the Phase Estimator block.

7.2 Test System

The test setup emulates the analog-to-digital converter, automatic gain control, and everything else sitting before the frequency estimator algorithm in the signal chain in an actual system. It also collects the output of the system. This way, the system is stimulated with known inputs and thus its behavior can be tested and compared to the MATLAB implementation. An outline of the work flow of the test system is depicted in Figure 7.3. An explanation of the MATLAB implementation is given in Appendix E.

7.2.1 The MATLAB part

MATLAB is used to create both the stimulus for the system and to process the output of the system calculating the BER. The stimulus creating part is divided into two stages, first the stage creating the actual simulated input signal and secondly the stage converting the MATLAB vector to an Intel HEX file for loading into the internal memory of the FPGA. The first stage is shown in Figure 7.4

The input signals are constructed in MATLAB using components from the communications toolbox. The input system includes a quasi-random data source, a QPSK-modulator, a root-raised transmit filter, an additive white Gaussian noise (AWGN) channel with frequency offset, a receive filter matching the transmit filter, and a automatic gain control (AGC). These quasi-randomness of the values are controlled by two seeds, one for the data and one for the AWGN in the channel, this ensures that a test can be repeated with the same random data and noise as in a previous test. The transmit-/receive filtering is done to match the performance of existing 3G systems, as the standard prescribes the use of these filters.

Because the system is operating in fixed point, the values must be bounded within the dynamical range of the system. This is no problem when no noise is present as the QPSK-modulator will represent the symbols with either -1 or 1. But when noise is added the input starts to saturate when the values exceed -1 or 1 on either axis. To counter this, an AGC is simulated. It does not work in the traditional way, by continually measuring the input power and then scaling it to a certain level. Instead the entire vector of generated samples are normalized by the maximum norm sample before before being filtered by the receive filter.

Downloading the Samples to the FPGA The test system uses the internal memory modules of the FPGA to store the input and output signals. The IP-core provided by Altera lets the In-System Memory Content Editor tool in Quartus®II edit this memory during runtime via JTAG. This way, the input vectors can be input to the system in an easy fashion without reconfiguring the FPGA or even resetting the delay elements.

The In-System Memory Content Editor takes an Intel hex-file as an input for loading the test signals into the memory and also outputs hex-files. Because of this, the MATLAB array of test input must be saved as Intel hex-file and the output converted back to MATLAB arrays. Two MATLAB functions for this has been written for the project: `Array2IntelHex` and `IntelHex2Array`.

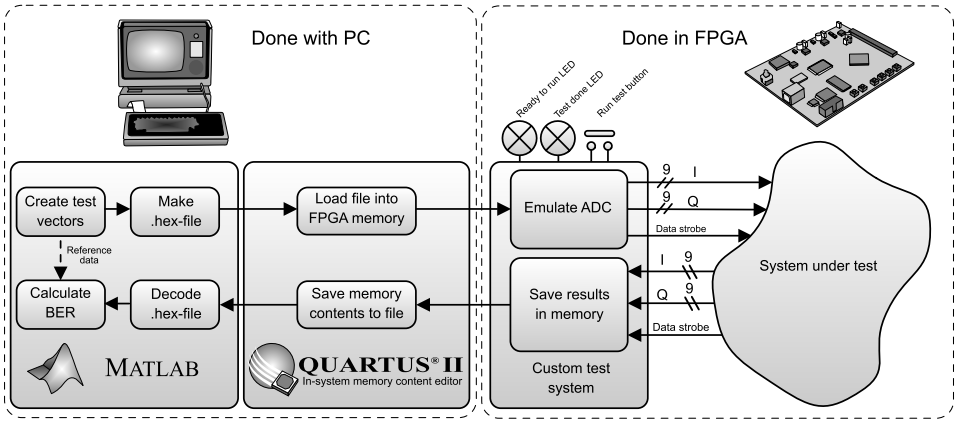


Figure 7.3: *The work flow for testing the frequency estimator includes; creating the input vectors, loading it into the FPGA, running the test, retrieving the output and calculating the BER*

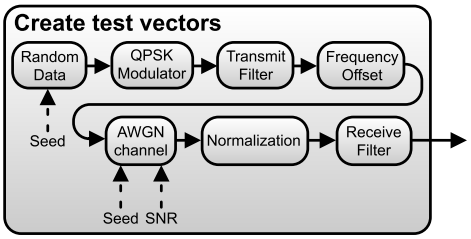


Figure 7.4: *The first block in the test system in Figure 7.3. The block constructs a vector on the basis of a data seed, a noise seed, and a SNR.*

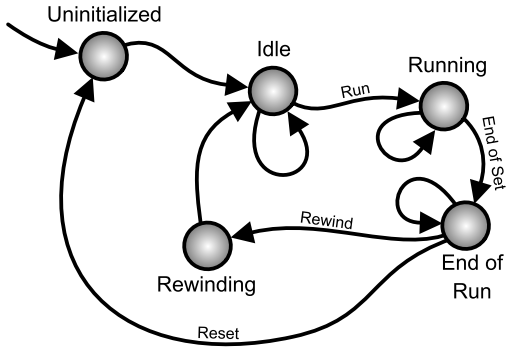


Figure 7.5: *The finite state machine governing the outer part of the test system in the FPGA. The transitions without labels are the default for the states if they receive no other input.*

The input samples might not fit into the integrated memory, as the memory is limited and the amount of samples needed for statistical significance at low SNR is considerable. For this reason the files has to be split, the MATLAB script `makeTestVectors` takes care of this.

Running a Test The test system in the FPGA takes only one user input and gives two feedbacks; that is the run command and the ready-to-run and end-of-test feedbacks. When a set of input samples has been loaded, the ready-to-run feedback is sent to the user by lighting up an LED on the board. Then the user can execute the test by pressing the run command button. When the entire set has been processed, the end-of-test LED lights up to indicate to the user that he can upload the output to his computer using the In-System Memory Content Editor.

Processing Output The MATLAB script `calculateBer` takes the output file (or files), converts it into a MATLAB array and compares this to the reference data provided by `makeTestVectors`. The outputs is the bit error rate (BER) vs. SNR and the relative angle error also computed in the initial simulations.

7.2.2 The FPGA part

As shown in Figure 7.3 the FPGA part of the test system is divided into an outer and an inner system. The outer system takes care of the input from, and output to the user along with the routing of the input and output of the phase estimator to the ADC emulator.

The finite state machine that is running in the outer part of the test system is shown in Figure 7.5. The user has three buttons as input to the test system: A run button, a rewind button, and a reset button, and there are two outputs to the user: A ready to run LED and an end of run LED. The buttons changes the states as shown in the state machine diagram. The ready to run LED flashes when the state is “Idle” and the end of run LED is constantly on when the state is end of run. The rewind button resets the address counter in the ADC emulator but leaves the delay elements of the phase estimator untouched, this way more files can be run in sequence. The reset button on the other hand resets everything.

The ADC emulating block is an interface to the internal memory of the FPGA, holding the

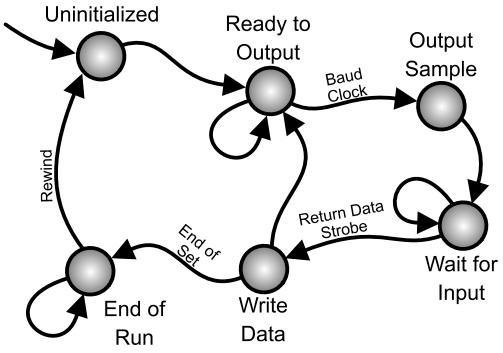


Figure 7.6: *The finite state machine governing the ADC-emulating block of the test system. The transitions without labels are the default for the states if they receive no other input.*

stimulation data for the phase estimator. When supplied with a baud rate signal, it outputs the data in the memory as if it was being sampled at the given baud rate. It employs a binary counter to traverse all of the addresses of the memory.

The finite state machine for the ADC emulator is shown in Figure 7.6. It starts in the "uninitialized" state. Here it sets the address counter to the highest possible address, so that it is ready for the first count, wraping around to the 0x00 address. The "Ready to Output" state is an idling state where the system is waiting for the first signal from the baud rate clock, when this is received it moves on to the "Output Sample" state. Here the next sample in the memory is output and the data strobe asserted. Then the emulator waits in the "Wait for Input" state for a data strobe from the phase estimator. When this comes, the data is written to memory and the emulator once again is ready to output a sample. If, however, the counter has reached the last address and asserts the carry-out, the state changes to "End of Run". The emulator then stops counting and signals to the outer test system that the entire memory has been traversed. The state machine has an extra state, the "Error" state, that is not displayed in the figure. This is the state that the emulator defaults to if the input is undefined, or if a baud clock signal is received while the emulator is in the state "Wait for Input".

The same memory is used for both the input to and the output from the phase estimator. This is done in order to maximize number of samples in the test vectors, and thus reduce the confidence interval of the calculated bit error rate. But doing this imposes some constraints on the read/write cycles. This means that the phase estimator must react on the recently output data before the next baud rate clock signal. An example of a memory write/read cycle is presented in Figure 7.7. Here we see the assertion of the incoming data strobe from the phase estimator, that the data is written into the memory and the next sample is output. In this example, the rising edge *a* of the system clock moves the state of the emulator into "Write Data" where the write enable signal is sent to the memory that writes the data on the data-in port (not shown) into the memory (thus changing the data on the output). On the rising edge *b* the state changes to "Ready to Output" that again changes to "Output Sample" on the rising edge *c* because of the Baud Clock signal being high. The new data is available on the data-out and the state changes to "Wait for Input" on the rising edge *d*. The write/read operation, in the end, takes three clock cycles i.e. 60 ns at a clock frequency

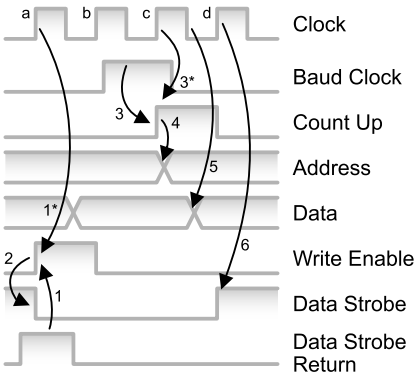


Figure 7.7: *Write cycle for the test system memory. Here, the phase estimator is delivering a data strobe (1). This combined with the rising edge a (1*) asserts the Write Enable, that in turn deasserts the Data Strobe (2). The Baud Clock (3) combines with rising edge c (3*) and counts up the address counter (4). The memory content is clocked out on the falling edge c (5) and the Data Strobe is asserted on rising edge d (6).*

of 50 MHz. This is quite a lot compared to the baud time of 260 ns, but that is the price for reusing the memory for the output. A way to save a some time is to make a “shortcut” from the state “Write Data” to “Output Sample” so that the waiting state “Ready to Output” is bypassed if the baud rate signal is already available.

With the time for reading and writing deducted from the symbol time, the phase estimator has 200 ns to finish a calculation. In case pipelining are being used, the total execution time through the system might be some multiple of the symbol time plus 200 ns. The baud rate clock is generated by an internal phase locked loop (PLL) tuned to 260.4 ns. For prototyping purposes, the baud rate can be lowered by setting the PLL to a lower output frequency.

7.3 Phase Estimator

As previously mentioned the Phase Estimator block controls the other blocks in the algorithm and this is done through a Finite State Machine (FSM).

The FSM in Figure 7.8 shows the behavior of the Phase Estimator block. The block includes others blocks as depicted in Figure 7.1 and its main functionality is to control and provide inputs to the blocks. This means that the block also controls the pipeline of the implementation.

As seen by Figure 7.2 all the blocks are controlled using the strobe and done in- and output respectively. Therefore when a new input to the phase estimator is received the input is stored in a register until it is used in the "Complex Multiplication" block and also copied as input to the "Power Four" block. The other blocks inputs are also set as the previous outputs from the respective preceding blocks, thus forming a pipeline.

The next state of the "Phase Estimator" then waits for all the blocks to return a done signals

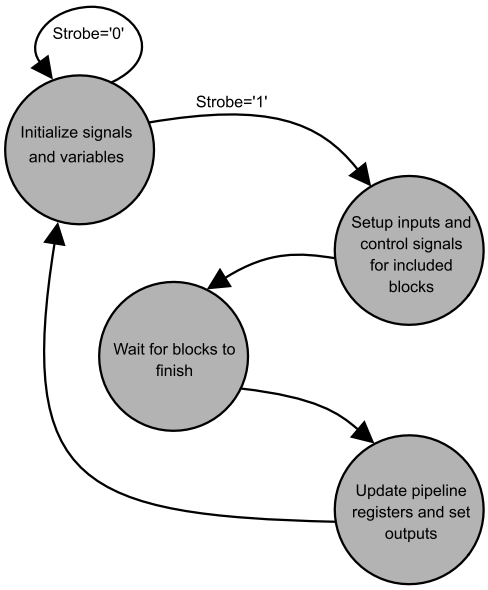


Figure 7.8: *Finite state machine of the Phase Estimator block.*

and then the output of the blocks is stored until new data is ready for the algorithm again.

