UPPSALA
UNIVERSITET

UPTEC F15 037
Examensarbete 15 hp
Juni 2015

Hardware design and implementation of the Schmidl-Cox synchronization algorithm for an OFDM transceiver

Peter Morris

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Hardware design and implementation of the Schmidl-Cox synchronization algorithm for an OFDM transceiver

Peter Morris

The subject of this document is the VHDL firmware implementation of a coarse synchronization method for a 4G/5G transceiver. The method of choice is the Schmidl-Cox synchronization algorithm that is applied to the OFDM transmission standard as preparation for later conversion to the FBMC method. This algorithm is first developed and validated in a MATLAB floating point environment. After this a thorough analysis step is conducted to devise a fixed point implementation of negligible perfor- mance loss. Thereafter a main contribution of this work comes through the proposal of a low-complexity hardware architecture that efficiently implements this fixed point Schmidl-Cox algorithm. This architecture is described in VHDL and validated through extensive simulations after integration with the transceiver model. Simulation results and logic syn- thesis targeting a Zynq 7020 FPGA board illustrate the efficiency of the proposed implementation.

Handledare: Amer Baghdadi
Ämnesgranskare: Leif Gustafsson
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F15 037

Contents

Acknowledgments

First of all I would like to thank my supervisors at Telecom Bretagne, Amer Baghdadi and Jeremy Nadal. The knowledge they shared along with the aid in the execution of this project they provided was irreplaceable.

I would also like to thank the exchange coordinators Ulrika Jaresund and Anders Berglund for helping me to discover, arrange, and organize this project abroad.

In addition I would like to thank my subject reviewer Leif Gustafsson for his advice and input on the paper, presentation, and project aims.

Finally, I would like to thank my family for their continued support throughtout my education and the occasional spellcheck session.

Abbreviations

FBMC - Filter Bank MultiCarrier
OFDM - Orthogonal Frequency Division Multiplexing
VHDL - Very High Speed Integrated Circuit Hardware Description Language
FPGA - Field Programmable Gate Array
4G/5G - Collective term for 4th/5th Generation Mobile Networks
LTE - Long term evolution
METIS - Mobile and Wireless communications enablers for the Twenty-Twenty Information Society
SNR - Signal-to-Noise Ratio
CFO - Carrier Frequency Offset
ISI - Inter-Symbol Interference
AWGN - Additive White Gaussian Noise

1 Introduction

The Internet has been a transformative force of human civilization over the past few decades, with its importance only growing each day. Access to the internet has changed from usage of obscure academic departments to nearly every single person having a device in their pocket which is connected to the internet at all times.

With the advancement of communications technology, how the internet is used has also changed, from sending the occasional email to now streaming cinema-quality films on a 5-inch screen. As this evolution continues, there will be a need for faster, more efficient communication methods.

This document discusses the design, testing and implementation of the wireless synchronization firmware for an Orthogonal Frequency Division Multiplexing (OFDM) transceiver. The synchronization method of choice is the Schmidl-Cox algorithm(see Section 4.1). This synchronization method will detect when an incoming signals arrives, and correct any accrued frequency offsets so the rest receiver can view the signal as intended. The current implementation of this project is geared towards OFDM, to enable future designs for the next generation of internet communication methods Filter Bank MultiCarrier (FBMC) transmission method.

The work described in this paper is part of a project that aims to test and implement FBMC methods of wireless transmission, or 5G as per the EU project Mobile and wireless communications Enablers for the Twenty-twenty Information Society (METIS). FBMC is the proposed next generation of the current OFDM transmission methods found in 4G/LTE communications, and will allow for greater spectral efficiency and higher robustness against adverse signal conditions. The current FBMC transceiver prototype discussed in this paper was recently demonstrated at the 2015 Mobile World Congress in Barcelona.

2 Background

2.1 OFDM and Wireless Communications

Orthogonal Frequency Division Multiplexing (OFDM) is a method to wirelessly transmit data over multiple orthogonal carrier frequencies. Due to this orthogonality, sub-channels do not interfere with each other, alleviating many problems found in traditional Frequency Division Multiplexing methods. OFDM is the current transmission method of choice for both LTE and the more recent Wi-Fi standards due to its high spectral efficiency, robustness, and relative simplicity of hardware implementation.[1]

In OFDM, each carrier has its own amplitude and phase that holds the binary QAM information. The signal is broken up into individual "symbols" that are the OFDM data carriers. 15 kHz is the frequency spacing between each OFDM carrier in LTE.

Figure 1 shows how a transmitted OFDM signal is built up. It consists of two separate portions, the symbol and the cyclic prefix. The symbol is the binary data meant to be transmitted after having been transformed to a Quadrature Amplitude Modulation map (Section 2.1.2) and then converted from a signal in the time domain to a signal in the frequency domain by a Fourier Transform.

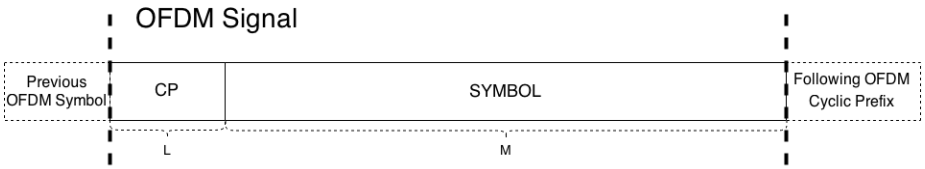

Figure 1: OFDM Signal Buildup.

The cyclic prefix is a copy of the last L samples of the M symbol length and prepends this copy to the beginning of the symbol. The length of the cyclic prefix L is generally on the order of 5-10% of M, depending on desired robustness properties. In this project, the length M of the symbol is 512, while the length L of the cyclic prefix is 51. The cyclic prefix is discussed further in Section 2.1.1.

The OFDM signals created in this manner are then transmitted, the beginning of the cyclic prefix of the second immediately following the end of the symbol data of the first.

2.1.1 Cyclic Prefix

The multipath propagation that the cyclic prefix is intended to guard against occurs due to the wireless nature of the OFDM transmissions. Multipath refers to the transmitted signal being reflected off of objects such as building or mountains, and finding several paths to the same receiver, an example shown in Figure 2.

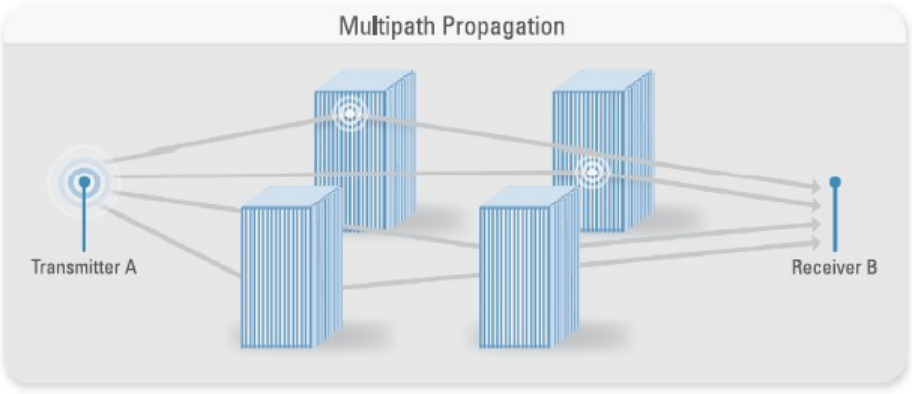

Figure 2: Multipath Propagation Illustration. [2]

Should a receiver receive both the signal from the transmitter, and from an object reflecting this signal, it's as if the receiver is receiving the same signal at several different points in time. This can lead to either constructive or destructive interference as well as phase shifting, collectively referred to as "intersymbol interference" as these effects cause one symbol to disturb another symbol. [1]

The cyclic prefix is used as a guard interval between two separate OFDM symbols, the buildup of which can be viewed in Figure 3. The reason for using

a guard interval is to combat intersymbol interference in the case of multipath propagation described later in this section.

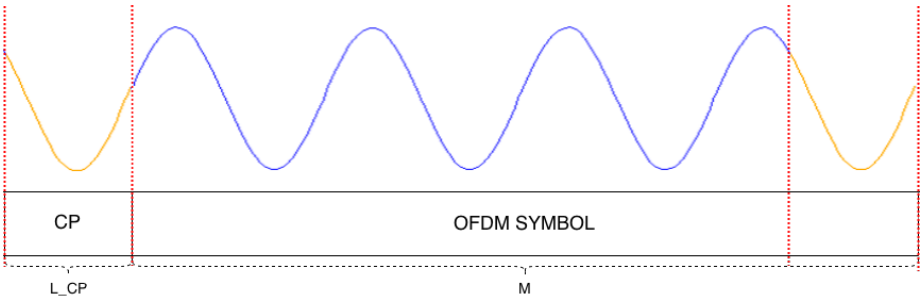

Figure 3: Depiction of Cyclic Prefix for one carrier.

The cyclic prefix is a copy of the final portion of the following OFDM symbol, thus making the beginning and ending of this signal identical. The length of this cyclic prefix should include a higher number of samples than the presumed worst-case multipath propagation delay in order to allow the receiver to recover the entire OFDM symbol intact and avoid intersymbol interference. The cost of introducing a cyclic prefix is that the spectral efficiency is reduced, therefore, while a longer cyclic prefix improves robustness, it should be kept as short as possible to increase data transmission rates.

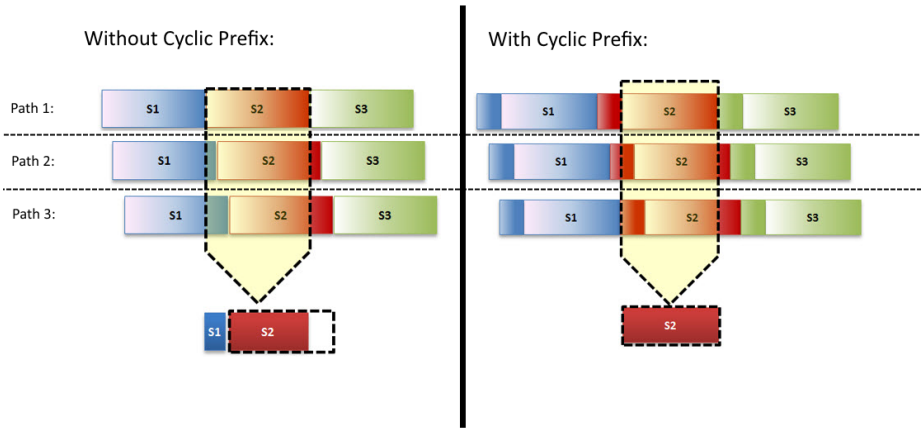

Figure 4: Comparison of Multipath Signals with and without Cyclic Prefixes. [3]

The difference between using and not using a cyclic prefix is depicted in Figure 4. The illustration on the left shows how the signal would be received without a cyclic prefix, on the right with a cyclic prefix. One can see therefore that through the use of the cyclic prefix the receiver is able to correctly receive all of the transmitted information without intersymbol interference despite the multipath propagation due to the overlap of the same signal. However without a cyclic prefix there is no opportunity to cleanly sample only the desired signal.

This figure also aptly illustrates why the cyclic prefix length should be greater than the expected multipath delay.

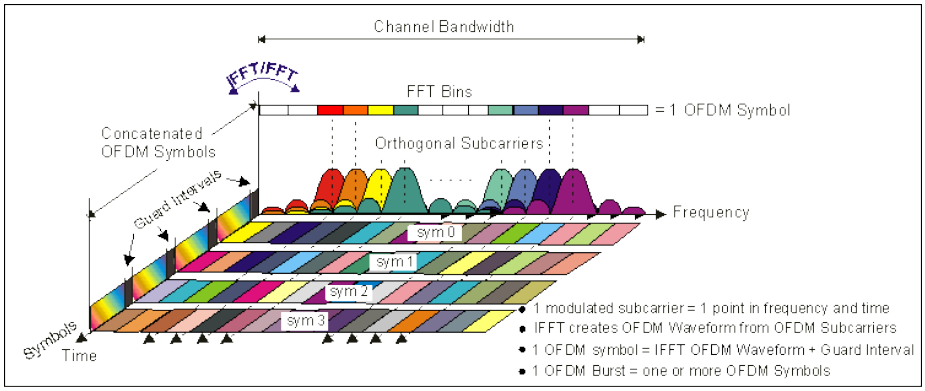

Figure 5: Representation of OFDM in Frequency and Time. [14]

Figure 5 shows a representation of what an OFDM broadcast looks like in the frequency and time domain. The frequency portion has a number of subcarriers, the subcarriers with the high peaks are the active carriers. Also note the guard intervals in the time domain where the cyclic prefix would be inserted. The amalgamation of these signals constitute the transmitted data.