7.3.1 Power four

This blocks functionality is to find the complex fourth power of the input. This is done by using one complex multiplier twice. This is also indicated by the FSM describing the block in Figure 7.9.

The complex multiplier that is used by this block is implemented using a Quartus II IP-core where the in- and output port widths are define. This IP-core is then implemented to be used by the "power four" block and as such the block is a matter of interconnections to the IP-core, and scheduling when which input is given to the IP-core, and when to store the outputs. The transitions between the nodes in the FSM are given as the rising edge of the input clock to the block. This does however demand that the IP-core is fast enough to calculate the output before the next rising edge. This is the case as seen by Section 6.3 when the clock frequency is 20 ns.

The block is tested as described in Section 7.1 using complex random inputs and the output of the block is then compare to the same fixed point calculation in MATLAB. The test shows that the block behaves as expected which means only small rounding errors differ from the results of the MATLAB calculation.

7.3.2 Filter

The filter block is as mentioned in Chapter 5 used to compensate for the uniform distributed frequency offset. The filter FSM is straight forward and can be seen in Figure 7.10.

As seen by the Figure the FSM controls the scheduling of the filter onto the, by the FPGA,

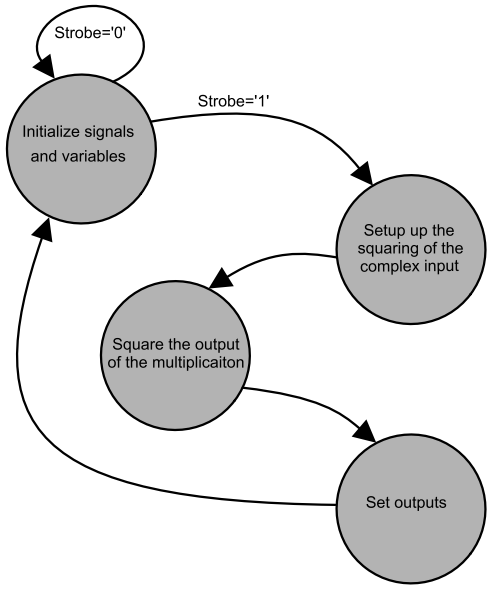


Figure 7.9: *Finite state machine for the Power Four block.*

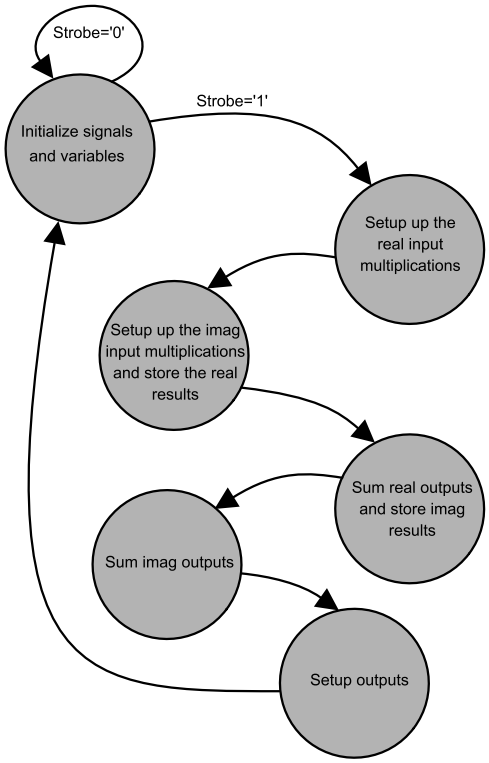


Figure 7.10: *Finite state machine for the Filter block.*

given resources. As seen only a number of multiplications equal to the filter order are executed at the same time, even though the double is needed by the filter because of complex inputs. When the results of these multiplications are stored they have to be summed. This is done in state 4 and 5. The additions are made in a sequential manner with a maximum of additions in parallel equal to the filter order for each part (real and imaginary).

The additions also introduce another problem. The outputs from the multiplications are 18 bits because the two inputs to the multiplier is 9 bit each, which are truncated to the 9 most significant bits (omitting one of the sign bits). When adding this number overflow will happen and as such the in- and outputs of the adders has to accommodate for this problem by making the word lengths of the input values longer. This is done by appending sign bits of the inputs to the input values. The number of sign bits needed are found by finding the maximum possible output value of the filter. This value is found by knowing that the inputs and filter coefficient are values between -1 and 1. This means that in the worst case the maximum output must be the sum of the absolute values of the filter coefficients. This is approximately equal to 108 for 128 filter taps. Because of this the number of bits has to be able to represent 108, therefore 7 bits has to be appended to the inputs.

When the outputs of the block is set the results are again truncated to nine bits, which is then used in the proceeding blocks.

Again the input test vectors are created using MATLAB and evaluated using ModelSim and the results are equal to the fixed point implementation in MATLAB and there are only minor differences due to roundoff errors.

7.3.3 Atan

This block finds the argument (angle) of the inputted complex number. This means, as mentioned, that it divides the inputs imaginary part with the real part and finds the arctan of the result. The FSM for the block is shown in Figure 7.11.

First the two input parts are divided and then the argument is found. To implement the division another IP-core is used, where the inputs word length are defined and the function is then generated by Quartus II. The function is not clocked and therefore this is done as seen by the FSM in the Figure.

The actual arctan function is implemented using the look up table (LUT) with interpolation as designed in Appendix D.1. This means that comparators are used to find the interval in which the input lies and the value of arctan in this interval is then estimated using a straight line. This means that this is implemented using comparators, one multiplier and one addition. The values used for the comparators and the coefficients for the linear approximation is shown in Appendix D.1.

The test vectors for this block is made in the same way as for the other blocks and the results are the same meaning that the results are approximately equal to the MATLAB implementation.

7.3.4 Phase Jump Detector

This block takes care of the making the angle output go from $-\pi$ to $+\pi$ instead of from $-\pi/8$ to $+\pi/8$. The FSM for this block is shown in Figure 7.12.

As seen by the figure the essential state is to determine whether a phase jump has taken place. This is done be comparing the inputs sign and the previous inputs sign and if these

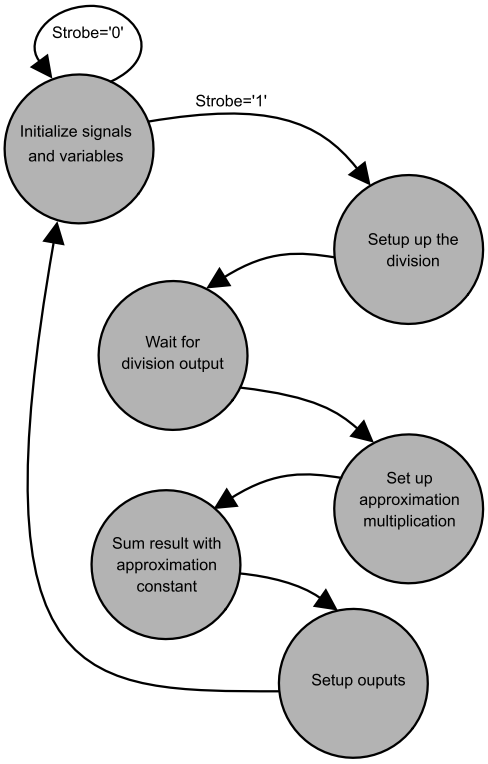


Figure 7.11: *Finite state machine for the Atan block.*

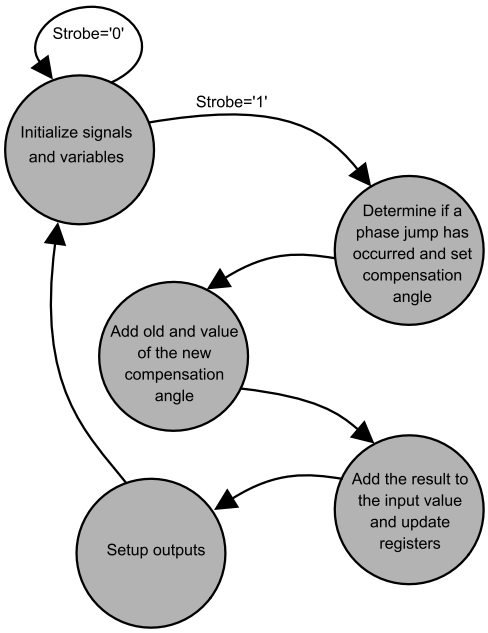


Figure 7.12: *Finite state machine for the Phase Jump Detector block.*

are different a phase jump has taken place or the input has crossed zero. Because a zero crossing does not necessarily mean a phase jump this is taken care of by checking if the absolute value of the current input is larger than $pi/16$. This could in practice give problems for large frequency offsets, but not in this case due to the chosen f_{max}. Therefore if the system is implemented in a scenario with large frequency offsets care has to be taken in choosing this threshold value. With a given sample frequency the maximum f_{max} for this design is given by $1/(2 \cdot 16 \cdot \text{sample time})$. The other states in the Figure are implemented as written in the Figure.

The block is tested with both increasing and decreasing input values in the entire input range and it behaves as expected.

7.3.5 Sine/Cosine

The block finds the sine and cosine values of the input and outputs these as the real and imaginary numbers for the complex phase estimator. The FSM for this block is similar to the one for "Atan" block as seen in Figure 7.13.

As shown in the figure the sine and cosine values are calculated in parallel which naturally increases the number of resources used for the given time instant, but it has been found the saving the comparators, one multiplication and one addition does not significantly improve the performance such that e.g. a longer filter could be used.

Here the scaling of the coefficients from the Atan block also come into account, because this block also needs to scale the coefficient which is multiplied onto the input. The coefficient is scaled by a factor of π, and the coefficient which is added to the result of the

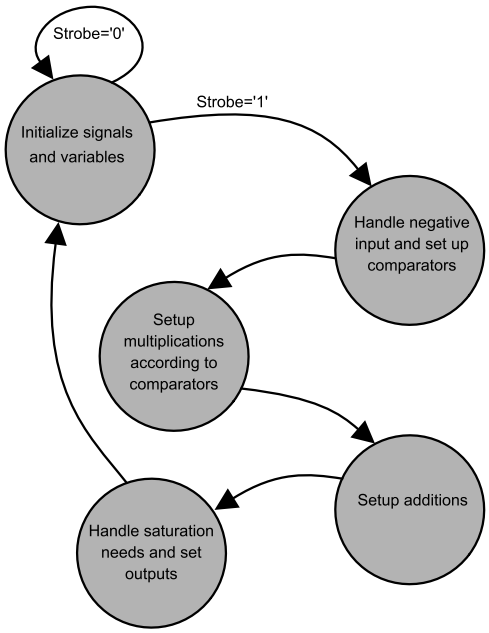


Figure 7.13: *Finite state machine for the Sine/Cosine block.*

multiplication remains the same.

Again the test is conducted in the same manner as for the other blocks, and ending up with the same positive result.

7.4 Integration

The integration of the system is made in two steps. First the Phase Estimator is constructed with all the blocks that it includes and second the Phase Estimator is included into the Test system.

Because the Phase Estimator and Test System blocks is designed this section concerns the output of the tests.

The tests for the Phase Estimator is conducted using test vectors designed in MATLAB with values similar to the values used for the performance tests. Because of this the values can also be compared to the MATLAB fixed point implementation which should reveal the same similar outputs in ModelSim. This is and should be the case as all the parts of the algorithm is tested and this test only test the integration of the blocks.

In similar manner the Test System integration test is constructed using the same test vectors as for the Phase Estimator integration test and thus the result should be same as this block also includes the includes other block. Because of this the test only tests the interconnection between these blocks. Along with the test vectors different control signals for this block emulating the user interacting is also included in the test vectors. The result of the test is compared to the results from the Phase Estimators integration test and they are the same and therefore the system is concluded to have the desired behavior.

Chapter 8

Test Results

Two different analyses of the data from the tests has been done. The one is the bit error rate as a function of the signal-to-noise ratio (BER vs. SNR), the other is the distribution of the relative phase difference between the original modulated signal before frequency offset and the frequency corrected signal from the system. The tests have been conducted with offsets of: 0, 10, 100 and, 1000 Hz at SNRs between -10 and 10 dB.

The BER vs. SNR analysis is not a direct measure of the performance of the frequency estimator. It actually measures the performance of the entire system, right from the modulation scheme to the wireless channel. However, we can construct a mathematical model for a perfect system with an additive white Gaussian noise channel. When comparing the performance of the system with that of the model we will see how close the system performance comes to the theoretical case. If the system performs a perfect frequency correction, the results should match those of the theoretical calculations. If the frequency correction is less than perfect, the system will increase the BER as more samples are allowed to cross the decision boundary (see a description of this in Chapter 3). The plots are shown in Figure 8.1.

In all four frequency offset settings does the system perform well, in the sense that the results tends to follow the theoretical performance, they are, however, generally slightly worse. Figure 8.2 shows the difference between the phase of the original modulated signal and the signal processed by the system along with the phase correction done by the system for the test run with 4 dB SNR and 100 Hz offset. It is seen from the figure that the approximations of trigonometric functions results in sinusoidal-like errors. The shift in correction from $-\pi$ to π at 3.5 ms may seem extreme, but as angles are only defined within this region, the shift is not at all noticeable in the system.

The implemented system does not output the phase estimate directly, so the phase correction must be deduced by comparing the inputs and outputs. The two plots are constructed from the difference between the original modulated signal and the output of the system and the difference between the offset-impaired input signal and the output respectively. This means that the phase error plot also includes the phase offset introduced by the channel noise.

At some points in the plots in Figure 8.1 the BER "jumps" to around 0.5. The reason is a wrong guess of which half-quadrant the correct phase is within. Such an event can be seen in Figure 8.2 around 0.7 ms where the correction (and thus the relative phase error) shifts

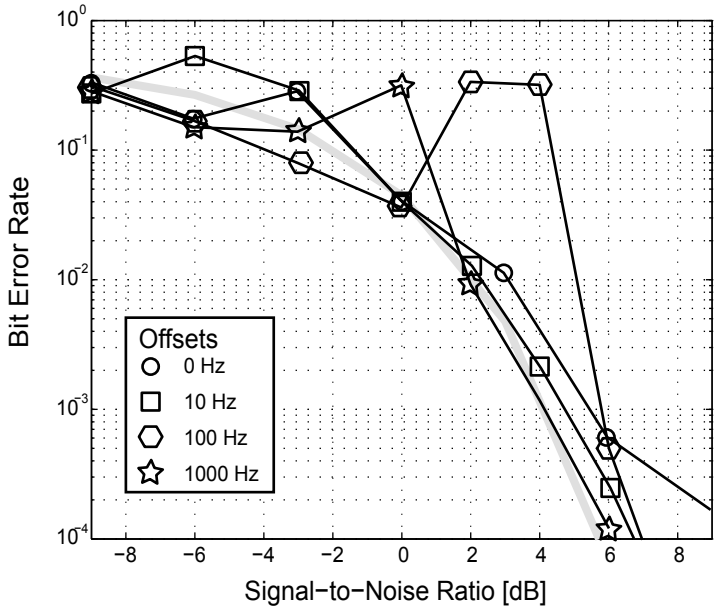


Figure 8.1: *BER vs. SNR plots of the test data. The theoretical BER performance is outlined in light gray. The relative phase error of the 4dB run from the 100 Hz set is shown in Figure 8.2*

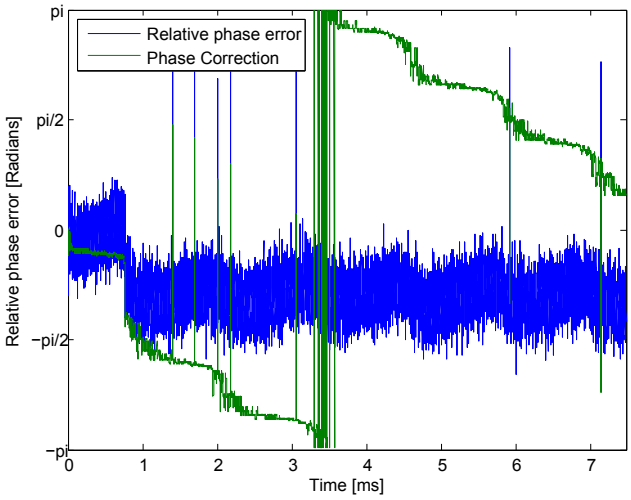


Figure 8.2: *The relative phase error between the original non-offset signal and the output of the system along with the phase correction done by the system. This plot is from the run shown in Figure 8.1 with a 100 Hz offset and an SNR of 4 dB. Notice the "half-quadrant" error at around 0.7 ms resulting in the poor BER.*

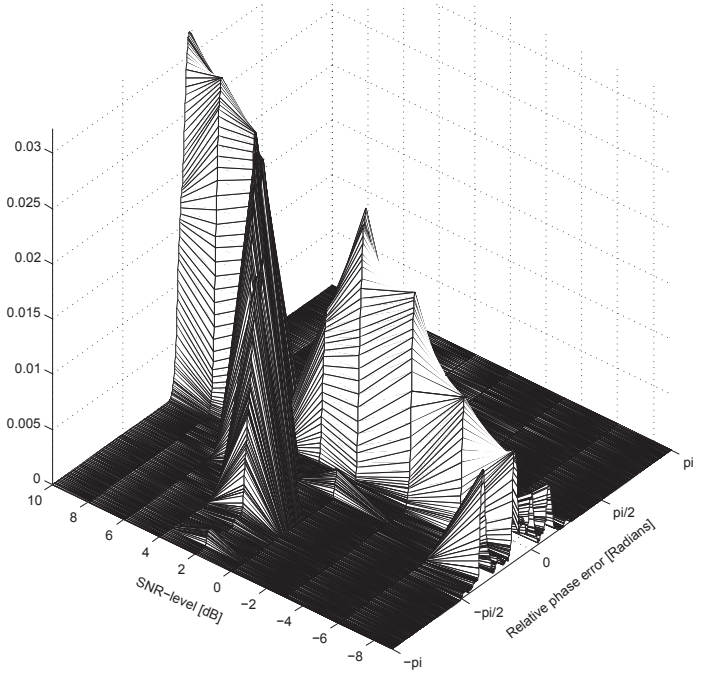


Figure 8.3: *The distribution of the correction errors for the 100 Hz offset test. Notice the two wrong half-quadrant guesses at the SNRs 2 and 4, resulting in the poor BER shown in Figure 8.2. The distributions are normalized with reference to the greatest value in the set, and filtered by a moving average filter to print better.*

$\frac{\pi}{4}$ radians. In Figure 8.3 the distributions of the correction errors from the 100 Hz offset test run (shown in Figure 8.1) are presented. The correction error is the difference between the phase correction (or equivalently the phase estimate) seen in Figure 8.2. In this figure the two wrong half-quadrant guesses are visible, and it is seen that if some system were to correct this offset, the overall system would not suffer from such high BER.

8.1 Discussion

The biggest defect of the system is that it sometimes makes wrong "guesses" about the phase of the signal. The operations throughout the algorithm creates an ambiguity for every $\pi/4$ radians on the unit circle. The shifts between these half-quadrants are done by the phase jump detector, when a sudden change of phase is detected, but sometimes the noise spikes leads the phase jump detector to erroneously change to another half-quadrant. But this static offset seems quite stable, and could possible be corrected by a channel estimator, if the phase estimator is incorporated into a larger radio system. Figure 8.3 shows that the phase error is quite stable around 0 even at low SNRs. If the phase jump detector was doing a lot of wrong "guesses", the surface would have more peaks per SNR like the test run at -9 dB. But this is not the case.

Chapter 9

Conclusion

This project concerned the development and implementation of an algorithm which compensates for the effects that frequency drift has on the wireless communication links in 3G telephony system.

The nature of frequency drift in wireless transmission schemes has been investigated in theory and the implications are shown to significantly degrade the transceiver system's performance. This shows the necessity of an algorithm, which handles the problem. The algorithm is also constrained by the structure of 3G such that the algorithm does not demand changes to the RF front end, but instead can be "plugged" into existing systems.

Three algorithms has been analyzed and one was chosen to be the best suited for implementation in the 3G scenario. The algorithm is an extension of an algorithm developed by D. Divsalar and M.K. Simon [3]. The new algorithm was extended from BPSK to QPSK, and therefore it was analyzed thoroughly to verify that the algorithm keeps its good properties when used for QPSK systems. The algorithm consists of 2 parts; a phase estimator and a phase jump detector. The phase estimator averages the phase error by using a filter and the phase jump detector removes ambiguities introduced by the phase estimator. Simulations of the developed extended algorithm done in MATLAB shows good performance in terms of BER, which indicates that the algorithm is suited for implementation in the 3G transceiver systems, and that the aforementioned filter should be 128 taps long.

This choice of platform is based on analyses of the complexity of the algorithm and it is found that both parallelism and pipelining should be utilized to obtain an implementation which could work within the real time requirements for the system. The platform chosen for the implementation was the Altera Cyclone 3 FPGA with 132 embedded multipliers. To implement the algorithm onto this FPGA the word lengths by which variables should be represented was found. This was done through an analysis of the fixed point effects on the filter and it was found here that 9 bit gives a suitable precision.

The actual implementation of the algorithm was done within a test system which makes it possible to interact with the implementation on the FPGA and thus to evaluate the performance of the implementation, which is then compared to the performance of the algorithm in MATLAB. It was found that the algorithm almost performs as good as the MATLAB implementation and as such the results from MATLAB can be extrapolated when changing values the in MATLAB implementation, such as the filter length or trigonometric functions.

On this ground it is concluded that the developed algorithm can reduce the effects of frequency drift in 3G transceiver systems, within the structure constraints set by the 3G front end which are not changed.

9.1 Further work

There are some topics which should be researched further on basis of the initial research done in this project. The following describes these topics which is of interest for future work.

An interesting topic is to look at the distribution function of the frequency error. The algorithm developed in this project assumes the frequency error to be uniform within some limit, but it has not been researched if there are better functions which exist based on an even more detail analysis of when the problem occurs, e.g. urban areas traveling with car.

The filter length could be adaptive. As seen by the simulation results, the long filters performs better with small frequency offsets, but worse for larger frequency offsets. It could be analyzed if there is an optimal length for this filter or if it could be changed adaptively to improve the performance of the algorithm.

The test system developed for the algorithm could be used for many other algorithms and not necessarily wireless communication algorithms. The test system is very generic and the timing setup is easy to change, which makes it ideal to verify the performance of algorithms implemented on FPGAs by utilizing MATLAB.

Appendix A

Phase Estimation Rewritting

$$
\begin{aligned}
p(\bar{r} \mid \phi, \theta) &= \left(\frac{1}{2\pi\sigma_n^2}\right)^N e^{-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}\left|r_{k-i}-\sqrt{2P}e^{j(\phi_{k-i}+\theta)}\right|^2} && \text{(A.1)}\\
&= \left(\frac{1}{2\pi\sigma_n^2}\right)^N e^{-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}\left(r_{k-i}-\sqrt{2P}e^{j(\phi_{k-i}+\theta)}\right)\cdot} \\
&\cdots \quad \left(r_{k-i}^*-\sqrt{2P}e^{-j(\phi_{k-i}+\theta)}\right) && \text{(A.2)}\\
&= \left(\frac{1}{2\pi\sigma_n^2}\right)^N e^{-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}\left(r_{k-i}r_{k-i}^*+\sqrt{2P}e^{j(\phi_{k-i}+\theta)}\sqrt{2P}e^{-j(\phi_{k-i}+\theta)}-\right.} \\
&\cdots \quad \left. r_{k-i}\sqrt{2P}e^{-j(\phi_{k-i}+\theta)}-r_{k-i}^*\sqrt{2P}e^{j(\phi_{k-i}+\theta)}\right) && \text{(A.3)}\\
&= \left(\frac{1}{2\pi\sigma_n^2}\right)^N e^{-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}|r_{k-i}|^2+\left|\sqrt{2P}e^{j(\phi_{k-i}+\theta)}\right|^2-} \\
&\cdots \quad 2\Re\left[r_{k-i}\sqrt{2P}e^{-j(\phi_{k-i}+\theta)}\right] && \text{(A.4)}\\
&= \left(\frac{1}{2\pi\sigma_n^2}\right)^N e^{-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}|r_{k-i}|^2+2P-2\Re\left[r_{k-i}\sqrt{2P}e^{-j(\phi_{k-i}+\theta)}\right]} && \text{(A.5)}\\
&= Fe^{-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}-2\Re\left[r_{k-i}\sqrt{2P}e^{-j(\phi_{k-i}+\theta)}\right]} && \text{(A.6)}\\
&= Fe^{-\alpha\sum_{i=1}^{N}-\Re\left[r_{k-i}e^{-j(\phi_{k-i}+\theta)}\right]} && \text{(A.7)}\\
&= Fe^{\alpha\sum_{i=1}^{N}\Re\left[r_{k-i}e^{-j(\phi_{k-i}+\theta)}\right]} && \text{(A.8)}
\end{aligned}
$$

Appendix B

Phase Estimator Derivation

$$\begin{aligned}
\sum_{i=1}^{N} \Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^3 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right] \quad &\cdots \\
-\Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^3 \Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right] \quad &= \quad 0 \qquad \text{(B.1)} \\
\sum_{i=1}^{N} \Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right] \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right] \quad &\cdots \\
\cdot\left(\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 - \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2\right) \quad &= \quad 0 \qquad \text{(B.2)} \\
\frac{1}{2}\sum_{i=1}^{N}\left(r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right)^4 - \left(\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 - \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2\right)^2 \quad &\cdots \\
-4\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \quad &= \quad 0 \qquad \text{(B.3)} \\
\sum_{i=1}^{N}\left(r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right)^4 - \left(\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^4 + \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^4 \quad &\cdots \\
-2\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2\right) \quad &\cdots \\
-4\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \quad &= \quad 0 \qquad \text{(B.4)} \\
\sum_{i=1}^{N}\left(r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right)^4 - \quad &\cdots \\
\left(\left|r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right|^4 - 2\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \quad &\cdots \\
-2\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2\right) \quad &\cdots \\
-4\Re\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \Im\left[r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right]^2 \quad &= \quad 0 \qquad \text{(B.5)}
\end{aligned}$$

$$\sum_{i=1}^{N}\left(r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right)^4-\left|r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right|^4 = 0 \tag{B.6}$$

$$\sum_{i=1}^{N}\left(r_{k-i}e^{-j\hat{\theta}_{ML,k}}\right)^4-|r_{k-i}|^4=0 \tag{B.7}$$

$$e^{-j4\hat{\theta}_{ML,k}}=\frac{\sum_{i=1}^{N}|\,r_{k-i}\,|^4}{\sum_{i=1}^{N}r_{k-i}^4} \tag{B.8}$$

$$e^{j\hat{\theta}_{ML,k}}=\sqrt[4]{\frac{\sum_{i=1}^{N}r_{k-i}^4}{\sum_{i=1}^{N}|r_{k-i}|^4}} \tag{B.9}$$

Appendix C

Simulink Simulation Model

Using Simulink for modeling is convenient for both constructing and presenting the model. The graphical editor enables quick and robust code generation and a nearly one-to-one mapping of the data flow diagram in [2].