2.1.2 QAM Mapping

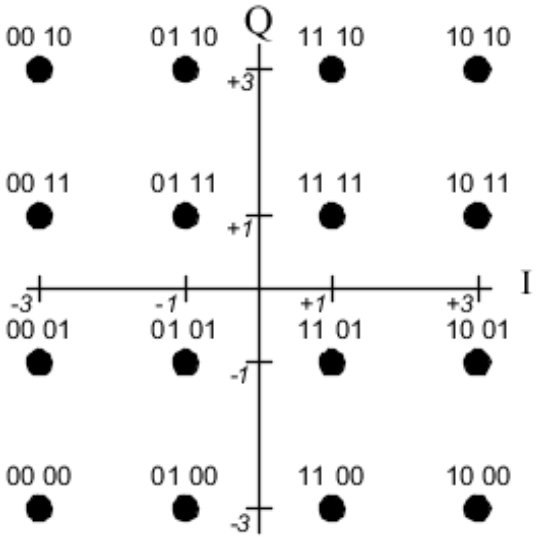

Figure 6: QAM Map example. [4]

Quadrature Amplitude Modulation, or QAM mapping, is a method by which several bits of data can be transmitted using only the transmitted I and Q signals. Figure 6 shows how this can be achieved. The transmitted signal is made

up of the two I and Q signals, which are orthogonal to each other. Variation of these two separate signals points to different parts of the "constellation map" in Figure 6. For example, I='1' and Q ='3' would lead to the "1110" value in the constellation. Setting I='3' would then point to the "1010" signal, etc. While this figure can display up to 4-bits of information with a given IQ sample, the limit of QAM is theoretically arbitrarily large and instead limited only by signal noise and quantization constraints.

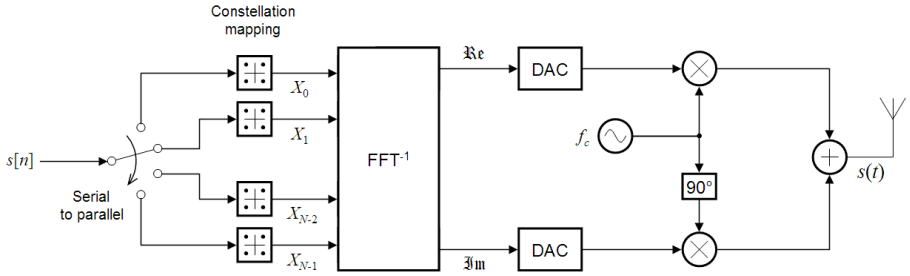

Figure 7: QAM Transmission overview. [6]

Figure 7 shows how the binary input is first converted into the corresponding QAM value. This is done by "chunking" the input data to get discover the appropriate QAM symbol, then processing this so that the outgoing I and Q signals are transmitted as this QAM symbol. Each input/subcarrier of the FFT holds a QAM modulation. This figure also shows the two signals in the intermediate stages being at 90° from each other, thus orthogonal. QAM mapping is very widely used across all types of wireless communications, including for the OFDM method used in this project. [7]

2.1.3 CFO and Rotation

Carrier Frequency Offset (CFO) is an issue in wireless communication systems that occurs when there is a discrepancy in carrier frequency between the transmitter and the receiver due to a doppler shift, doppler spread, refraction, local oscillator frequency offsets between transmitter and receiver, etc. [8]

Figure 8 shows QAM maps for two received signals as an example of the rotation caused by CFO. The QAM map on the left has compensated for the 3% CFO relative to the subcarrier space (which is 15 kHZ in LTE), while the QAM map on the right has not compensated for this CFO. The uncompensated signal can be seen to rotate, with one discrete QAM mapping eventually passing into another as a result of intersymbol interference caused by this CFO.

In addition to this intersymbol interference, there is an intercarrier interference as can be seen in how the spread of the discrete QAM placings increase. This intercarrier interference is also caused by the significant amount of CFO as the OFDM subcarriers leak into one another.

To correct this, the CFO contained in the incoming signals must be estimated by the synchronization unit. After this estimation is performed, the incoming signals can then be rotated to their correct positions, compensating for this CFO and rendering their appearance similar to the left QAM map.

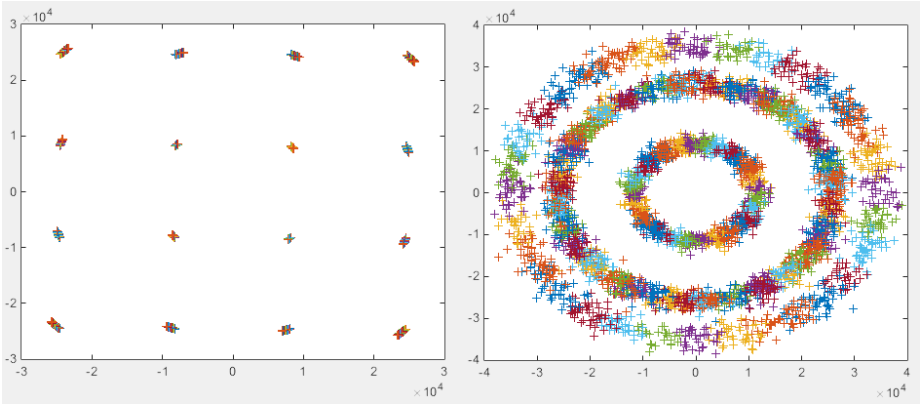

Figure 8: QAM Map of Adjusted Vs. Unadjusted CFO of 3 percent.

2.1.4 FBMC

While the implementation of the Synchronization Unit described in this paper is only performed using OFDM methods, the ultimate goal of this project to implement a Filter Bank MultiCarrier (FBMC) transceiver prototype. FBMC is a transmission method that is a proposed successor to OFDM, as championed by METIS.

One of the problems found in OFDM is that of out of band leakage seen in the sinc function side lobes of Figure 9. This stems from the discontinuous nature of OFDM. Figure 1 shows an example of an OFDM signal, and should two of those be sent back to back, there would be a discontinuous state between the end of the first signal and the start of the second.

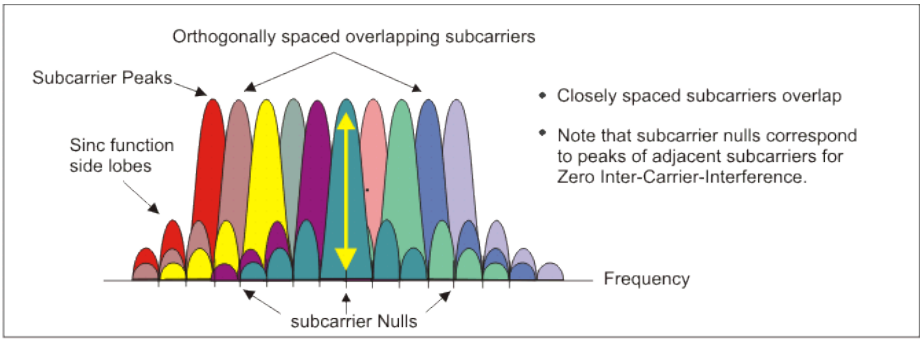

Figure 9: Depiction of OFDM leakage. [5]

FBMC aims to improve this by filtering each subcarrier independently to limit these secondary lobes and improve the time and frequency localization. [9] This can be implemented through use of a PolyPhase Network block at the output of the IFFT. Figure 10 shows how this network block behaves. First, to solve the discontinuity the transmitted signals are filtered, or shaped, as in Figure 10. However, this would lead to other issues, such as the edges of the symbol being dampened. To remedy this, the symbol is repeated twice in a

row. In that manner, the entire symbol is visible to the receiver at some point. This however, would cause a very large loss in bandwidth, as each symbol is repeated twice. Therefore, two different symbols are added together and sent out concurrently, all still illustrated in Figure 10.

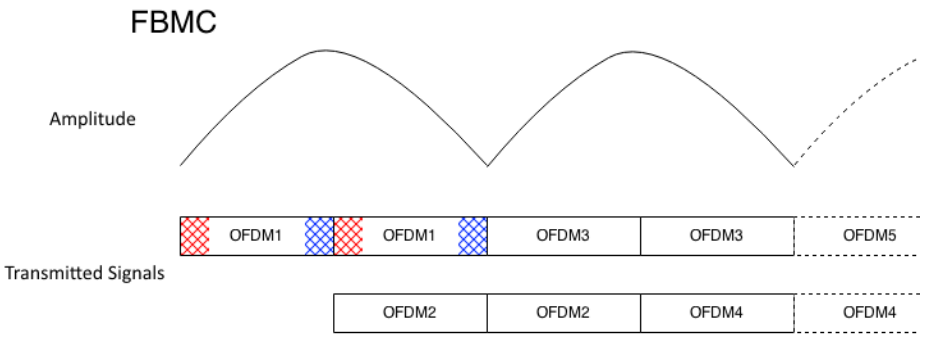

Figure 10: FBMC Transmission Overview

Now the signal is continuous and has a high bandwidth, but the received symbols would have major intersymbol and intercarrier interference, having transmitted together. This is solved by using the Offset-QAM scheme, which enables shifting of the orthogonality in real domain by separating I and Q, applying an offset of half the symbol length to Q, and finally alternating real and imaginary values between each subcarrier. At the output of the demodulation on the receiving end, the transmitted data is recovered by taking the real part of the received signals, while all the interference is only apparent in the imaginary part, and thus discarded.

Implementation of FBMC allows for higher spectral efficiency, increased robustness, and easier sampling methods at the cost of requiring special filtration techniques. Research into FBMC is ongoing, and this particular project has been presented at the 2015 Mobile World Congress, and aims to be complete in 2017.

2.2 Transceiver Component Overview

Figure 11 is a simplified component overview of the OFDM transceiver implemented prior to the commencement of this project, and the basis of the simulations performed in this paper. The Sync Unit, represented by the red block in this picture, is the component whose design is described in this paper. The idea behind the transceiver design is to emulate a signal being sent wirelessly from transmitter to receiver, while the actual components are all on the same device. [10]

The signal meant to be transmitted starts as the "binary in" signal and traverses clockwise through this design, eventually exiting as the "binary out"signal. The QAM mapper (Section 2.1.2) starts this process, taking the "binary in", "select QAM", and "select carriers" signals in order to create the QAM symbols that will be sent (as opposed to the binary). The "select QAM" signals the order of the QAM mapping (which is generally 16 bits in this project) and the "select carriers" signal indicates which carriers the outgoing signal will be

broadcast on. The pilot insertion component immediately following the QAM mapper periodically inserts a pilot signal into the transmission data used later for channel estimation.

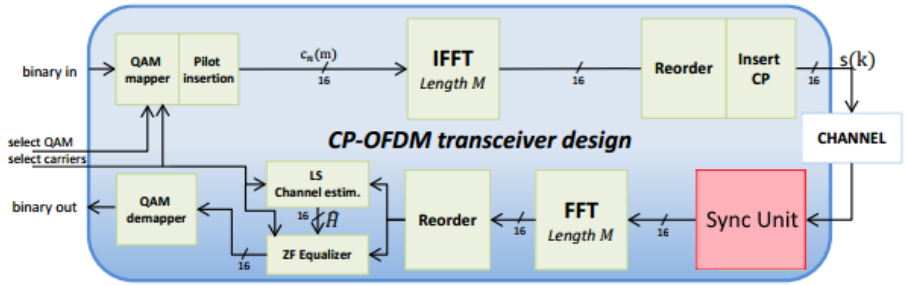

Figure 11: Component overview of the OFDM transceiver.[10]

This signal is then sent to the Inverse Fast Fourier Transform (IFFT) block, and is converted from a signal in the time domain to the frequency domain. As the implementation of the IFFT block leaves the output signal in disorder so these signals must then be reordered. Simultaneously, the cyclic prefix (Section 2.1.1) is appended to the signal to be transmitted. This is performed simultaneously in order to share a memory, as the Cyclic Prefix simply consists of the final portion of the OFDM symbol. At this point, the signal is fully prepared to be transmitted.

The transmission is emulated through the "channel" block, which simulates elements of a wireless broadcast. Additive white gaussian noise (AWGN) produced in MATLAB, multi-path effect, CFO, etc. can all be introduced in the channel block for testing purposes.

This signal is then passed to the receiver portion of the transceiver, starting with the "Sync Unit", the design of which is the subject of this paper. The Sync Unit needs to recognize when the transmitted signal arrives, identify at which time the OFDM data symbols begin, correct any introduced CFO, and remove the Cyclic Prefixes before then passing the data further along to the FFT.

The FFT block converts the signal back from the frequency into the time domain. The resulting signal also has to be reordered as in the end of the transmitter.

From there the signal is passed to the LS (Least-Square) channel estimation block. Channel estimation is performed to estimate the channel's frequency response, and use this estimations to further calibrate the received signals.

The "ZF" ("zero-forced") equalizer performs channel equalization on the received symbols in order to prepare them for the QAM demapper. The QAM demapper then takes the received symbols and converts them to a stream of binary output data. Should everything function correctly, the binary output should exactly match the binary input.