The model is divided into four parts: A transmitter, a channel, a receiver and an error rate calculator. The Simulink top-system is shown in Figure C.1. The receiver part holds the proposed algorithm, whereas the transmitter and channel are providing the test input. The error rate calculator measures the bit error rate by comparing the data input for the transmitter to the data output of the receiver.

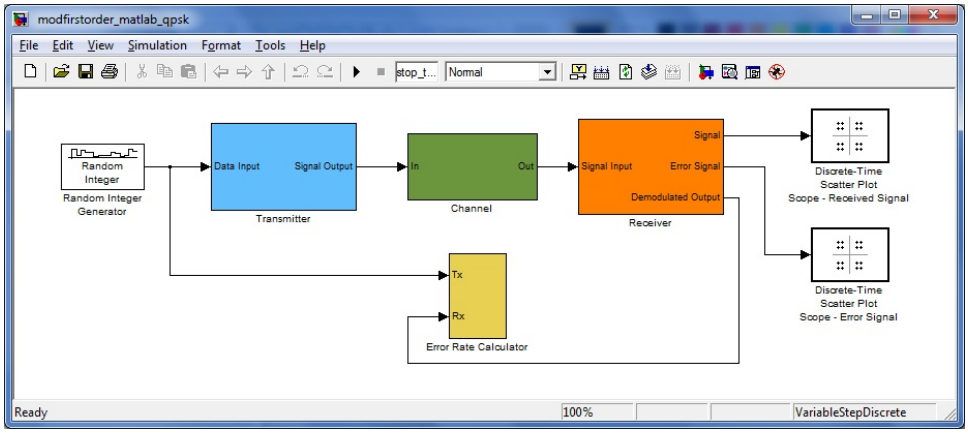


Figure C.1: *The top system of the Simulink model is divided into a transmitter, a channel, a receiver and an error rate calculator. Each of these boxes is a subsystem containing more components.*

C.1 The Transmitter

The Transmitter takes an input in the form of an integer between 1–4. This input is converted into a complex QPSK-modulated baseband signal and filtered by a root-raised cosine

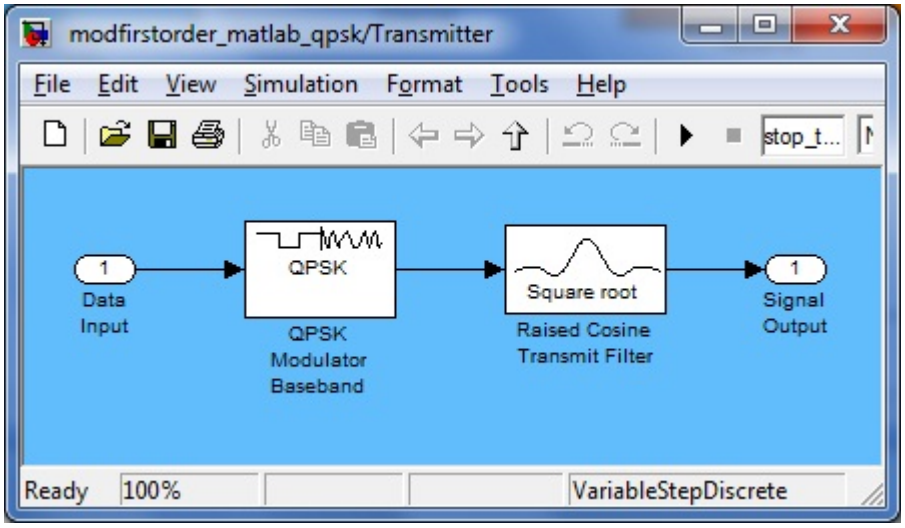


Figure C.2: *The Transmitter model uses a build-in QPSK baseband modulator from the communications block-set in Simulink and a root-raised transmitter shaping filter.*

filter. The output is up-sampled 4 times, as this matches the input specified for the receiver. The Simulink subsystem is showed in Figure C.2.

C.1.1 QPSK Baseband Modulator

The input to the Modulator is an integer in the range 1–4. Each of the integers is mapped to a corresponding messaging point in the complex plane. The points are Gray coded in order to minimize the bit error, as described in the specification.

C.1.2 Root-Raised Cosine Transmit Filter

The filter up-samples the symbols from the modulator and shapes them according to the root-raised cosine response with a roll-off factor of 0.22 specified in [12, Sec.6.8.1]. A group delay of 4 symbols is introduced, as the filter coefficients at that width are negligible.

C.1.3 The Channel

The purpose of the channel subsystem is to introduce the frequency offset and additive white Gaussian noise (AWGN). The Simulink subsystem is displayed in Figure C.3.

C.1.4 Phase/Frequency Offset

As the signal from the transmitter is a complex baseband signal, the frequency offset is accomplished by turning this signal around the origin in the complex plane at the given offset frequency. This is done with the Phase/Frequency Offset block from the Communications Toolbox. The block takes the offset frequency (both positive and negative) as input.

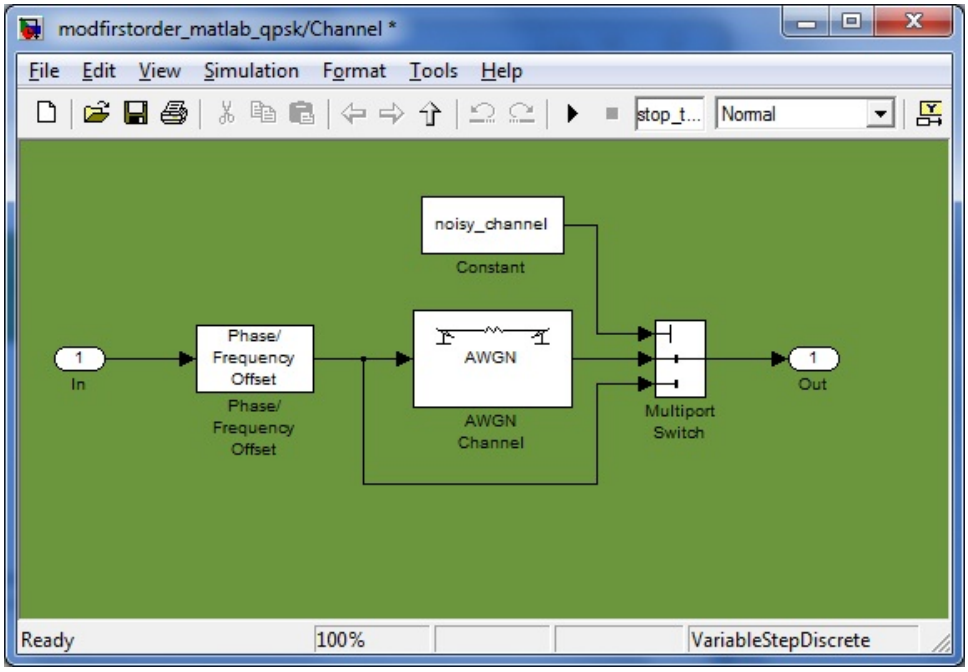


Figure C.3: *The channel models both phase- and frequency offset and adds white noise. The noise can be bypassed if needed.*

C.1.5 Additive White Gaussian Noise

White Gaussian noise is then added to the signal using the AWGN block. This block can also be bypassed if the simulation demands it. The variance of the noise is determined by the SNR of the signal, for this calculation, the strength of the signal needs to be known. As the length of the complex numbers produced in the QPSK modulator is always one, the signal power is also one, and so this is put into the AWGN block.

As some simulations might need to run without noise present, a switch has been included in order to bypass the AWGN block, this is done from the MATLAB workspace.

C.2 The Receiver

The receiver includes a root-raised cosine receive filter, the proposed algorithm to correct frequency offset and a baseband QPSK demodulator. And outputs the estimate of the data sequence input to the transmitter. Due to group delays in both receiver and transmitter, the data is delayed in time. The Simulink model is shown in Figure C.4.

C.2.1 Root-Raised Cosine Receive Filter

The receive filter is a time-reversed version of the shaping filter in the transmitter i.e. a root-raised cosine filter with a roll-off factor of 0.22. For the simulation, in order to match the filter to the transmit filter, a group delay of 4 symbols is introduced.

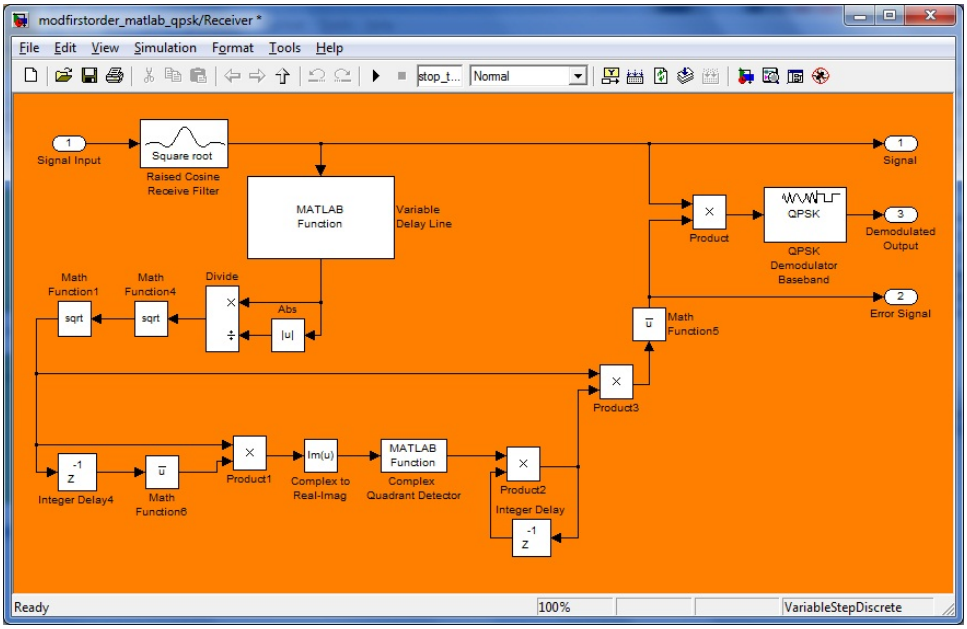


Figure C.4: *The receiver mainly consists of the phase estimation part and the phase jump detector part. In the end a QPSK demodulator estimates the output symbols.*

C.2.2 Phase Estimating Algorithm

The Simulink model closely resembles the derived algorithm showed in Section 5. However two MATLAB functions are used instead of built-in Simulink functions. The first is the Variable Delay Line and the second the Complex Quadrant Detector.

In order to evaluate the impact of extending the delay line of the algorithm, a MATLAB function has been written that enables the simulation to work with variable delay line lengths. It needs to be initialized through the MATLAB workspace by passing it a vector with a length of equal to the desired length of delays. The entries of the vector should be the sinc function filter coefficients of each of the delays (see Section 5). Before the filtering, the values on the delay line are raised to the power of four.

The Complex Quadrant Detector acts as a part of the 90° phase jump detector. It monitors the imaginary part of the resulting vector of the one-delayed product of the error signal (dicussed in Section 5.2). If it raises above a value determined though the MATLAB workspace, it will output $-j$ (imaginary j) if it on the contrary drops below the negative of the value a j is output, otherwise the output will be 1. Multiplying the output with the error signal, enables it to turn 2π in the complex plane, where it otherwise would only be defined within $-\pi/2$ and $\pi/2$.