2.2.1 Current Hardware State

The hardware used for the transceiver consists of two Xilinx "Zynq" Field Programmable Gate Array (FPGA) boards. Despite the data being transmitted

wirelessly, these two boards are wired together to properly ensure synchronization and emulated CFO between the two separate devices.

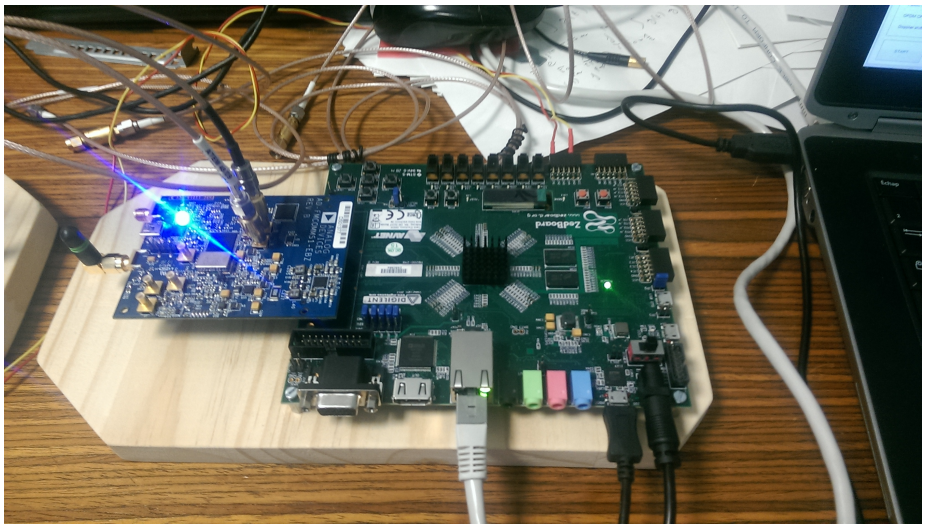

Figure 12: FPGA Board of Transceiver Prototype.

After the completion of the synchronization unit described in the project, the cables between these two boards will no longer be needed, as the synchronization unit will be able to gather all required information simply from the transmitted signal.

3 Project Aims

The final goal of this project is to successfully implement a time and frequency synchronization method in the VHDL firmware of an FPGA board allowing for two separate FPGA boards to successfully communicate wirelessly. The synchronization method of choice in this paper is the Schmidl-Cox algorithm for OFDM systems.

Firstly, the algorithm should be tested in a software environment, which in this project is MATLAB. There is a version of the transceiver without synchronization which this project is built upon, with the portions required for synchronizing added. After the transceiver is seen to successfully synchronize in MATLAB, a fixed point version is created, modifying the original floating point implementation in order to make the MATLAB simulations closer to that of the hardware implementation.

After the fixed point implementation of this project works correctly, the hardware design can begin. The hardware design indicates which components will be needed, how they will be connected and the role of each of these components. This design details down to how many multipliers, adders, FIFOs, memories, etc. are needed for the design. Once this design is complete, the MATLAB fixed point implementation is updated to match as closely as possible to the new hardware design, and tested again.

From this point, the implementation in the hardware description language VHDL can begin. Each of the required components are desribed in VHDL code and tested in Mentor Graphics's VHDL simulation tool ModelSim. After the VHDL implementation is seen to be correct, this entire sync component is then integrated into the rest of the transceiver's VHDL implementation, and further tested. Once all of this is correct and coherent with the results of MATLAB, the project can then be tested on the actual hardware.

The VHDL component required for the Schmidl-Cox synchronization unit requires:

- Calculation Unit
  The calculation unit takes in the incoming signal data, upon which it constantly computes the Schmidl-Cox algorithm.

- Detection Unit
  The detection unit takes in the Schmidl-Cox output, and constantly checks if this signal can be used for synchronization. When it detects the sync point, it notifies the retention unit.

- Retention Unit
  The retention unit stores both the incoming signal data, and the Schmidl-Cox P values for as long as they may be needed. After being notified by the detection unit that the sync point has been found, it passes the appropriate P values to the estimation unit, and the incoming signal values to the rotation unit after the estimation unit has calculated the estimation.

- Estimation Unit
  The estimation unit calculates $arctan(x/y)$, or the angle of an complex number, for the two incoming Schmidl-Cox P values from the retention unit corresponding to the sync point. The result of this calculation is sent to the rotation unit as the magnitude of rotation required.

- Rotation Unit
  The rotation unit receives the incoming signal data from the retention unit, along with the amount required to rotate this incoming signal data from the estimation unit. It outputs the correctly rotated input signals to the rest of the transceiver.

This is all in addition to the minor modifications needed to the transmitter portion of the project required to send the correct Schmidl-Cox preamble that will be used for the detection.

After all of this is completed, the project is then meant to be converted to FBMC and a similar procedure will be performed to ensure its correctness.

4 Theory

4.1 Schmidl-Cox Synchronization Algorithm

For correct interpretation of wirelessly transmitted data it is necessary to for the receiving device to be able to both know where the transmitted signal begin and to correct any frequency adjustments that may have happened during transmission. In other words, the time and frequency must be synchronized between the transmitter and the receiver. This project achieves this synchronization through implementation of the Schmidl-Cox algorithm.

The Schmidl-Cox algorithm is a time and frequency synchronization algorithm designed by Dr. T.M. Schmidl and Dr. D.C. Cox for use in OFDM transmissions. This method of synchronization works by prepending a specially generated Schmidl-Cox preamble to the data that is to be transmitted, which is used by the receiver to extract the information necessary for synchronization. This synchronization is performed as necessary, as expected channel conditions dictate how often the synchronization will need to be updated. [11]

The Schmidl-Cox preamble is generated by creating a pseudo-random complex Gaussian noise on the odd frequencies, while having zero amplitude on even frequencies. This construction of the pilot gives the transmitted preamble certain properties that can later be exploited in calculations. By processing this preamble on the receiving end, the necessary signal components for both time and frequency synchronization can be interpolated.

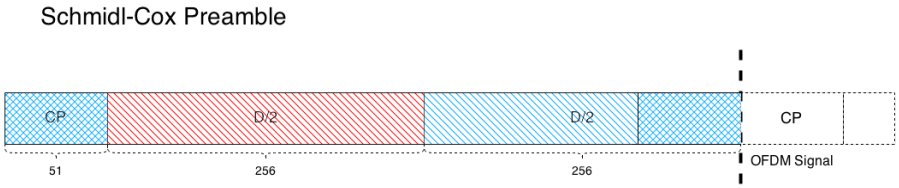

Figure 13: Appearance of Schmidl-Cox preamble.

Figure 13 shows the makeup of a Schmidl-Cox preamble. Note that the OFDM symbol section of this preamble is divided into two halves, marked in red and blue respectively. These parts are the inverse of each other due to the absence of amplitude on even frequencies before the signal is sent through the FFT. Also note that the Cyclic Prefix (CP) is identical to the last portion of the preamble (both these signal portions marked in darker blue), as per regular OFDM transmissions.

Through continuous computation of the Schmidl-Cox algorithm on incoming signals, the receiving device attempts to locate the beginning of any transmitted data.

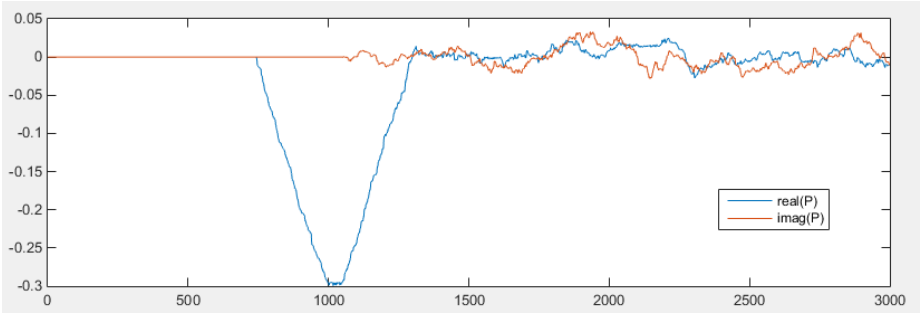

Figure 14: Example of the complex P signal generated by the Schmidl-Cox algorithm. (CFO = 0).

$$P(d) = \sum_{m=0}^{L-1} (r^*_{d+m} r_{d+m+L}) \tag{1}$$

$$P(d+1) = P(d) + r^*_{d+L} r_{d+2L} - r^*_d r_{d+L} \tag{2}$$

Figure 1 shows a typical result of the calculation of the Schmidl-Cox autocorrelative P value and the formula used for its calculation along with its iterative version. The autocorrelation is performed at L, or half of the OFDM symbol length, which for the Schmidl-Cox preamble is the inverse value. The iterative version of the autocorrelative formula (2) shows that the iterative addition and subtraction performed at the tail end of the Schmidl-Cox preamble symbol are equivalent, as is shown in Figure 13. This equivalency during the iterative summation causes the flat portion, seen in the graph of P in Figure 14. Something else to notice in Figure 14 is that the CFO estimation is obtained by finding the angle of the complex P value at the point of synchronization. In this case, as the CFO is zero, the imaginary part of P is also zero at the point of synchronization (the approximate center of the flat portion of the real component of P.).

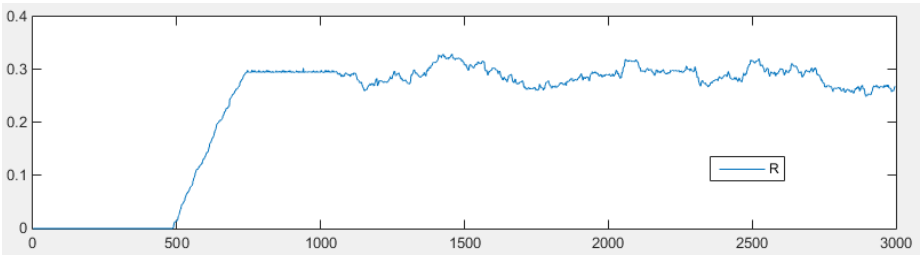

Figure 15: Example of the R signal generated by the Schmidl-Cox algorithm. (CFO = 0).

$$R(d) = \sum_{m=0}^{L-1} |r_{d+m+L}|^2 \tag{3}$$

$$R(d+1) = R(d) + |r_{d+L}|^2 - |r_d|^2 \tag{4}$$

The R value of the Schmidl-Cox algorithm, can be viewed as the power portion of the signal, as it estimates the variance. A typical result of the R signal is displayed in Figure 15, also showing the formula used for its calculation (3) along with its iterative version (4). This estimation of variance rises at the beginning only to somewhat stabilize over the rest of the function. Note again the stable flat portion, which endures for the duration of the Schmidl-Cox preamble symbol, again due to the periodic equivalency contained within the symbol.

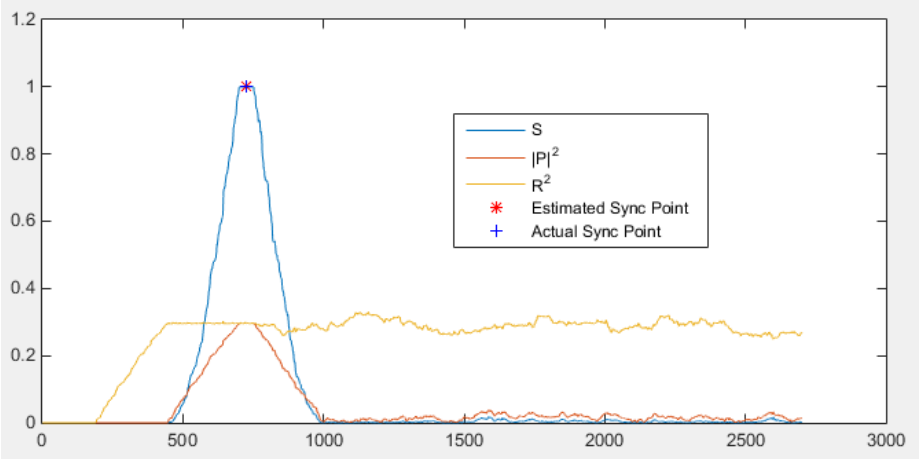

Figure 16: Example of the S signal generated by the Schmidl-Cox algorithm.

$$S(d) = \frac{|P(d)|^2}{(R(d))^2} \tag{5}$$

Finally, the desired signal S is obtained through the calculation of the quotient of the squared P and R values. An example of the S signal along with the numerator $|P|^2$ and denominator R^2 is seen in Figure 16, along with the relevant formula (5).

As the quotient of the P and R values, the S value displays a similar shape to the numerator P, albeit with a more pronounced peak. The flat part at the top of the S signal in 16 is an indicator of the end of the Schmidl-Cox preamble's OFDM symbol, the calculation exposing the equivalency of the Cyclic Prefix and the final portion of the symbol, thus causing the flat portion to be the same length as the Cyclic prefix. A demonstration of this alignment is shown in Figure 17.