C.2.3 Error Rate Calculator

The bit error rate (BER) is the metric used in most of the simulation test cases for evaluating the performance of the algorithm. The Error Rate Calculator compares the transmitted information stream to the one estimated by the receiver. The output is the ratio between the amount of wrongfully estimated bits and total amount of bits. Because of the group delay

in the filters of the transmitter and receiver, the comparison between the two information sequences is delayed accordingly.

The simulation works with symbol values between 1–4, but each symbol equals two bits. To obtain the BER, each symbol must be converted into bits before comparing. If e.g. a symbol 1 (bits 00) was transmitted but a symbol 2 (bits 01) was estimated, the symbol error rate is 1 whereas the BER is only 0.5. The Error Rate Calculator accomplishes this by converting each symbol into a vector with two entries, and comparing the entries in parallel. The output is written into the MATLAB workspace. The Simulink model is shown in Figure C.5.

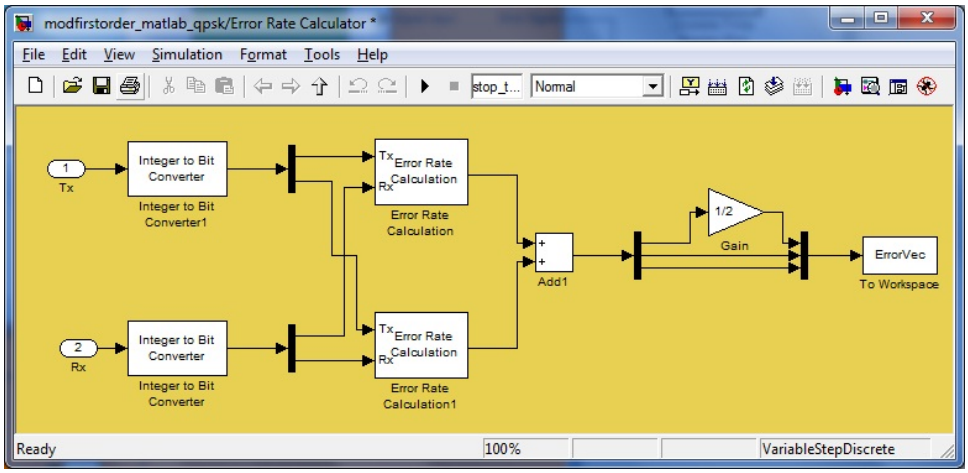


Figure C.5: *The error calculation compares sent and received bits rather than symbols, the reason why two calculators, as each QPSK-modulated symbol represents two bits.*

C.3 Simulation

The Simulink model is used for evaluating the functionality of the receiver, as well as providing a floating point reference for the fixed point results. The variable constants of the simulation, i.e. delay line length, decision threshold of the Phase Jump Detector, as well as the waveform, i.e. offset frequency and baud rate, are independent of the implementation of the algorithm. So the results of the Simulink floating point simulation is readily comparable with the results of any other implementation of the algorithm.

The simulation is run by executing the MATLAB "main" file. The program sets up the required constants, and runs the Simulink model. The output of the Error Rate Calculator is stored into the MATLAB variable "ErrorVec" as a vector containing the BER, faulty symbols and total received symbols.

Appendix D

Trigonometric Functions Approximations

To implement the trigonometric functions they have to be designed. As mentioned in Section 6.1 the operations are implemented as a piece wise linear approximation. To find these linear pieces MATLAB simulations are conducted.

First the error introduced by the approximation, that can be tolerated by the algorithm is found. This is done through simulations of the algorithm with noise added to the angle of the estimator. The noise is uniform within ranges with a mean value of 0 and a maximum value. This maximum value is changed to find the value where the error introduced by the noise is acceptable. In Figure D.1 the performance of the algorithm is shown for different values of this maximum error.

As seen by the figure an error of 0.01 gives almost the same performance in terms of BER as the case where no noise is introduced. Therefore the trigonometric approximations are designed such that they do not have a maximum error larger than the 0.01. The following sections describes the design of the approximations of the trigonometric functions and how they are made in order to minimize the look up table.

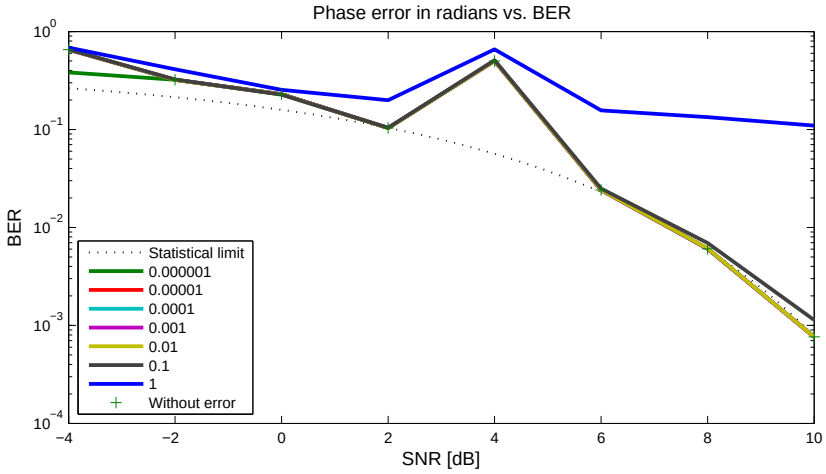


Figure D.1: *The performance of the algorithm with different maximum errors in radians introduced on the estimator angle. This simulation shows how sensitive the algorithm is to angle errors. The simulation is made with a frequency offset of 250 Hz. All errors less than 0.01 gives the same results as 0.01 and the results are therefore plotted on top of each other.*

D.1 Arctangent Approximation

To minimize the number of lookups the longest linear pieces are wanted, but the longer they are the more likely it is, that the error between the real arctan and the approximation will grow. Because of this the length of the pieces where maximum error is 0.01, which then results in the approximation respecting the requirements. To find the length brute force is employed using MATLAB.

To find the length a vector of values between 0 and 100 is made with a sufficiently small sampling interval. The sampling interval is determined on how accurate the point between these linear pieces are wanted. For this simulation 0.001 is found to be a suitable trade-off between simulation time and precision. Then the arctan of this vector is found. This arctan vector is used as a reference signal of which the lines approximate. The lines are made by using the polyfit function in MATLAB which uses the least squares method and then finding the error between the real arctan and the approximation. If the error is below 0.01 then the line is made longer and a new approximation is found. This is repeated until the error exceeds 0.01. Then the coefficients, the start and end point for the line are stored and then the next approximation is found in a similar manner. Thereby it is ensured that the maximum error does not exceed 0.01.

The results of the simulation is shown in table D.1 and Figure D.2. The output for the negative input values are found by the fact that $\text{atan}(-x) = -\text{atan}(x)$, as it is an odd function.

Start	End	a coefficient	b coefficient
0.001	0.502	0.9328	0.0073
0.502	0.941	0.6600	0.1438
0.941	1.473	0.4108	0.3783
1.473	2.232	0.2290	0.6469
2.232	3.463	0.1123	0.9089
3.463	5.765	0.0465	1.1388
5.765	10.976	0.0149	1.3230
10.976	27.140	0.0031	1.4561
27.140	100.00	0.0003	1.5331

Table D.1: *Table of the borders and coefficients of the linear approximation of arctan. The linear model is $y = ax + b$.*

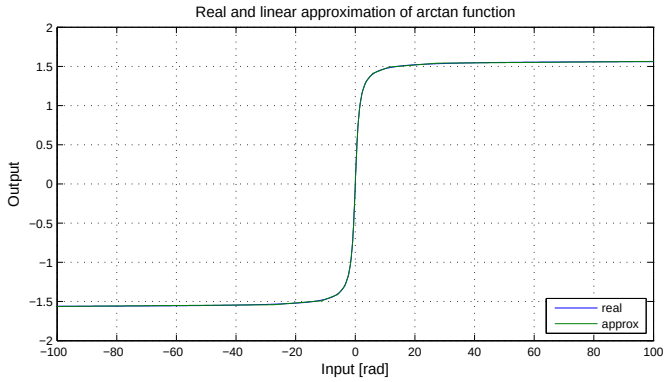


Figure D.2: *The linear piece wise approximation of arctan and the real arctan function. The two curves are almost on top of each other.*

D.2 Sine and Cosine Approximation

The sine and cosine approximations are made in the exact same way. The only difference are the negative input values where the cosine function is an even function and as such $\cos(-x) = cos(x)$. The sine function is odd which means that the output for negative input values are found as for the arctan approximation. The resulting linear approximations and the figure of the function is shown in Table D.2 and Figure D.3.

Start	End	a coefficient	b coefficient
0	0.5930	0.9483	0.0068
0.5930	0.9980	0.6981	0.1540
0.9980	1.3580	0.3829	0.4678
1.3580	1.7070	0.0397	0.9334
1.7070	2.0630	-0.3067	1.5241
2.0630	2.4530	-0.6308	2.1925
2.4530	2.9630	-0.9009	2.8554
2.9630	3.6720	-0.9725	3.0549

Table D.2: *Table of the borders and coefficients of the linear approximation of sinus. The linear model is $y = ax + b$.*

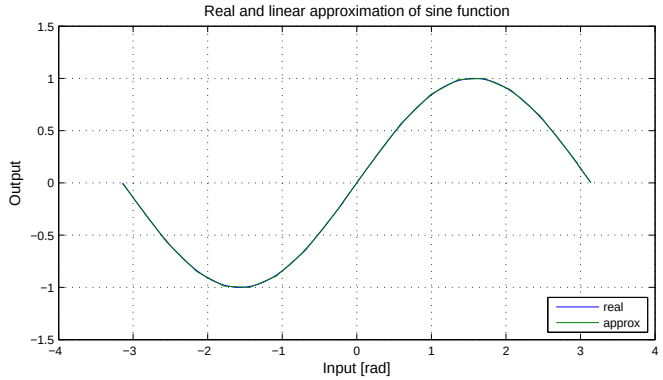


Figure D.3: *The linear piece wise approximation of sine and the real sine function. Again the curves are almost on top of each other.*

The cosine approximation is likewise shown in Table D.3 and Figure D.4.

With all these functions defined their performance can be simulated in form of BER plots. The result of this is shown in Figure D.5. As seen from the figure the performance is roughly equal to the performance of the of algorithm without approximations, except for high SNR. This can be corrected by creating a more accurate approximation, but results in bigger look up tables, therefore this is chosen as a reasonable compromise.

Start	End	a coefficient	b coefficient
0.0010	0.3520	-0.1736	1.0101
0.3520	0.7240	-0.5094	1.1278
0.7240	1.1660	-0.8057	1.3423
1.1660	2.0860	-0.9776	1.5363
2.0860	2.5050	-0.7464	1.0553
2.5050	2.8700	-0.4385	0.2849
2.8700	3.2190	-0.0981	-0.6915

Table D.3: *Table of the borders and coefficients of the linear approximation of cosine. The linear model is $y = ax + b$.*

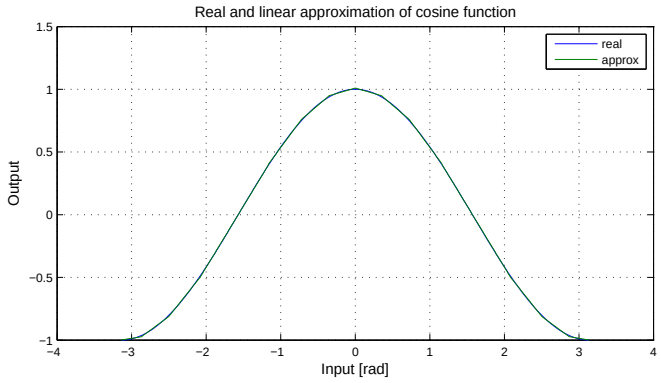


Figure D.4: *The linear piece wise approximation of sine and the real sine function. Again the curves are almost on top of each other.*

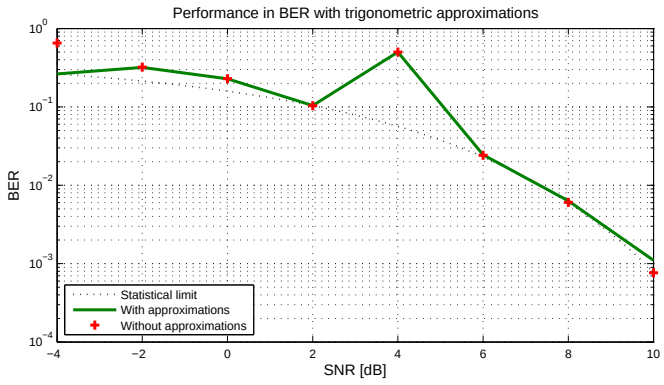


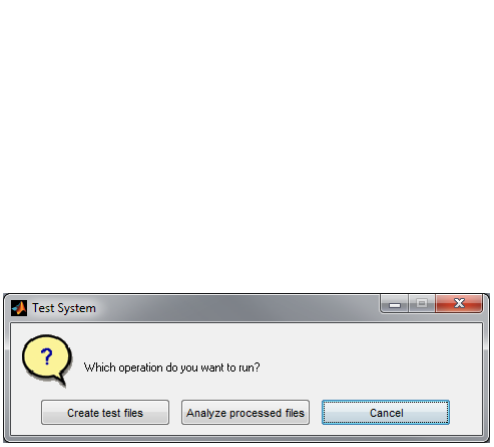
Figure D.5: *The performance of the algorithm in terms of BER with the trigonometric approximations. The two curves are very close to each other.*

Appendix E

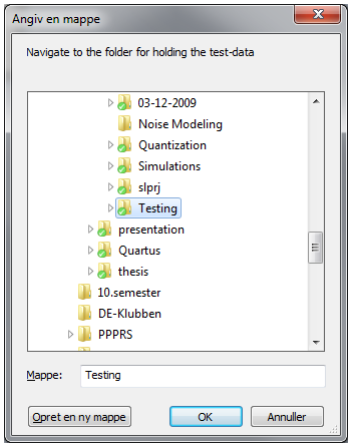
Test System

The test system is divided into two parts. The first part produces the input for the phase estimator, the second receives and analyzes the out put. The test system is implemented partly in MATLAB and partly in the FPGA. A block diagram of the overall test system is shown in Figure 7.3 in Section 7.2. The MATLAB part of the test system uses a graphical user interface to receive the parameters for the test set from the user. The input process is shown in Figure E.1 and the output processing in Figure E.3.