By knowing where the Schmidl-Cox preamble ends, it is now possible to conclude where the next part of the signal begins. Detection of the ending of this portion of the S signal is covered in Section 4.2.

4.2 Detection Methods

For this project two separate methods for locating the peak of the Schmidl-Cox S signal are tested, each attempting to find the center of the peak of the S value allowing for relative localization of the beginning of the next data part of the

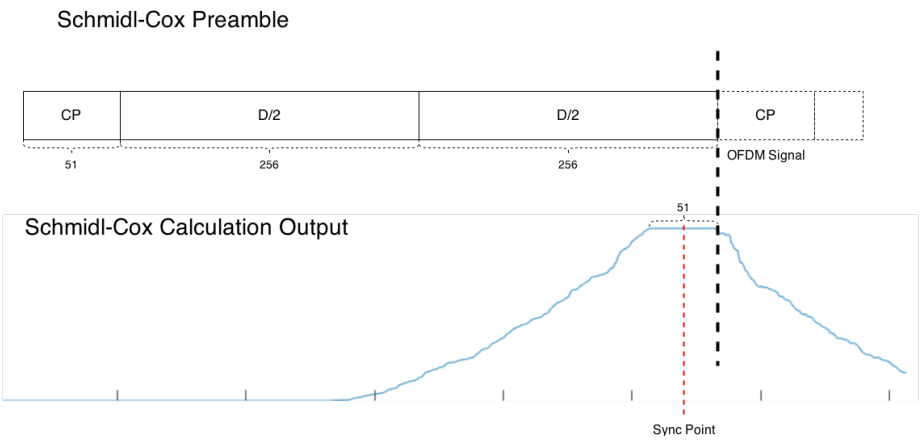

Figure 17: Schmidl-Cox Preamble.

incoming signal. As displayed in Figure 17, the S signal peak has the same length as the cyclic prefix, therefore the beginning of the next portion of data is half the cyclic prefix length samples later.

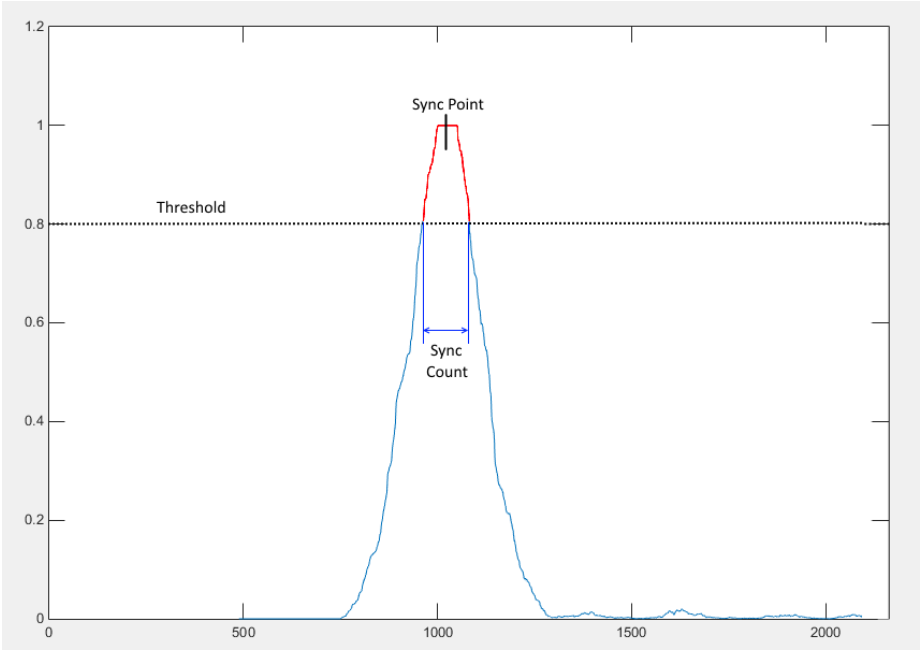

Figure 18: Threshold Detection Example.

4.2.1 Threshold Method

The threshold detection method works by recording the points at which the S signal crosses a predefined threshold value, first above then below. By averaging these two points, the center of the peak is hopefully found, as illustrated by

Figure 18.

Other minor considerations include insurance this is indeed the intended peak of the S signal and not just a temporary spike. This is done by ensuring that the duration the threshold has been crossed exceeds the cyclic prefix length, otherwise waiting for the next crossing of the threshold to issue a sync point.

4.2.2 Window Minimum Method

The other method of sync point detection instead relies upon observing a moving window of samples after the threshold value has been crossed. The length of the window is equivalent to the cyclic prefix length, and is centered on the current sample point. By storing the minimum value contained in each of these windows, the beginning of the flat portion of the S signal can be discovered.

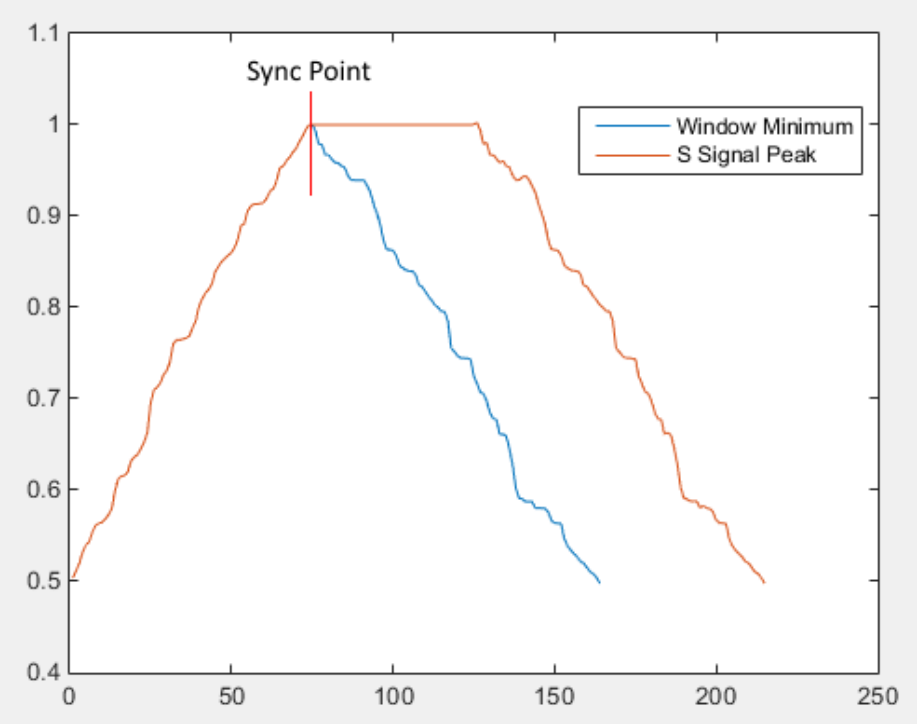

Figure 19: Window Detection Example.

Due to the window length being equivalent to the cyclic prefix length (which is also equivalent to the flat portion of the Schmidl-Cox S signal), when the beginning of the window is set on the exact sync point, the entire window should rest on the flat portion. An illustration of this method is shown in Figure 19. Simply put, this method takes the maximum value of each recorded window minimum value between the S signal threshold points.

4.2.3 Alternative Synchronization Methods

- Minn-Bhargava:

The Minn-Bhargava [12] is not entirely unlike the Schmidl-Cox algorithm insofar as it attaches a self-correlative preamble to the OFDM symbol, however if uses 4 sections of repeating data, instead of two. Its results are illustrated in Figure 20.

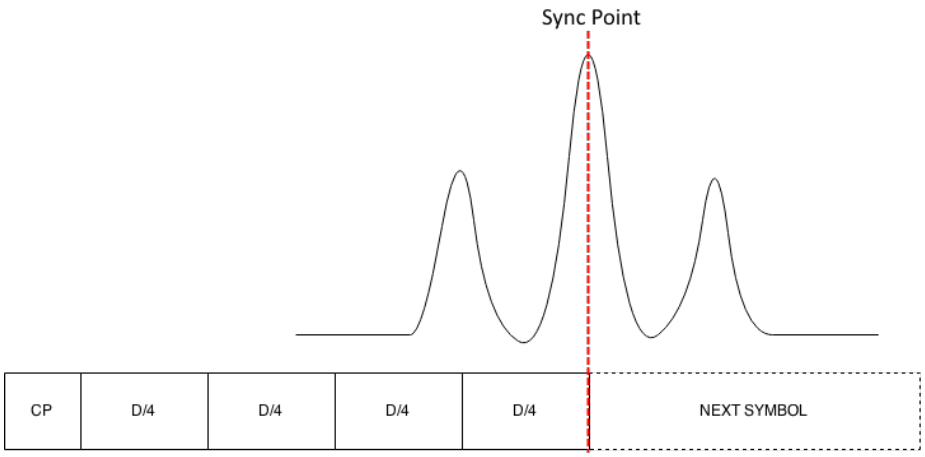

Figure 20: Minn-Bhargava Synchronization Example.

The Schmidl-Cox algorithm is used in place of this as it simplifies some sections of detection. Also, the Minn-Bhargava is more subject to false detection, given the extra two peaks, it is also less robust against multipath propagation. The positive side of Minn-Bhargava method is that finding the exact point of synchronization is more likely due to the extreme spike, but this property was not deemed necessary for this project, as the estimated sync point being within a few samples of the exact sync point is acceptable.

- Costas Loop:

  Costas Loop [13] is a analogue technique that uses a Phase-Locked Loop to synchronize the CFO of an incoming signal. While reasonably simple and effective, it is not used for time synchronization, which the Schmidl-Cox algorithm performs in addition to the CFO estimation. Future revisions of this project will look into using and adapting a Costas loop for improving accuracy, or for sake of comparison to the Schmidl-Cox algorithm.

4.3 CORDIC Theory

The CORDIC (an abbreviation of COordinate Rotation DIgital Computer) algorithm is used to greatly simplify hardware complexity for various mathematical functions. [14] In this project, CORDIC implementations are used in both the Estimation Unit and the Rotation Unit which use the CORDIC algorithm in Vectoring mode and Rotation mode respectively.

4.3.1 CORDIC in Rotation Mode

The CORDIC algorithm in vectoring mode is used for rotating an input vector, consisting of x and y, by an angle θ. This is done using an iterative approach that removes the need for a hardware multiplier, and instead functions not entirely dissimilar to a binary search. Figure 22 shows the first few iterations of the CORDIC algorithm in vectoring mode.

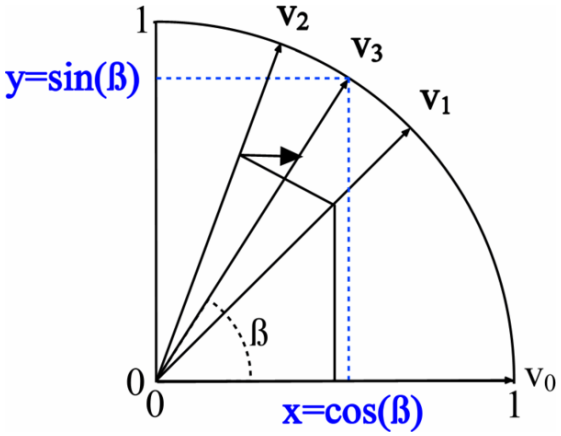

Figure 21: CORDIC Rotation Example.[14]

Figure 21 shows the CORDIC algorithm at v_0, and attempting to locate the vector v_3. It first rotates $\frac{\pi}{4}$ radians counterclockwise to v_1. At point v_1, the algorithm discovers that the angle between it and v_3 is greater than zero, and therefore performs another rotation, this time of $\frac{\pi}{8}$ radians counterclockwise, to v_2. Now the difference between the angle of v_2 and v_3 is less than zero, so the CORDIC algorithm rotates the vector and angle of $\frac{\pi}{8}$ in the clockwise direction. At this point, the difference in angle is zero, and the algorithm has been correctly rotated by an angle β.