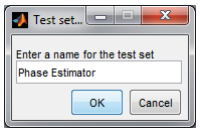
The MATLAB code uses some functions from the communications toolbox. These are the random data source, the QPSK modulator/demodulator, the square root-raised cosine transmit/receive filter, and the AWGN channel. The following MATLAB code is an example of how to create and process the test vectors:



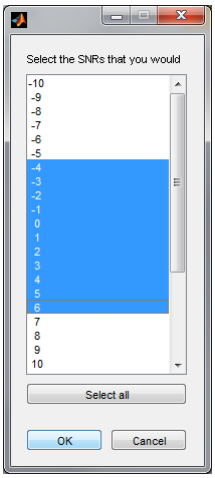
(a) *Choose to create files*



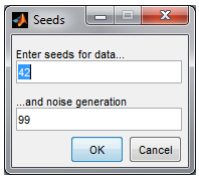
(b) *Specify a folder for the files.*



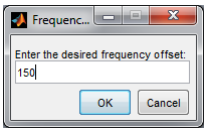
(c) *Enter a name for tis test set.*



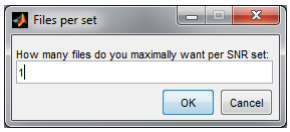
(d) *Choose the SNR levels.*



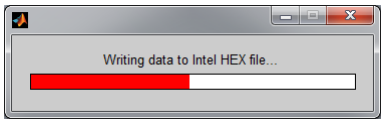
(e) *Set the seeds.*



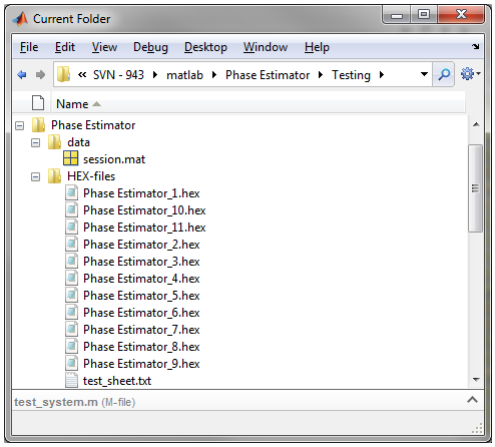
(f) *Set the frequency offset.*



(g) *Set a maximum amount of files per SNR.*



(h) *The program creates the files.*



(i) *The folder structure with HEX files.*

Figure E.1: *The process of creating the HEX files for a test set.*

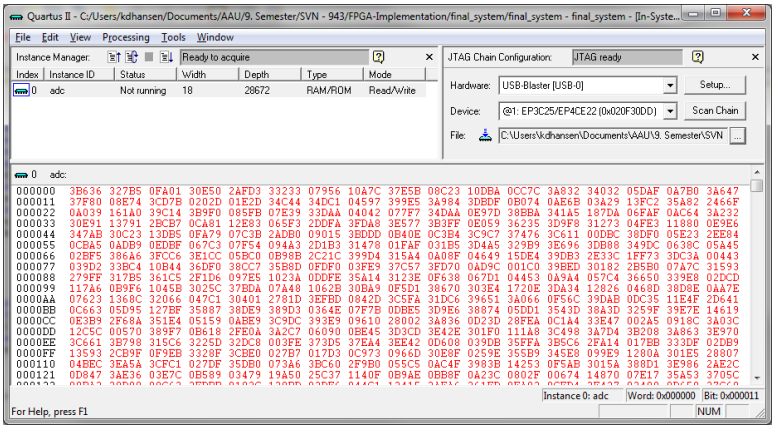
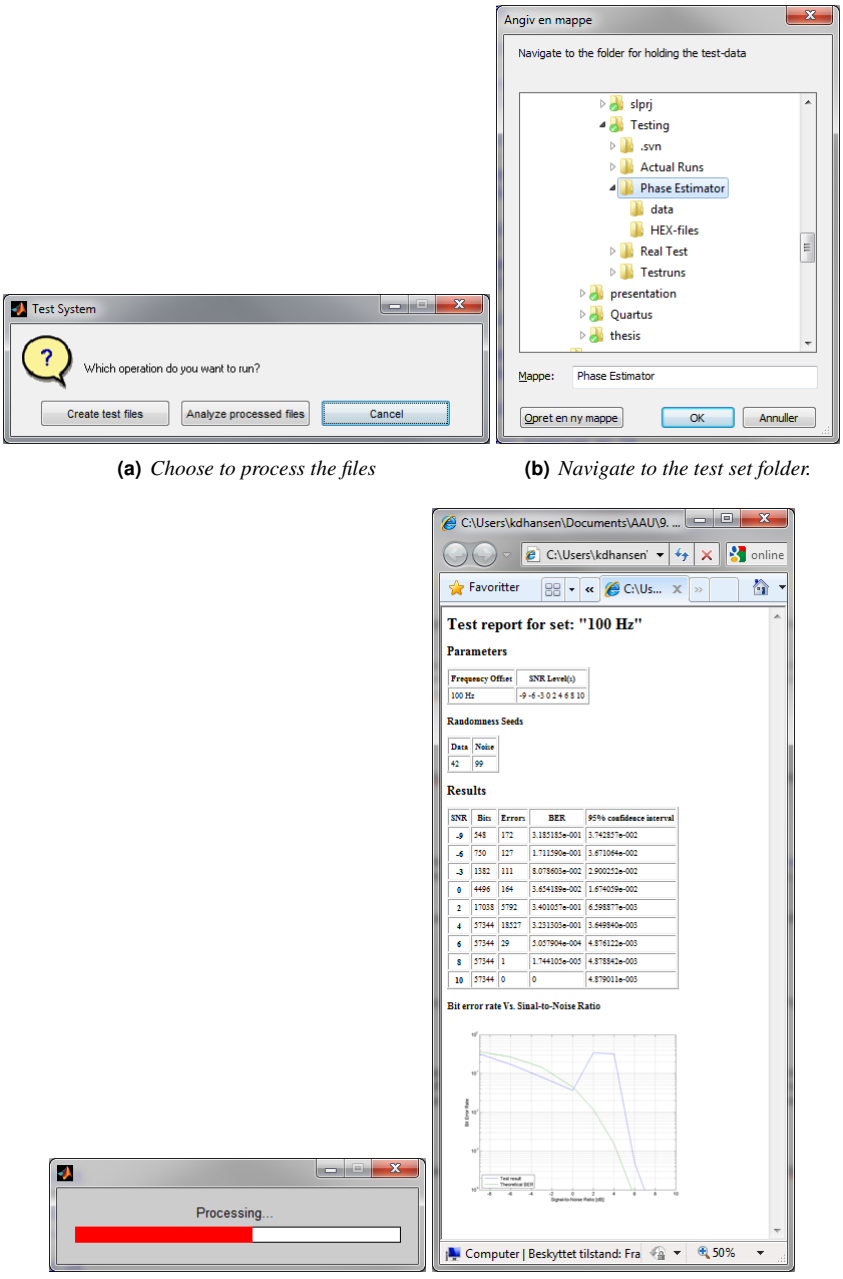


Figure E.2: *The Altera Quartus In-memory contents editor, reading the contents of the internal memory of the embedded test system.*



(a) *Choose to process the files*

(b) *Navigate to the test set folder.*

(c) *The program processes the files.*

(d) *A test report is produced.*

Figure E.3: *Processing the output files from the phase estimator.*

E.1 Creating Test Vectors

A bitstream is created:

```
reference_bitstream = ...
   randi(randomstream_data, [0 1], number_of_samples, 1);
```

The bitstream is modulated and transmit-filtered:

```
modulated_signal = ...
   modulate(qpsk_modulator, reference_bitstream);
transmit_signal = ...
   rcosflt(modulated_signal, 1, oversampling_factor, 'filter', rrc_filter);
```

A frequency offset vector is created and multiplied width the signal:

```
offset_frequency_signal = ...
   exp(1i*2*pi*offset_frequency/(baud_rate*oversampling_factor) ...
      *(1:length(transmit_signal)));
impaired_signal = ...
   transmit_signal .* offset_frequency_signal';
```

Noise is added to the signal:

```
noisy_signal = ...
   awgn(impaired_signal, test_snrs(idx), 'measured', randomstream_noise);
```

The signal is receive-filtered and down-sampled to one sample per symbol. The extra couple of samples coming from the filter are truncated:

```
filtered_signal = ...
   rcosflt(noisy_signal, 1, oversampling_factor, 'Fs/filter', rrc_filter);
downsampled_signal = ...
   downsample(filtered_signal,oversampling_factor);
downsampled_signal = ...
   downsampled_signal(group_delay*2+1:end-group_delay*2);
```

The amplitude of the signal is normalized:

```
normalized_signal = ...
   downsampled_signal/ ...
      max([real(downsampled_signal)' imag(downsampled_signal)']);
```

The signal is quantized. The function used, does not recognize negative values, so these are wrapped to the values above the positive values:

```
q = quantizer('fixed', 'floor', 'saturate', [word_length word_length-1]);
real_part = quantize(q, real(normalized_signal))*2^(word_length-1);
```

```
for idx2 = 1:length(real_part)
   if sign(real_part(idx2)) == -1
      real_part(idx2) = 2^word_length+real_part(idx2);
   else
      real_part(idx2) = real_part(idx2);
   end
end
imag_part = quantize(q, imag(normalized_signal))*2^(word_length-1);
   for idx2 = 1:length(imag_part)
      if sign(imag_part(idx2)) == -1
         imag_part(idx2) = 2^word_length+imag_part(idx2);
      else
         imag_part(idx2) = imag_part(idx2);
      end
end
```

The real and imaginary parts are concatenated and passed to the function that creates an Intel HEX file:

```
mem_array = (real_part*2^word_length + imag_part);
Array2IntelHexSmpl(file_name, width_byte, depth, ...
   mem_array((idx2-1)*depth+1:idx2*depth));
```

E.2 Processing Test Vectors

The contents of the output file are read:

```
file_content = IntelHex2Array(file_name);
```

The data is converted to fixed point objects:

```
fi_data = fi(file_content, 0, word_length*2, 0);
```

The real and imaginary parts of the data is extracted, converted to the wanted numeric type (signed and 8 bit fraction length) and combined to form a complex number:

```
T = numerictype(1, word_length, word_length-1);
real_data = bitsliceget(fi_data, word_length*2, word_length+1);
real_data = reinterpretcast(real_data, T);
imag_data = bitsliceget(fi_data, word_length, 1);
imag_data = reinterpretcast(imag_data, T);
combined_data = double(real_data) + 1i * double(imag_data);
```

The data is demodulated while accounting for the latency in the system:

```
demodulated_stream = demodulate(qpsk_demodulator, compensated_signal);
[errors, ber] = biterr(...
   reference_bitstream_log{idx}(1:(end-symbols_latency*2)),...
   demodulated_stream((1+symbols_latency*2):end)...
   );
```

The confidence interval is calculated:

```
x = xor(reference_bitstream_log{idx}, demodulated_stream);
sample_variance = sqrt(sum((x-ber).^2)/(number_of_symbols(idx)*2-1));
confidence_interval = tinv(0.95,(number_of_symbols(idx)*2))...
   *sample_variance/sqrt(number_of_symbols(idx)*2);
```

Appendix F

Papers

The following papers were prepared for the course "Publishing Papers in the Peer-Reviewing System" conducted at Aalborg University 2010 by Prof. Jakob Stoustrup. The students of the course included several branches of study, and because of this, the papers were expected to be written in a more popular manner.

The paper "Maximum Likelihood Frequency Offset Compensation for Quadrature Phase Shift Keying Systems" reviews the algorithm designed for this project.

The paper "Modeling Quantization Noise in Finite Impulse Response Filters" describes the method for estimating the quantization noise in an FIR filter used in this project.

Maximum Likelihood Frequency Offset Compensation for Quadrature Phase Shift Keying Systems

Kasper Lund Jakobsen *

* *B.Sc.EE, Aalborg University, Danmark, Email: kalund@kom.aau.dk*

AbstractThis paper presents a Maximum Likelihood (ML) frequency error estimator for QPSK (Quadrature Phase Shift Keying) digital communication systems. The estimator is based on findings from Divsalar [1995] which showed the ML frequency error estimator for BPSK (Binary Phase Shift Keying) signals. Therefore the presented algorithm is development of earlier works. The estimator is based on averaging earlier input signals and thereby uses this information to estimate the present frequency error. Simulations have shown that the estimator improves the performance of the receiver and converges towards the performance of the QPSK communication system without frequency error.

1. INTRODUCTION

Wireless communication has, over the last few years, been subject to a huge development, due to a constant demand for more and more wireless devices. For example a mobile phone can now, through the wireless mobile phone networks, access the Internet, at almost as high speeds as computers through wired connections. The demand for higher speeds is always present which makes development of better communications systems an important research area.

When data is transferred over a wireless communications channel, it has to be different, with respect to frequency, data or time, from all other wireless communications taking place in the same geographical place. Further than that, it also has to have a certain range dictated by the specific application. For example there are many differences between the communication for mobile phone communications and satellite tv communications.

To distinguish between the different communications and to gain a large connection range, the different communications are transmitted within certain frequency bands [Haykin, 2001, p.183]. How this is done is not discussed here, but it is an effective way of separating the communication. A problem arises when the frequency band in which the data is transmitted changes during the transmission (see Figure 1). Normally the frequency band is predefined for the application, but it can change during the transmission e.g. due to the doppler effect [Raymond A. Serway, 2004, p.525]. This is the effect that happens when the transmitter and receiver moves relative to each other. These changes, though they should not happen seriously degrades the performance of the wireless communication system. This can be seen by Figure 1.

This article describes the extension of a previous described algorithm Divsalar [1995], which makes it suitable for modern wireless communication and more specifically the 3G communication system which is used in modern mobile phones.

To explain the algorithm a basic understanding of the communication is needed. When data is transmitted digitally either a “0” (zero) or a “1” (one) is transmitted. When these are transmitted they can be represented graphically as shown in Figure 2. In Figure 2 it is shown that one bit (a “1” or a “0”) is represented

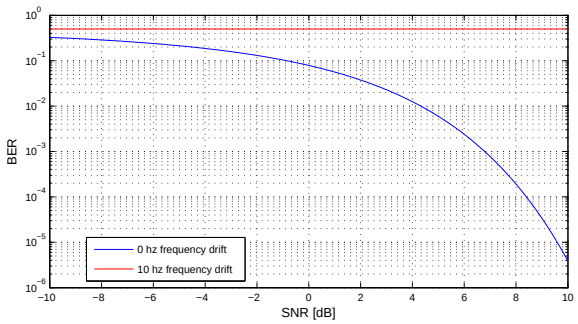


Figure 1. Theoretical performance of wireless transmission with and without frequency error Land and Fleury [2005]. The lower the BER (Bit Error Rate) is the better performance. As seen the performance is severely degraded when a frequency drift is present.

by a message point in a two-dimensional plane. This plot is known as a constellation diagram.

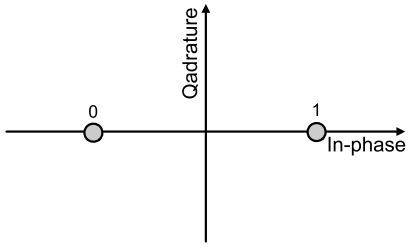


Figure 2. Constellation diagram with two message points, one per bit.

The algorithm, previously mentioned (Divsalar [1995]), which the algorithm presented in this paper is developed from is made for this constellation. It is, however, possible to put more points into this constellation by using the quadrature channel (see Figures 2 and 3), which makes the performance better. An example of this is shown in Figure 3.

When these are received they are corrupted by noise due to the wireless transmission. This influences the constellation diagram

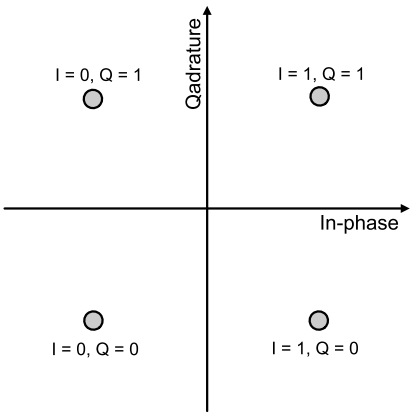


Figure 3. Constellation diagram with four message points. Notice that two bits are transmitted per message point, however, the points are closer to each other compared to the two point scenario.

at the receiver where the points that are received deviates from the original points. This is shown as the difference between Figure 3 and 4.

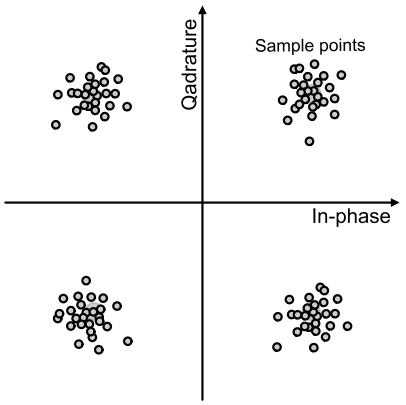


Figure 4. Constellation diagram at the receiver with four message points and with noise added. This figure shows how the noise corrupts the signals.

When the frequency changes during the transmission this means that the message points will go around origo (where the two axis intersect) in the constellation diagram as shown in Figure 5.

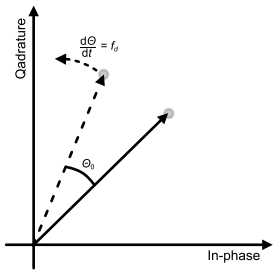


Figure 5. Part of the constellation diagram at the receiver with frequency error. The figure shows how frequency errors cause the message points to rotate and an instantaneous phase error (Θ_0).

With both low SNR (Signal to Noise Ratio) and frequency error it proves very difficult to use the communications system, which again can be seen by Figure 1.

2. METHOD

The main idea is; if a point is received then it is dependent on the bit or bits that were transmitted, but if the dependency can be removed from the signal, the frequency error for that signal can be found by averaging the phase error (difference in angle between actual sample and message point (see Figure 5)) for all the previous signals that are received. It has in Divsalar [1995] been shown that for two message points, the data dependency can be removed by taking the square of the input, and it has been shown that this is the theoretical best solution, when it is assumed that these frequency errors are equally likely within some interval.

The method is to use the same mathematical verification method as for two message points to find the best theoretical solution for four message points.

The derivations use complex numbers and statistics, and will therefore not be shown here.

To find the performance of this estimator simulations of its performance are conducted. The simulation results are the number of bit errors per transmitted bit for a given SNR value. These results are compared to the theoretical performance of QPSK. If the results converges towards the theoretical, it can be concluded that the esitmator can improve the performance of a QPSK system where frequency errors are present.

3. RESULTS

It is with the method derived by Divsalar [1995] shown that the data dependency with four message points in the constellation can be removed by squaring the input signal twice (taking the power of four of the input). The estimate of the frequency error is found by averaging the frequency error of as many previous inputs as possible. Mathematically the estimator can be shown by the following:

$$e^{j\hat{\theta}_{ML,k}} = \sqrt[4]{\frac{\sum_{i=1}^{N} r_{k-i}^4 \mathrm{sinc}(8 f_{\max} i T)}{\left|\sum_{i=1}^{N} r_{k-i}^4 \mathrm{sinc}(8 f_{\max} i T)\right|}} \quad (1)$$

This functionality can also be shown in a block diagram, this is shown in Figure 6.

The performance of this estimator is then simulated to test the performance of the derived estimator. The performance is shown in Figure 7 and 8 for two different numbers of inputs that are averaged over.

As seen the performance converges towards the theoretical performance for wireless transmission without frequency error. It is also important to notice that the performance of the algorithm does not necessarily increase with averaging over more samples, as this is dependend on the frequency error.

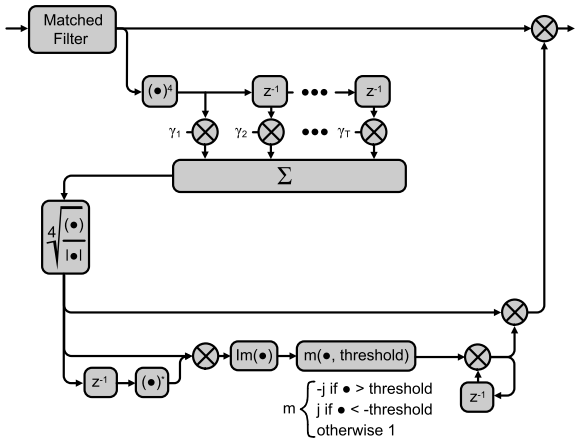


Figure 6. Block diagram of the developed algorithm. The matched filter should be disregarded here.

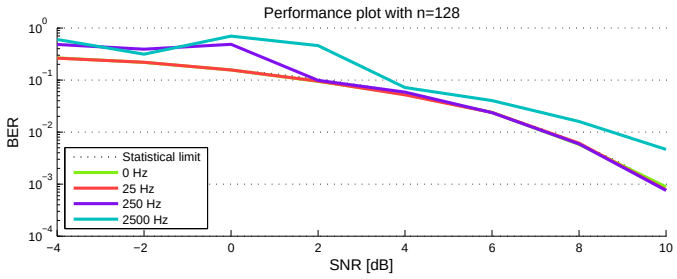


Figure 7. Theoretical performance of wireless transmission with and without frequency error averaged over 128 inputs. As seen the performance converges towards the statistical performance of a QPSK system without frequency error. Low BER means better performance.

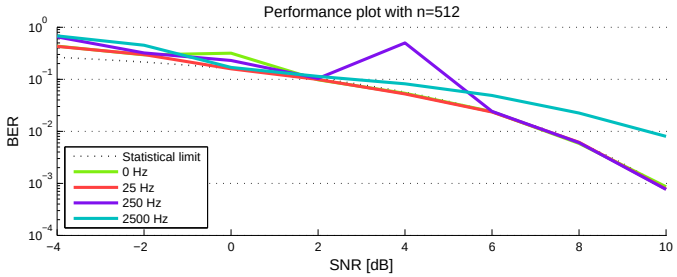


Figure 8. Theoretical performance of wireless transmission with and without frequency error averaged over 512 inputs. The performance still converges towards the statistical performance of a QPSK system without frequency error, for small frequency errors, but the performance degrades for large frequency errors.

4. DISCUSSION

Simulations have shown that the estimator performs as the math has shown it should. This means that frequency error can be removed from a QPSK signal. The optimal number of inputs that are averaged over is, to be determined for the specific application, where the algorithm is to employed. The initial simulations shows the tendency that; the more inputs that are averaged over the better performance, but an optimal number might exist for a specific application and with better knowledge about the expected frequency error.

Another research that could be conducted is to further develop the algorithm to work for even higher numbers of message points, which then makes the algorithm usable for even more communication systems.

If the algorithm is to be used in practice the number of inputs that are averaged over should be carefully analysed to reduce the computational complexity as much as possible while maintaining the performance needed for the algorithm given a specific application.

ACKNOWLEDGEMENTS

Thanks to Karl D. Hansen for coorporation in developing the algorithm, simulating it and creating the figures for this paper. Thanks to Yannick Le Moullec the supervisor of the project for ideas and advice. Finally a thanks to Jes Toft Kristensen (Rohde & Schwarz) for the project proposal and advice.

REFERENCES

Marvin K. Divsalar, Dariush; Simon. Pseudocoherent demodulation of dpsk radio signals. *NASA Tech Briefs*, 19(6):48–49, June 1995. ISSN 0145-319X.

Simon Haykin. *Communication Systems*. John Wiley & Sons, 4. edition, 2001. ISBN 0-471-17869-11.

Ingmar Land and Bernard Fleury. Digital modulation 1, 2005.

John W. Jewett Jr. Raymond A. Serway. *Physcis for Scientists and Engineers*. THOMSON BROOKS/COLE, 6 edition, 2004. ISBN 0-534-40844-3.