Mathematically, the algorithm begins at zero radians. This is represented by the coordinates in (6).

$$v_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6}$$

Each iteration of the CORDIC algorithm rotates the input values x and y by an angle α as shown in equations (7) and (8).

$$v_n = R_n v_{n-1} \tag{7}$$

$$R_n = \begin{bmatrix} \cos(\alpha_n) & -\sin(\alpha_n) \\ \sin(\alpha_n) & \cos(\alpha_n) \end{bmatrix} \tag{8}$$

Using trigonometric identities, the sinus and cosinus terms in (8) can be rewritten as in (9) and (10).

$$\cos(\alpha) = \frac{1}{\sqrt{1+\tan^2(\alpha)}} \tag{9}$$

$$\sin(\alpha) = \frac{\tan(\alpha)}{\sqrt{1+\tan^2(\alpha)}} \tag{10}$$

Thus, the iterative equation from (7) may be rewritten as in (11).

$$v_n = \frac{1}{\sqrt{1+\tan^2(\alpha)}} \begin{bmatrix} 1 & -\tan(\alpha_n) \\ \tan(\alpha_n) & 1 \end{bmatrix} \begin{bmatrix} x_{n-1} \\ y_{n-1} \end{bmatrix} \tag{11}$$

The core of the CORDIC rotation algorithm is seen in (11), as the rotational part of each iteration is only dependent on $tan(\alpha_n)$ values. By forcing these rotational values to be powers of two, shown in (12), the hardware complexity is greatly diminished. (As a hardware shift operation is far simpler than a standard multiplication).

$$\tan\alpha = 2^{-n}, n = 1, 2, ...N \tag{12}$$

The iterative equation then becomes as in (13), where σ_n indicates whether the next rotation is to be performed clockwise or counter clockwise. The K_n value in (14) is a scaling factor denoted by (15) which has a limit and can be precalculated, as in (15).

$$v_n = K_n \begin{bmatrix} 1 & -\sigma_n 2^{-n} \\ \sigma_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} x_{n-1} \\ y_{n-1} \end{bmatrix} \tag{13}$$

$$K_n = \frac{1}{\sqrt{1+2^{-2n}}} \tag{14}$$

$$K = \lim_{n\to\infty} K(n) \approx 0.607253... \tag{15}$$

All of this results in finding the desired angle β through an iteration of (16), where σ_n is ± 1 indicating the direction of rotation, and α indicating the angle of rotation is seen in (17).

$$\beta_n = \beta_{n-1} - \sigma_n \alpha_n \tag{16}$$

$$\alpha_n = \arctan(2^{-n}) \tag{17}$$

The values of (17) are stored on a lookup table, to avoid calculation. This size of this lookup table is only required be $n \times n$, where n is the number of iterations performed.

Therefore, through use of the CORDIC algorithm the rotation of a vector that would generally require several multiplications and a large sinus/cosinus lookup table can instead be performed using only addition and subtraction, bitshifts, and a modestly sized lookup table, all of which relatively simple in hardware.

This rotation mode of CORDIC is used by the rotation unit, which rotates the input IQ data by the input Carrier Frequency Offset estimation angle.

4.3.2 CORDIC in Vectoring Mode

The CORDIC algorithm in vectoring mode can be viewed quite similarly to the rotation mode. In the rotation mode, an input vector is rotated by an input angle, receiving the rotated vector as output. In vectoring mode, an input vector is instead rotated to zero radians, giving the amplitude and original angle of the vector as output.

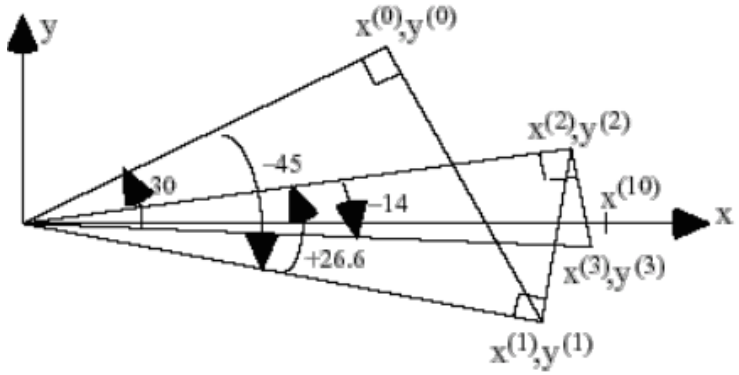

Figure 22: CORDIC Vectoring Example.[14]

Figure 22 shows how the CORDIC algorithm behaves in vectoring mode. The vector is always rotated towards $y = 0$, taking smaller steps with each iterations.

This mode is used in the estimation unit to discover the angle between the complex P values by calculating $arctan(y/x)$. This angle corresponds to the Carrier Frequency Offset, and is used by the rotation unit to rotate the input IQ values.

5 Software Implementation

5.1 Schmidl-Cox Floating Point

Initial implementation of the Schmidl-Cox algorithm was performed with MATLAB in a floating-point environment. Before introduction of the synchronization method, there was a working testbed for the OFDM transceiver.

The first step in implementing the synchronization is the appending of the Schmidl-Cox preamble to the input signal before it is passed on from the transmitter to the simulated channel. This preamble was generated through creation of a 2-wide array, indicative of the real and imaginary portions of the preamble, with all even rows initialized to zero, and all other values having a randomized value of +/- 1. In addition to this, all of the non-active carrier indexes were set to zero. (The code required for this seen in Figure 23)

```
schmidlcox_sign = (-1).^(randi([0,1],signal_length,2));

schmidlcox_sign(1:2:length(schmidlcox_sign)) = 0;

sc_preamb(AC) = schmidlcox_sign(AC,1) + 1i*schmidlcox_sign(AC,2);
```

Figure 23: Generation of Schmidl-Cox preamble.

After the generation of this preamble array is complete, it is then appended to the beginning of the OFDM signal before it is sent to the QAM mapper and the rest of the transmission portion of the design.

Calculation of the various Schmidl-Cox values is derived from the preamble after it has passed through the simulated channel. (These values seen in Section 4.1) These are calculated as in Figure 24. Important to note the simplicity of the calculations in this floating point version, as it becomes more complex when implemented in hardware.

```
for I = 1:N-M
    P(I) = sum(conj(IQ(I:I+L)).*IQ(I+L:I+2*L));
    R(I) = sum(abs(IQ(I+L:I+2*L)).^2);
end

S = (abs(P).^2)./(R.^2);
```

Figure 24: Calculation of Schmidl-Cox algorithm.

Both the threshold and the minimum window method are implemented to locate the synchronization point of the calculated S value from the Schmidl-Cox algorithm. These detection processes are implemented as described in 4.2.

```
P_angle = angle(P(sync_point));
CFO_estimation = 1 - P_angle / pi;
```

Figure 25: Calculation of CFO Estimation.

Once this sync point has been located, the angle of the complex P values at this index is found. This angle is the estimation of the CFO for this data signal. The Schmidl-Cox preamble is then removed from the signal, and the rest of the signal is rotated by the CFO estimation amount. The synchronization is thus complete, entailing that this method would allow for wireless communications to work.

5.2 Conversion to Fixed Point

In order for the MATLAB code to be run on the hardware, it must be converted from floating point variables to fixed point. Figure 26 illustrates the difference between the interpretation of fixed point and floating point values.

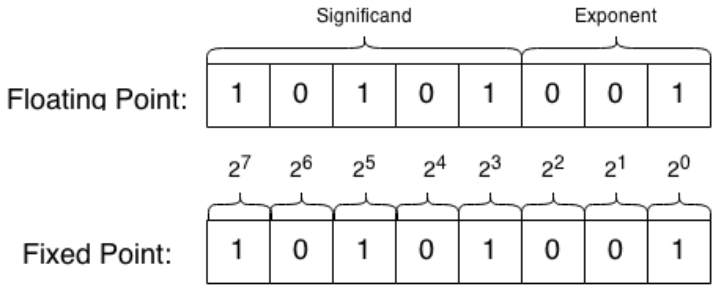

Figure 26: Difference Between Floating And Fixed Point

The floating point value consists of a section of bits reserved for the significand (which represents the significant bits) with another section for the exponent (representing the magnitude). With this implementation, high accuracy across a high range of values can be achieved, so it is often used for scientific tests and calculations. However it is very complex to implement in hardware and therefore the MATLAB floating point values must be converted to fixed point.

In the fixed point interpretation, each bit corresponds to a magnitude of 2, and summation of these bits corresponds to the stored value. This gives perfect accuracy, as each bit corresponds to an exact integer value, but this gives a range of only 2^L values that can be expressed. (For comparison, a 64-bit floating point variable has a range greater than 10^{100})

This small range (16-bits in this project) is not large enough to store all significant values for each stage of the calculations. Therefore the values are resized, taking only the most significant bits and losing the rest of the data.

In order to transform the floating point Schmidl-Cox algorithm implementation to its fixed point version, it is necessary to first convert all of the calculations into their fixed point versions. This implies more than simply changing the type of variable. When adjusting to 16 signed bits, one is limited to only a $\tilde{}3.2 \cdot 10^4$ values of differentiation, or less than a 10^5 order difference between smallest and largest possible values. This limitation is unsuitable for the Schmidl-Cox floating point implementation, as it deals with values both larger and smaller than can be represented by a 16-bit integer. By simply limiting all of the intermediate results of the Schmidl-Cox algorithm to 16-bits, overflows would occur at several junctions, rendering the calculations useless. Therefore, after each multiplication, the results must be resized into a reasonable representation.

5.3 Schmidl-Cox Fixed Point

The floating point implementation of the Schmidl-Cox algorithm as described in Section 5.1 is relatively straight forward. The issue lies in managing to take this implementation, and convert it into something that can be performed directly in hardware.

The limitation imposed by the hardware include generating sequential calculations (i.e. while MATLAB can perform an addition and a multiplication in one line, the hardware has to first perform the addition, then the multiplication.) and instead of using Floating Point variables, the stored values have to be in Fixed Point (Which happens to be 16-bit throughout this project).

Breaking up the one line of MATLAB code used to calculated P in Figure 24 into a series of sequential operations that can be translated into hardware is seen in Figure 27. This method ensures that only one operation is performed on a single piece of data at a time (the ".*" can be viewed as atomic processes performed over time).

```
Pmult1_re = real(IQ(1:N-M+L)) .* real(IQ(1+L:N-M+2*L));
Pmult2_re = -imag(IQ(1:N-M+L)) .* imag(IQ(1+L:N-M+2*L));

Pmult1_im = -imag(IQ(1:N-M+L)) .* real(IQ(1+L:N-M+2*L));
Pmult2_im = imag(IQ(1+L:N-M+2*L)) .* real(IQ(1:N-M+L));

rePM_tab = Pmult1_re - Pmult2_re;
imPM_tab = Pmult1_im + Pmult2_im;

for I = 1:N-M-1
    rePnode(I) = rePM_tab(I+L) - rePM_tab(I);
    reP(I+1) = rePnode(I) + reP(I);

    imPnode(I) = imPM_tab(I+L) - imPM_tab(I);
    imP(I+1) = imPnode(I) + imP(I);
end

P_A2 = abs(reP) .* abs(reP);
P_B2 = abs(imP) .* abs(imP);

P_F = P_A2 + P_B2;
```

Figure 27: Calculation method for P value in Fixed point.

Figure 27 is closer to how the calculation works in hardware, but does not satisfy the fixed point values requirement. A multiplication between two binary values results in a value with the sum of the bit-order of the two input values. Therefore, after each multiplication, the results need to be resized back to an acceptable bit-order. (In this project, the multiplication is of two 16-bit values resulting in a 32-bit value needing to be resized back to a 16-bit value.)

Figure 28 depicts the reason for needing to resize these binary numbers. As the total bits in the output are the sum of the total input bits, and the design is mean to accommodate only the same amount of bits as the inputs of this multiplication, the output has to be cut down to the same size as the input. If one wants to be entirely sure that no overflow can occur, one can simply take the top portion of the bits, as indicated by A in Figure 28. However, by

ignoring all of the upper bits that remain 0 in the product, one can instead take position B, which contains more overall information now shown in A. The goal is to maximize the relevant information contained in all of the multiplications.

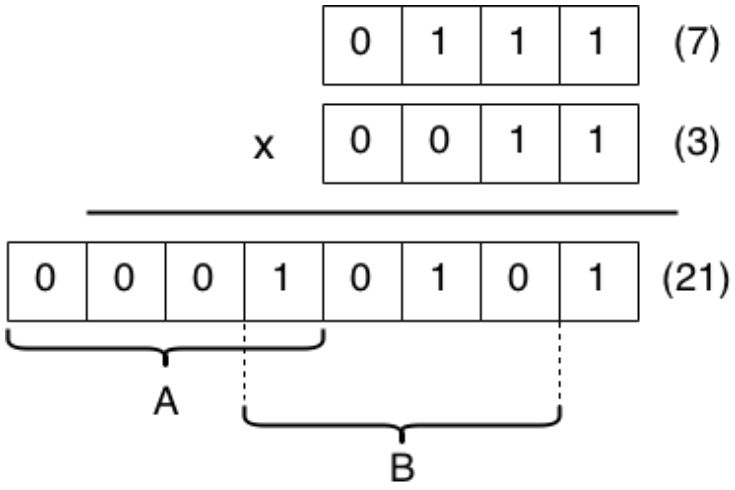

Figure 28: Multiplication of two binary numbers.

To that end, each calculation of the MATLAB code was performed in fixed point, and the highest shift of the register not resulting in an overflow recorded. By having the proper shift values after each of these operations, the quantization affords the highest amount of precision possible.

Finally, as the MATLAB fixed point implementation is meant to emulate the hardware as closely as possible for testing purposes, it was kept up to date to behave logically identical to the hardware design.