Modeling Quantization Noise in Finite Impulse Response Filters

Karl D. Hansen B.Sc.EE , Kasper L. Jakobsen B.Sc.EE

***Abstract*—Quantization noise can be regarded as a random process, and can thus be analyzed like any other random process. However, the prevailing method of estimating the quantization noise in a signal processing system is by simulation. This paper investigates the possibility of doing analytical computations of the resulting quantization noise in Finite Impulse Response (FIR) filters by estimating the noise by Additive White Gaussian Noise (AWGN). The possible impacts are: Lower design time and/or a more thorough design space exploration.**

***Index Terms*—FIR-filter, Characteristic Function, Quantization Noise**

I. Introduction

Signal processing algorithms have been developed and refined for over 50 years; they work well and are well understood. Usually these generic algorithms are designed using algebraic math. But computers do not understand algebraic math, they use numbers with finite resolution, this is called fixed point representation. Converting the, otherwise, mathematically perfect algorithms to the fixed point domain introduces imperfections. By increasing the resolution of the computations, these imperfections can be reduced. But doing this demand more resources of the computer and this is a problem in small battery-driven applications where cost and energy considerations dictate small and inexpensive devices.

A much used approach to convert these algorithms from algebraic math to fixed point math is to run simulations of the algorithm with a range of different resolutions, and compare them to a high resolution simulation to find the resulting errors from the different implementations. This is a relative slow process and thus often ends up as being considered a final step in the design process where the designer accepts the resulting error and moves on to implementation.

If models for the resulting quantization noise can be constructed, the final step of finding the quantization noise will be quicker and more exact than the simulation-based approach, and will thus be easier and more convenient to use as an integrated part of the design process. But trouble arises because the quantization process is not linear, i.e. the noise from one part of the algorithm cannot simply be added to the noise of another part, so more elaborate methods must be used.

In this project, a method for linearizing the quantization process is investigated. An analog to the well known digital Fourier transform of discretely sampled time signals is used on the discrete probability density function (PDF) resulting in a "band-limited" characteristic function, i.e. it is undefined for values over half the quantization interval (much like the Nyquist sampling theorem, see [1] and [2]). As the combination of PDFs in the value domain is done by convolution, they are multiplied in the characteristic domain:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \overset{\mathcal{F}}{\rightarrow} F(u)G(u)$$

This way the quantization process is linearized within the limiting band, allowing for model-based quantization noise estimates.

This paper exemplifies how the above method can be used to model the quantization noise of an FIR-filter. It further shows that the noise of a filter with a reasonable amount of taps can be estimated with an Additive White Gaussian Noise (AWGN) source. Using this estimate, the bit lengths of the individual filter taps are reduced in order to achieve the lowest possible amount of bits being processed at a specified noise tolerance level.

II. FIR-Filters

The results are based on simulations of a FIR-filter. A small introduction to the nature of this kind of filter is included here.

Filters are used in many applications to either remove unwanted signals or (essentially the same) enhance wanted signals. One of the best known applications of a filter is the bass filter on a regular hi-fi stereo set. Turning the knob one way reduces the bass, turning it the other way enhances it.

The FIR filter is a discrete-time filter. Its behavior is completely described by the signal at its output port when presented with a Kronecker Delta signal [1] at its input port. The output will be different from filter to filter depending on the filter coefficients, but the main

characteristic is that the output at some point will return to zero.

A small example of a FIR-filter is a moving average filter with three taps, i.e. three filter coefficients. Its behavior can be mathematically described via its impulse response:

$$h(n) = \frac{1}{3}\delta[n]+\frac{1}{3}\delta[n-1]+\frac{1}{3}\delta[n-2]$$

Where $\delta[n]$ is the Kronecker Delta function. The data flow graph of the filter is shown in Fig. 1 and a graphical representation of its impulse response in Fig. 2

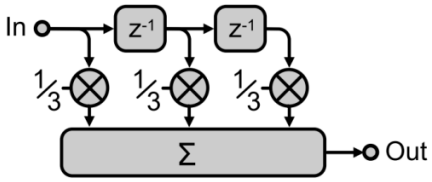


Fig. 1. The data flow graph for a three tap moving average filter. The input is delayed by one time unit in the boxes labeled z^{-1} and multiplied by one third in the multipliers marked with an X. In the end the processed samples are summed up in the box labeled Σ and output.

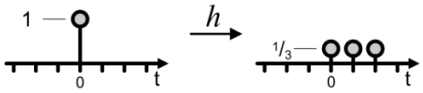


Fig. 2. The impulse response of the filter described in (1). On the input side a Kronecker Delta, on the output side three samples with the value $^1/_3$.

III. Quantization Noise

The trouble with the notation in (2) is that it is not necessarily using fixed point. The time is discretized but the values are not. This will lead to inaccuracies when implemented in a computer. This is called quantization noise, as it can be regarded as having a random distribution when the input is also randomly distributed, see [2] and [3] for elaboration. The filter in the example will quantize the values in 5 places. The first quantization takes place in the input. This is not directly visible in Fig. 1 but is necessary for the filter to "understand" the input. Next is the multiplication of the input with the coefficients. Each of the outputs of these multiplication will be quantized i.e. three more quantizations. In the end, the outputs from the taps are summed up. This leads to yet another quantization. In general we can say that given an FIR-filter with N taps, N+2 quantizations are done.

IV. Methods

As described by Widrow et al. in [3], the quantization process can be modeled as area sampling of the PDF of an input signal. The quantization noise is then modeled as a uniformly distributed additive noise. In order to linearize the process Widrow Fourier transforms the noise into the characteristic domain. The characteristic function (CF), i.e. the fourier transformed function, of a uniform PDF is a sinc function, defined by:

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

If several quantizations are done, then the CFs of each quantization stage can be multiplied together in order to obtain the resulting quantization noise, this is equivalent to convolving the PDFs of the quantizations in the value domain. Multiplying several sinc functions approximates a Gaussian distributed CF, which is also a Gaussian distribution in the value domain. Fig. 3 shows how the convolution of three equal uniform distributions approximates a Gaussian distribution.

Widrow's research leads us to this: Because of the combined noise being normally distributed, we can model the quantization noise as one single AWGN source instead of several separate quantization noises. As the PDF of AWGN is indeed a normal Gaussian distribution.

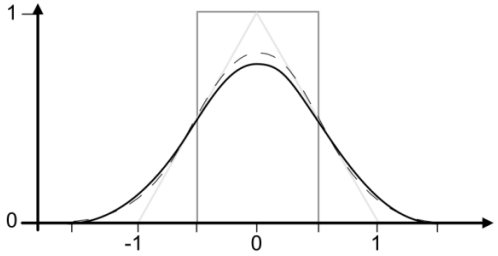


Fig. 3. The convolution two and three uniformly distributed PDFs. The original PDF is the dark gray graph. Convolved once with itself the light gray triangular PDF results. Convolved two times, i.e. three uniform PDFs convolved together gives the black PDF. The dashed line is a Normal distribution with a variance equal to the sum of the three uniformly distributed PDFs.

The approximate Gaussian distribution in the value domain can be estimated without transforming back and forth to and from the characteristic domain by summing up the variances of the individual quantization noises, thus creating a simple model of the quantization noise. This, however, requires that several quantizations are done, i.e. over 10. Fig. 3 shows an approximation by summing variances of only three uniformly distributed PDFs which rather well fits the actual convolution of the PDFs.

V. Results

An FIR-filter with 512 taps has served as an example for the examination of this method. This is a part of a frequency error detector depicted in Fig. 4. The entire system is described in depth in [4]. Just like in the moving average example. The noises of each of the taps

in the filter are being added by the summing of the taps, and because of the great number of taps, a AWGN source will adequately model the quantization noise of the filter. The approximate AWGN source of the noise is defined only by the variance of the noise. This variance has been obtained by simply summing the variances of the different noises in the taps alone. Compared to the large number of taps, the input and summing quantizations are negligible.

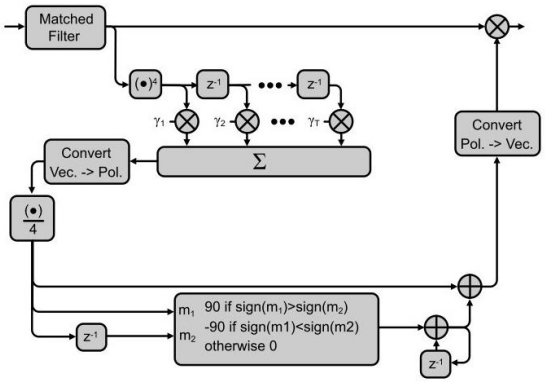


Fig. 4. The example filter is a part of a frequency error detector being developed for 3G telephony. The FIR-filter part sits in the middle, having 512 taps in the implementation proposed in [4].

The AWGN approximation allows for the simulation of the system, using several different fixed point representations. In [4] it shows that a 9 bit representation is adequate to meet the requirements.

VI. Conclusion

The simulation using AWGN sources are tractable as AWGN channels are widely used and already implemented in many simulation tools. But furthermore, this approach enables actual analytical evaluation of the quantization noise. This is very interesting in systems with many bits in its fixed point representation, as many bits results in a small quantization error. When the error is very small, a large number of simulation samples is needed in order to produce statistically significant results. And when the simulations are needed to run at e.g. everything between 24 and 32, it might take quite a while to get the results.

The AWGN modeling lets the engineer compare the results after just 9 computations (24—32 bit configurations). It even lets him compare every combination of bit lengths for the individual taps at a fraction of the time it takes to simulate the first results.

In the end this method might speed up design time and allow the designers to do a more extensive and thorough design space exploration.

VII. References

[1] E. Kreyszig, *Advanced Engineering Mathematics, 9th ed.* Wiley, November 2005.

[2] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Prentice Hall, January 1999.

[3] B. Widrow, I. Kollár, M. Liu, "Statistical theory of quantization," IEEE Trans. on instrumentation and measurement, vol. 45, no. 2, pp. 353—361, Apr. 1996.

[4] K. D. Hansen, K. L. Jakobsen, "Frequency error detector for 3G basestations using feed-forward technique," unpublished.

Bibliography

[1] J. Costas. Synchronous communications. *Communications Systems, IRE Transactions on*, 5(1):99–105, March 1957.

[2] Dariush Divsalar and Marvin K. Simon. Pseudo-coherent demodulation for mobile satellite systems. In *Proceedings of the Third International Mobile Satellite Conference (IMSC 1993)*, pages 491–496. Jet Propulsion Lab., California Inst. of Tech., Pasadena, June 1993.

[3] Marvin K. Divsalar, Dariush; Simon. Pseudocoherent demodulation of dpsk radio signals. *NASA Tech Briefs*, 19(6):48–49, June 1995.

[4] R. Hamila and M. Renfors. New maximum likelihood based frequency estimator for digital receivers. In *Wireless Communications and Networking Conference, 1999. WCNC. 1999 IEEE*, pages 206–210 vol.1, 1999.

[5] Anders Riis Jensen, Niels Terp Kjeldgaard Jørgensen, and Kim Laugesen. Non-data aided carrier offset compensation for SDR implementation, 2008.

[6] Ingmar Land and Bernard Fleury. Digital modulation 1, 2005.

[7] Heinrich Meyr, Marc Moeneclaey, and Stefan Fechtel. *Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[8] Steven P. Nicoloso. An investigation of carrier recovery techniques for PSK modulated signals in CDMA and multipath mobile environments. Master's thesis, Virginia Polytechnic Institute and State University, June 1997.

[9] Keshab K. Parhi. *VLSI Digital Signal Processing Systems - Design and Implementation*. John Wiley & Sons, 1999.

[10] John G. Proakis. *Digital communications*. McGraw-Hill, New York, 2001.

[11] 3^{rd} Generation Partnership Project. *TS 25.213 V7.5.0*, May 2008.

[12] 3^{rd} Generation Partnership Project. *TS 25.101 - User Equipment (UE) radio transmission and reception (FDD)*, September 2009.

[13] John W. Jewett Jr. Raymond A. Serway. *Physcis for Scientists and Engineers*. THOMSON BROOKS/COLE, 6 edition, 2004.

[14] A. Th. Schwarzbacher, A. Brasching, Th. H. Wahl, P.A. Comiskey, and J. B. Foley. Optimisation and implementation of the arctan function for the power domain. In *Electronic Circuits and Systems Conference*, pages 33–36, 1999.

[15] A. Viterbi. Nonlinear estimation of PSK-modulated carrier phase with application to burst digital transmission. *Information Theory, IEEE Transactions on*, 29(4):543–551, Jul 1983.

[16] Jack E. Volder. The CORDIC trigonometric computing technique. *Electronic Computers, IEEE Transactions on*, EC-8(3):330–334, Sept. 1959.

[17] Bernard Widrow, István Kollár, and Ming-Chang Liu. Statistical theory of quantization. *Instrumentation and Measurement, IEEE Transactions on*, 45(2):353–361, Apr. 1996.

[18] Zaihe Yu, Y.Q. Shi, and Wei Su. A blind carrier frequency estimation algorithm for digitally modulated signals. In *Military Communications Conference, 2004. MILCOM 2004. IEEE*, volume 1, pages 48–53 Vol. 1, Oct.-3 Nov. 2004.