6 Software Results

Prior to beginning on the hardware implementation, the MATLAB results were investigated. This was to ensure that the concepts were working in theory and needed only to be properly implemented on hardware. This also allowed for verification of results produced while implementing the hardware.

6.1 Schmidl-Cox Fixed vs. Floating Point

Figure 29 shows the S result calculated by the Schmidl-Cox algorithm, with the normalized fixed and floating point implementations overlain. The Y-axis is an arbitrary value simply depicting the relative strength of the Schmidl-Cox output while the X-axis shows the time in terms of received samples.

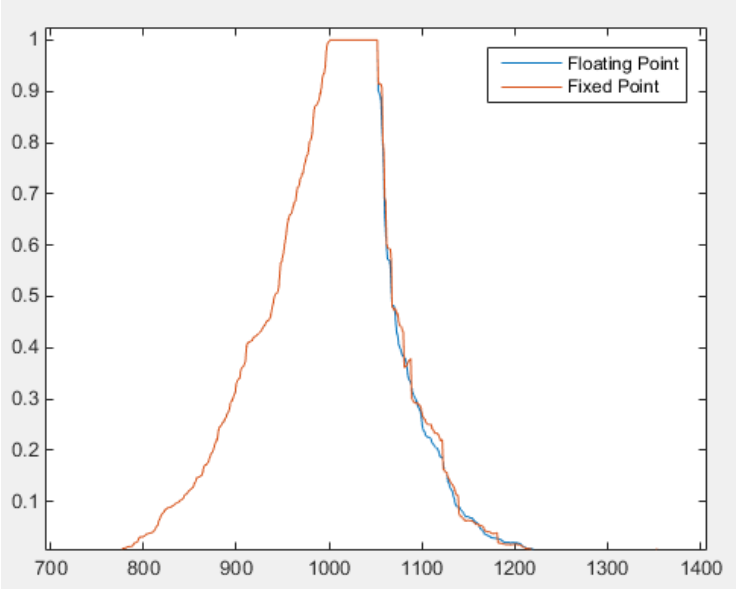

Figure 29: Schmidl Cox S value calculated in Fixed and Floating Point.

The results in Figure 29 are firstly very similar to the desired output of the Schmidl-Cox algorithm, suggesting that the algorithm is correctly implemented. In addition, the fixed and floating point versions are quite similar to each other, implying that the MATLAB conversion from fixed to floating point has been performed correctly.

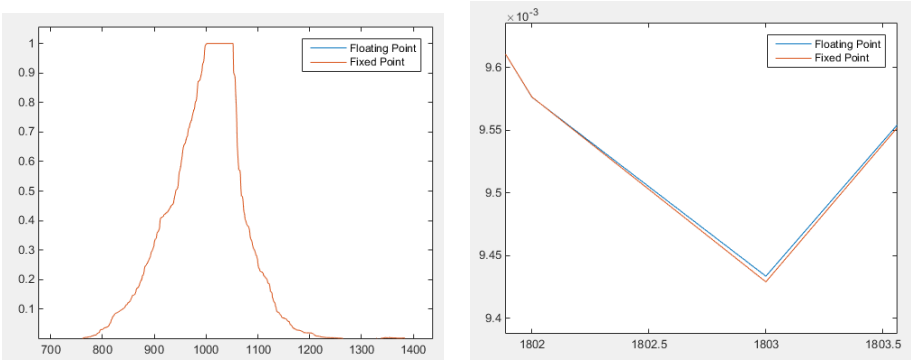

Figure 30: Comparison of Fixed/Floating Point with floating-point divider at extreme zoom levels.

The left side of Figure 30 shows the same graph as in Figure 29 albeit using the floating point division instead of the fixed point. This nearly eliminates the error found in Figure 29, with the zoomed in portion of shown on the right side of Figure 30 showing that there are actually very minor quantization errors.

Therefore the weak link in this operation is the hardware divider, and through increasing the accuracy of said divider, the accuracy of the Schmidl-Cox calculation is increased.

6.2 Sync Performance At Low SNR

Testing of the Schmidl-Cox synchronization at low Signal-to-Noise Ratios (SNR) reveal the robustness of the algorithm. It is important that the Fixed Point closely matches Floating point. The left part of Figure 31 show a comparison of Sync Point estimation error between floating and fixed point implementations at low SNR. This is using the Threshold Detection method shown in Section 4.2.1

A good sign is that the mean error remains very close between both the Fixed and Floating point implementations. The Fixed point however shows that at an SNR of 4 or lower the quantization fails to work properly, and the simulation becomes plagued by register overflows.

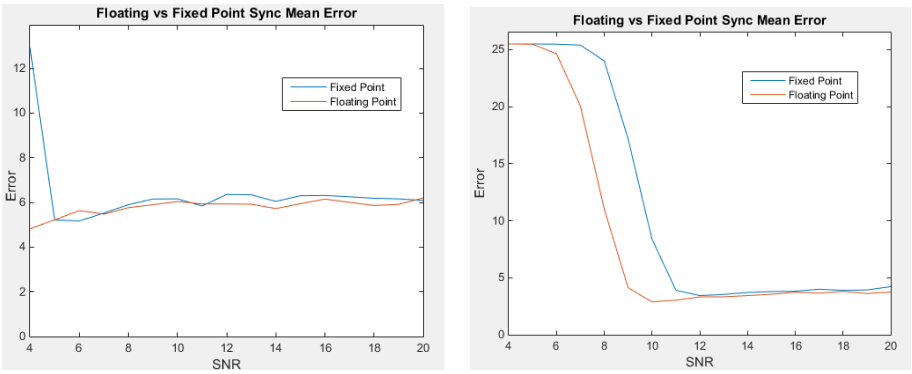

Figure 31: Comparison of Sync Mean Error at threshold values of 0.5 vs 0.8.

The right part of Figure 31 again shows the sync point estimation error of the Threshold detect at low SNR. However, the threshold value used for sync point detection in the left part has a threshold value 50% of the maximum value, while the right part now shows the same information with a 80% threshold value. While this does have a marginal increase of accuracy at high SNR values, it becomes a liability at low values, so instead of the threshold detection working all the way down to an SNR of 5, the 80% threshold causes major errors starting from an SNR of only 10.

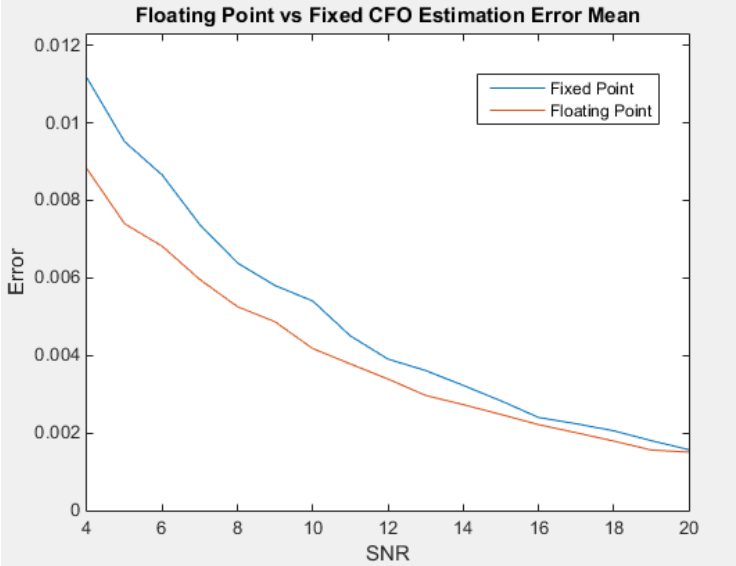

Figure 32: Comparison of Floating/Fixed Point CFO estimation error at low SNR.

Figure 32 shows the CFO estimation error between the fixed and floating point implementations. The fixed implementation is again quite similar to the floating point version, albeit somewhat higher as the quantization errors show themselves more readily in the division used for the CFO estimation.

6.3 Detection Methods

Both the threshold (Section 4.2.1) and window minimum (Section 4.2.2) were implemented and tested in the floating point version for MATLAB. Figure 33 shows the mean distance of the Sync point estimation from the actual sync point of each detection method over 5000 different simulations at varying SNRs. The SNRs showed in this figure are quite low, in order to test the robustness of these methods in adverse conditions.

The window minimum detection soundly outperforms the threshold method, while both of them have increasing mean error distances as the SNRs decrease, which is to be expected.

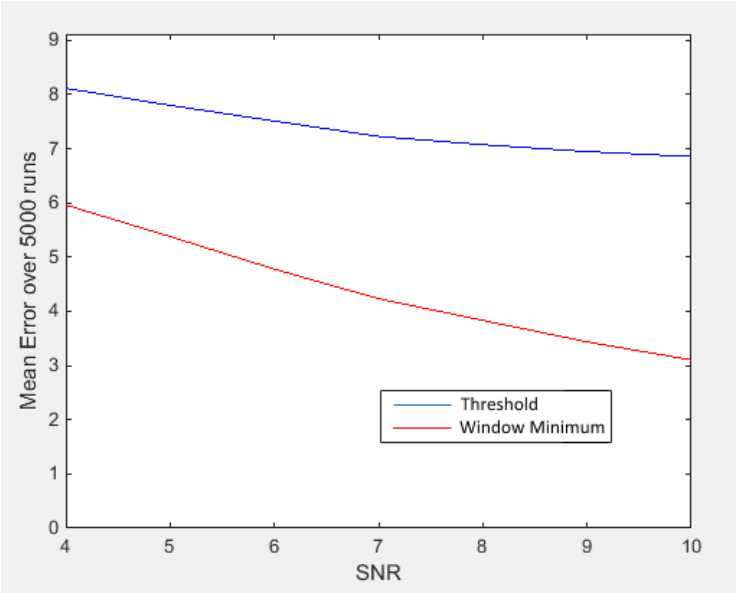

Figure 33: Detection method comparison at low SNR.

7 Hardware Implementation

After the results of the fixed point implementation of the Synchronization Unit, the hardware design could commence. This hardware design is done using the VHDL hardware description language, and written from scratch. (That is to say, not simulated using MATLAB tools or similar.)

7.1 Component Overview

The Synchronization Unit is positioned at the start of the receiver, as can be seen back in Figure 11. The task of the synchronization unit is to find the beginning of the transmitted signal, estimate and correct the Carrier Frequency Offset, and remove the Cyclic Prefixes from the OFDM symbols before sending the corrected data on to the FFT block.

Figure 34 is the component overview for the sync unit. This unit takes in IQ data and sends it to the Schmidl Cox Calculation unit for processing along with the Retention unit for storage and later use. The Calculation unit uses this IQ data to calculate the S and complex P output signals. The S signal is sent to a detection unit, where the point of synchronization will be determined. The complex P signals are stored in the Retention unit until the moment of synchronization is determined, at which point the appropriate complex P samples are sent to the Estimation unit. This estimation unit calculates the angle ($arctan(\frac{y}{x})$) of this complex P value, and sends this estimation of the Carrier Frequency Offset to the Rotation unit. The Retention unit also uses the Synchronization point, along with the latency of the Calculation Unit and the Estimation unit, to determine at which point of the stored IQ values the OFDM signal should be sent to the Rotation unit. Once the Rotation unit has received both the CFO estimation and the correct IQ values, it performs the rotation of these IQ values. The output of the rotation unit is thus the corrected input values, ready for processing by the FFT block and the rest of the receiver. (Not pictured in the figure are the various enable and valid signals.)

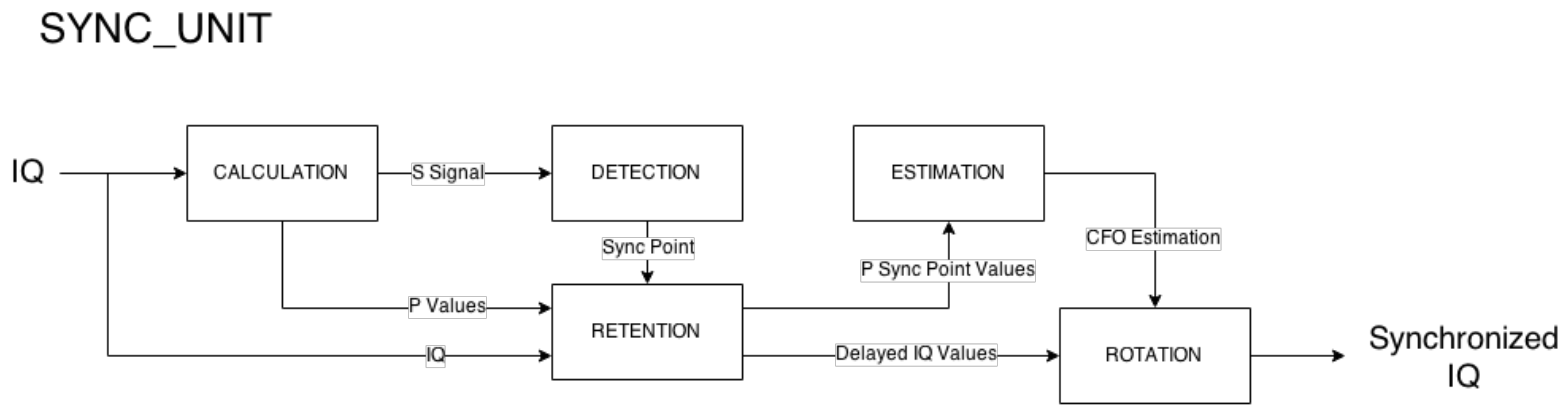

Figure 34: Hardware Overview of the entire Synchronization Unit.

7.2 Calculation Unit

The Calculation Unit is the part of the Synchronization unit that computes the various values of the Schmidl-Cox algorithm. (The entire hardware design can be found in the Appendix on page 46 in Figure 44.)

The inputs required Calculation Unit are only the incoming IQ values, along with a time-shifted version of the same values. For that reason, each of the incoming values are stored in a FIFO with a length half that of the OFDM symbol length, referred to in the figure as "L".

The top-left portion of this figure are a hardware conversion of a complex multiplication. Demonstrated in equation (18), a complex multiplication actually consists of 4 separate multiplications, along with an addition and a subtraction. (Reminder, in hardware the real and imaginary values are kept separate.)

$$(a + ib) \cdot (c + id) = a \cdot c - b \cdot d + i \cdot (a \cdot d + b \cdot c) \tag{18}$$

Therefore, implementation of the Schmidl-Cox complex P value multiplication (equation 2 on page 16) can be seen in Figure 35. The conjugation of the imIQ signal is performed by simply inverting the bits and adding one, as it is the case for signed values.

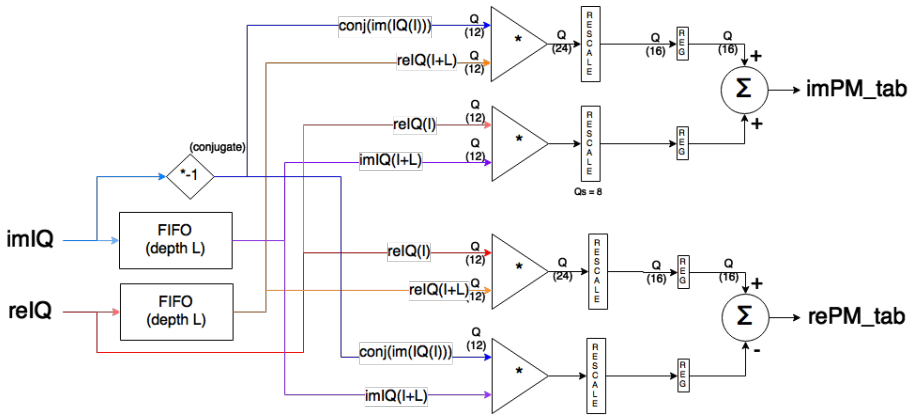

Figure 35: Hardware design of the intermediate P value calculation.

The iterative portion of the Schmidl Cox algorithm (again, seen in equation 2 on page 16) is broken up into two separate iterative sums, shown in equations 19 and 20.

$$\text{PM\_temp(I)} = \text{PM\_tab(I)} - \text{PM\_tab(I-L)} \tag{19}$$

$$\text{P(I+1)} = \text{P(I)} + \text{PM\_temp(I)} \tag{20}$$

The "L" length delay in equation 19 is created by the FIFO immediately preceding the first adder. Both of the real and imaginary portions of this calculation are performed in the same manner. After this calculation has been performed, the complex P values are ready to be passed on to the retention unit. The hardware implementation of this iterative P calculation is seen in figure 36.

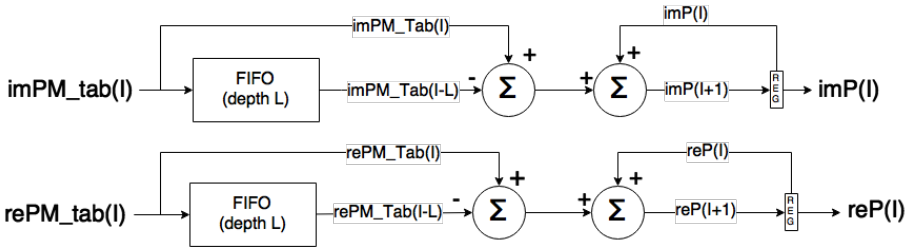

Figure 36: Hardware design of the iterative P calculation.

Meanwhile the hardware design is also calculating the R value of the Schmidl-Cox algorithm, seen in equation 4 on page 16. This begins with calculating the squared absolute value of the incoming complex signal, equation 21 showing how this is broken down in hardware.

$$\|a + ib\|^2 = \|a\|^2 + \|b\|^2 \tag{21}$$

The current design shows the implementation of 21 at two separate points in the lower left region, creating the "RM tab" and its L time-shifted sibling. (Of note, the absolute value calculation using the time-shifted input values could be replaced by a single FIFO, if desired.) These value are then summed in a manner similar to the P values shown in equations 19 and 20. At this point, the R value is obtained. This hardware design is illustrated in Figure 37.

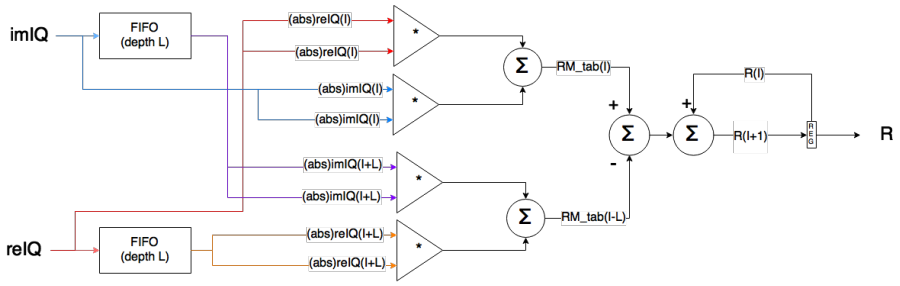

Figure 37: Hardware design of the R value calculation.

Finally, to calculate the S value, as per equation 5 on page 17, the squared absolute value of P is calculated just as in 21 and seen in Figure 38, while the squared value of R is obtained through a simple multiplication with itself. These values are then both passed to a divider, thereby calculating the S value to be sent to the Detection Unit.

Altogether this design calls for 11 separate multipliers, 11 adders, 4 FIFOs of length L, one divider (which contains a multiplier, along with adders, etc.), and several registers. (With the option to substitute 2 multipliers and an adder for another FIFO of length L.)

7.3 Detection Unit

The detection unit simply takes in the S signal calculated by the calculation unit, constantly monitoring whether the current S sample has passed a predetermined

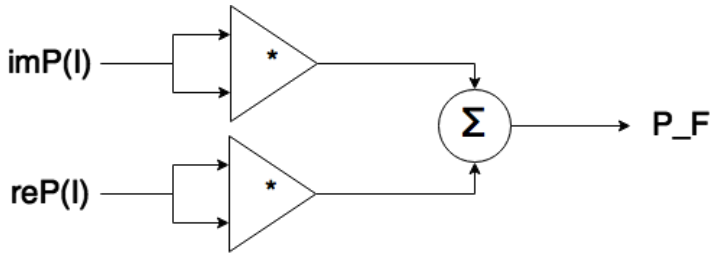

Figure 38: Hardware design of the absolute value calculation.

threshold value, indicating the arrival of the Schmidl-Cox preamble.

The method implemented for this detection unit was simply the threshold center detection method, as detailed in Section 4.2.1. This was mostly because the implementation was somewhat easier, as well as being less hardware complex.

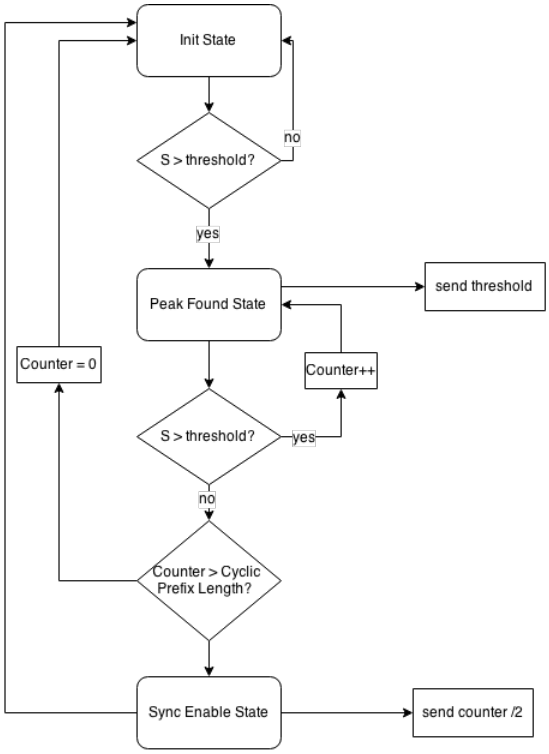

Figure 39: Detection State Machine.

Figure 39 shows a simplified state machine depicting how the detection unit behaves. The unit does nothing until the S values becomes greater than the threshold value, at which point it notifies the Retention unit. As long as the threshold has been surpassed, a counter increases. Once the incoming S value is no longer greater than this threshold, if the counter is longer than the length of a cyclic prefix, the counter length is divided by two to indicate the center

of this peak, and this value is also sent to the Retention unit. If the counter is less than the length of a cyclic prefix, this peak is treated as a temporary aberration, and the state machine resets itself.

7.4 Retention Unit

The Retention unit is a reasonably simple component that stores the incoming IQ values and calculated P values when it is not known whether they will be needed or not.

The reason it is not known is because there is a delay between the receiver receiving the Schmidl-Cox preamble and the receiver realizing that the Schmidl-Cox preamble has arrived. This is partially due to the time required by the calculation unit for calculation of the values indicating that the preamble has arrived, but mostly due to the time required by the detection unit to ensure that the central sync point has been found.

Once the Sync point has been found, the complex P values are required to make the Carrier Frequency Offset estimation. A memory begins to hold these P values once the Detection unit indicates that the threshold has been crossed. After the threshold is no longer surpassed, the complex P value contained in the middle of the currently stored data is sent to the estimation unit, as the middle sync point.

Until the sync point has been determined, there is no need for the IQ signals by the rest of the receiver. However, after discovery of the point, all of the IQ signals after that point must be sent to the Rotation unit, as they contain valuable data. It is for this reason that the IQ values are stored in an ongoing manner through use of a FIFO. After the Sync point has been discovered, the IQ FIFO data becomes valid after a period of time equal to the length of the FIFO minus the distance to the contained sync point.

Both of these memory components require to have length greater than the maximum time from the actual sync point to the point at which the S signal no longer surpasses the threshold value to ensure that all signals of value are still stored.

7.5 Estimation Unit

The Estimation unit makes use of the CORDIC algorithm in Vectoring mode to calculate the angle of the complex P value located at the sync point. In Vectoring mode, the input X and Y values constitute a vector that is iteratively rotated towards $y = 0$, as per the theory in Section 4.3. The output of CORDIC algorithm will then be the absolute value of the input vector (not used in this project) and also the angle of rotation, or the angle between the original values.

Algorithm 1 shows a simplified version of the algorithm as it is implemented in hardware. This algorithm takes in the P real value and the P imaginary value at the point of synchronization, and calculates the angle they form, or $arctan(y/x)$. In this case, the real portion of P is the x value, and the imaginary portion is the y value, and the angle between them θ.

At each stage of the algorithm, a check is made to see whether the stored y value is greater than or less than zero. If it is greater than zero the vector is rotated clockwise, otherwise the vector is rotated counter clockwise. This rotation is performed on each axis through addition or subtraction with a bitshifted

Algorithm 1 CORDIC Vector Mode Algorithm

while n < MaxStages − 1 **do**
  —*if y positive than rotate counterclockwise else rotate clockwise*
  if y(n) < 0 **then**
    x(n+1) = x(n) − 2^{-n}· y(n)
    y(n+1) = y(n) + 2^{-n}· x(n)
    β(n+1) = β(n) − atanTable(n+1)
  else
    x(n+1) = x(n) + 2^{-n}· y(n)
    y(n+1) = y(n) − 2^{-n}· x(n)
    β(n+1) = β(n) + atanTable(n+1)
  end if
end while
OutDataθ = β(maxStages − 1)
OutDataR = x(maxStages − 1)

version of the opposite axis. At each rotation the angle accumulator β adds or subtracts the next step on the arctan table.

Finally after having performed all of the steps, the y value will have been forced to zero, the x and β values will be output as the radius R and the angle *theta* of the original input vector.

In this particular implementation, there are 16 total stages of the CORDIC algorithm, that results in a 16 clock delay and requires only a 16x16 bit arctan table.

7.6 Rotation Unit

The Rotation Unit uses the rotation mode of the CORDIC algorithm discussed in Section 4.3. In this mode, one rotates a vector described by the x and y inputs by the amount described by the phase input. In the case of this synchronization unit, the phase is determined by the CFO estimation as calculated by the Estimation unit, and the input vector is the same as the incoming signal, except delayed by an amount necessary to locate the beginning of and process this incoming signal.

Algorithm 2 is very similar to that of the Estimation Unit's algorithm in Section 7.5. The difference is that this algorithm takes in the real and imaginary IQ values along with the angle of rotation to perform. Therefore, by checking if the cumulative angle at each step is greater than or less than zero, and rotating the vectors accordingly towards a zero angle, the vectors will eventually be rotated exactly as desired.

After the IQ data has been rotated by this unit, it should resemble the originally transmitted IQ data and is ready for processing by the rest of the receiver unit.

7.7 Cyclic Prefix Removal

The Cyclic Prefix is a repetition of OFDM information that is used as a guard interval, as described in Section 2.1.1. This repetition of information must be

Algorithm 2 CORDIC Rotation Mode Algorithm

while n < MaxStages − 1 **do**
  —*if* β *positive than rotate counterclockwise else rotate clockwise*
  if β(n) > 0 **then**
    x(n+1) = x(n) − $2^{-n}\cdot$ y(n)
    y(n+1) = y(n) + $2^{-n}\cdot$ x(n)
    β(n+1) = β(n) − atanTable(n+1)
  else
    x(n+1) = x(n) + $2^{-n}\cdot$ y(n)
    y(n+1) = y(n) − $2^{-n}\cdot$ x(n)
    β(n+1) = β(n) + atanTable(n+1)
  end if
end while
OutDataI = x(maxStages − 1)
OutDataQ = y(maxStages − 1)

removed by the synchronization unit before being sent further on to the FFT block and the rest of the receiver.

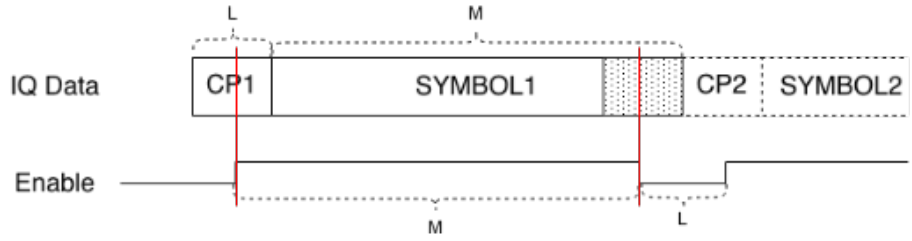

Figure 40: Removal of the Cyclic Prefix.

Once synchronization of the incoming signal has been achieved and the relative timings of the OFDM signal can be reconstructed, it is trivial to remove the cyclic prefix. This is simply done as in Figure 40 by sending a valid signal equal to the length of the OFDM Symbol, beginning halfway through where the cyclic prefix is presumed to be located. Sending the signal halfway through the cyclic prefix allows for an error in synchronization point timing by up to half of the cyclic prefix length.

8 Hardware Results

Results of the simulation of the hardware implementation of the Schmidl-Cox algorithm can be seen in Figure 41. The calculated S value seems to very closely resemble that of the fixed point implementation, which can be seen in Figure 29.

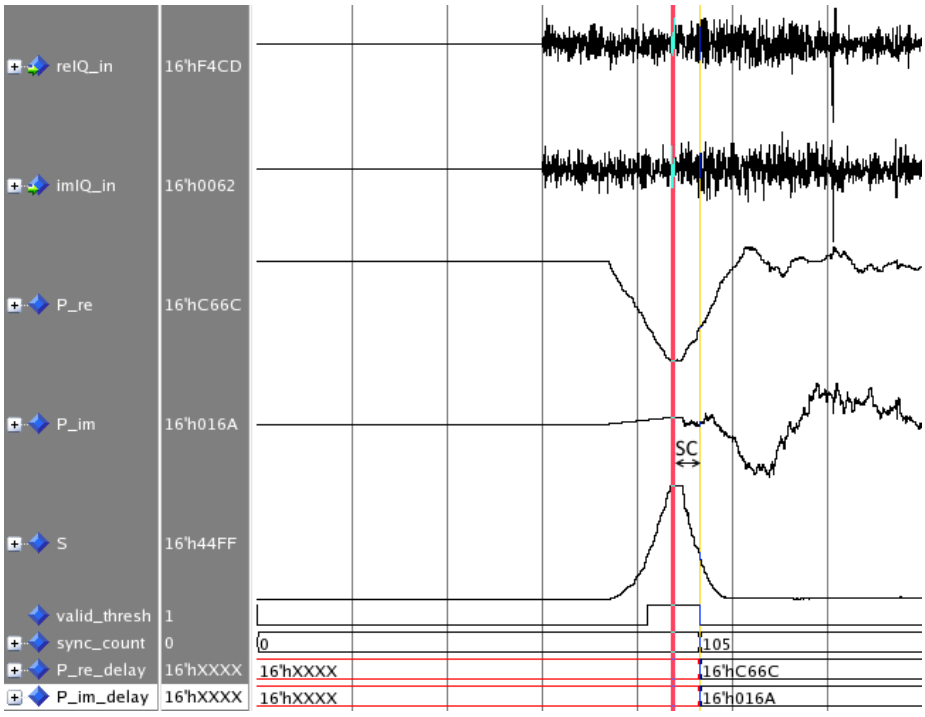

Figure 41: Schmidl Cox S signal in ModelSim.

Figure 41 also shows that the threshold detection is functional, with the sync count displaying 105 as the distance from that point to the center of the flat portion of the S signal. This distance is also marked by the yellow to red lines, which show that this distance does in fact align with the center of the S signal.

Figure 42: Zero Transmissions Errors Occurred.

Figure 42, taken from the same simulation as Figure 41, shows that the signal has been fully processed by the receiver, with 5568 samples received, and zero errors discovered. This would imply that the rotation, estimation, retention, detection and calculation have all worked correctly in unison to synchronize the incoming signal.

Figure 43 shows the hardware complexity after synthesis of the hardware design in the Xilinx ISE hardware description language development tool targeting the Xilinx Zynq 7020 FPGA board. An interesting result is that of the "Number of DSP48E1s" which refers to the number of multipliers used by this

project. As mentioned in Section 7.2 the design uses 11 separate multipliers, along with one more hardware multiplier used in the divider, totaling twelve, agreeing with Figure 44.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	2411	106400	2%
Number of Slice LUTs	4145	53200	7%
Number of fully used LUT-FF pairs	1991	4565	43%
Number of bonded IOBs	67	200	33%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48E1s	12	220	5%

Figure 43: Synthesis Results.

Otherwise it is difficult to find a point of comparison with which to compare these results, short of implementing a different method. This difficulty arises from different implementation targeting different hardware or having different limitations placed upon them. As the goal of this project is implementation over optimization, it is satisfactory to see that the hardware complexity is aligned with the theory and corresponds to a low number of occupied FPGA resources.

9 Discussion

9.1 Project Status

This project has successfully implemented a Synchronization Unit in VHDL, but time ran out before it could be successfully tested on hardware.

The unit is able to successfully recognize an OFDM signal, to calculate the contained Carrier Frequency Offset and locate the Synchronization Point, and finally send the rest of the extracted signal to the rest of the receiver, as per the project aims. This is integrated into the rest of the transceiver testbench, and this entire implementation manages to perform its duty with zero errors.

9.2 Continuation

The next stage of this project is to ensure bit-level correctness between the MATLAB fixed point implementation and the VHDL implementation. While the maximums of the calculated Schmidl-Cox P and S values seem correct, the incoming IQ data from the channel is not the same as in MATLAB, which points to something being different in the channel, which could point to issues with the introduction of the Schmidl-Cox preamble on the transmission side.

After this, synthesis of the VHDL implementation should be performed and compared to the theoretical hardware design. For example, the Schmidl-Cox calculation uses 11 multipliers, and the synthesis should be identical to this. This would be done to ensure that the VHDL code contains no coding abnormalities that result in improper synthesis.

When the synthesis corresponds correctly, it is ready to test in hardware. There is a pre-existing test-suite and user interface on the transceiver, and should the Synchronization unit function as desired, there would be no change to the functioning of the FPGA boards, except that they would no longer need to be connected, and thus finally use fully wireless transmissions.

9.3 Future Improvements

In addition to that which is left to be completed, there are several continuations that could be made to the code in this project.

- Genericness

  The VHDL and MATLAB in this project have been implemented solely with 16-bit in mind, as this was most appropriate for the current hardware. However, should the hardware design need to be changed in the future, or should testing of lower or higher bit counts desired to be performed, this firmware would need to be modified. Another benefit of making this code bit-generic would allow for it to be inserted into other projects, including other people's projects.

- Optimization

  Few hardware optimizations remain in this project and design. For example, the length of the FIFO implemented in the retention unit was chosen arbitrarily, while through testing a smaller FIFO could surely be found.

In addition to this, when implemented on a larger scale, the target hardware constraints would need to be taken into consideration, and there are many areas of code that could substitute certain FIFOs for hardware multipliers or similar, but this would need to be based on a case by case basis.

Overall however the footprint of the major parts of this design have been successfully implemented with relatively low hardware complexity.

- Detection Methods

  In the floating point comparison between the two detection methods, the minimum window method was found far superior to the threshold detection method. However, it was the threshold method which became implemented in hardware. There are several reasons for doing so, the most important of which perhaps is that the threshold detection method was more than adequate for all testing purposes while being easier to implement. However, the minimum window method is not necessarily all upside, as it entails a higher hardware complexity, and it's entirely possible that the minimum window method would not be enough better to warrant the hardware implementation cost.

- Hardware Divider

  While currently completely adequate, the accuracy of the hardware divider plays an extremely important role in the calculation of the Schmidl-Cox S value, as displayed in the software results Section (6.1). As the difference between the fixed point and the floating point calculations are nearly identical otherwise, improvements made to the hardware divider are the most noticeable. However, this will always be a trade-off in accuracy vs. hardware complexity, and as stated the current hardware divider is adequate for the needs of this testing device.

- Alternative Synchronization Methods

  Alternative synchronization methods were discussed in Section 4.2.3, and while the Schmidl-Cox algorithm selected, testing of other Synchronization methods would be useful, even if only for purposes of comparison. As the implemented algorithm was proved successful, the most interesting aspects of the alternatives would be robustness and hardware complexity, which can be difficult to simply envision.

- Extended Use of the CORDIC Algorithm

  The CORDIC algorithm is a very versatile algorithm that is used by both the Rotation and Estimation units. However, there are even other applications for this algorithm, for example when calculating the absolute value of real and imaginary vectors in the Calculation Unit. The complexity of this implementation could be compared to that of the otherwise required two multipliers, in addition to the concessions that would need to be made due to the delay added by the CORDIC algorithm.

- Bit-Error Rate & Other Testing

While Figure 42 showed that no errors occurred during transmission, this is just one example under ideal conditions. To properly test the effectiveness of the Schmidl-Cox synchronization method and to compare it to other solutions the bit error rate should be thoroughly tested at various SNRs.

There are many forms of testing, most notably at low SNR values and extreme CFO conditions that have not yet been performed on this synchronization unit. These tests are of high importance, and could indicate whether hardware redesigns or improvements are required.

Conclusion

In conclusion, the aims of this project are well met. The Synchronization Unit has successfully been implemented in VHDL, and performs correctly after being integrated with the test bench of the entire transceiver. The Schmidl-Cox algorithm is successfully able to locate the point of synchronization, and to infer the Carrier Frequency Offset of the incoming signal.

References

[1] Radio Electronics - Ian Poole, *Introduction to OFDM concepts*, http://www.radio-electronics.com/info/rf-technology-design/ofdm/ofdm-basics-tutorial.php

[2] National Instruments, *Multipath Propagation Example Figure*, http://www.ni.com/cms/images/devzone/pub/page%206_%20image%201.jpg

[3] Telecom Hall, *Cyclic Prefix Timing Illustration*, http://www.telecomhall.com/what-is-cp-cyclic-prefix-in-lte.aspx

[4] National Instruments AWR Corporation, *QAM Mapping Example Figure*, https://awrcorp.com/download/faq/english/docs/VSS_System_Blocks/images/i80211a_map_fig2.png

[5] Keysight Technologies, *OFDM Principals Illustrations*, http://rfmw.em.keysight.com/wireless/helpfiles/89600B/webhelp/subsystems/wlan-ofdm/Content/ofdm_basicprinciplesoverview.htm

[6] University of Colorado at Boulder, *Image of block overview for QAM transmission*, http://ecee.colorado.edu/~ecen4242/adsl/adsltechnology_files/moz-screenshot-1.jpg

[7] Radio Electronics - Ian Poole, *Introduction to QAM*, http://www.radio-electronics.com/info/rf-technology-design/quadrature-amplitude-modulation-qam/8qam-16qam-32qam-64qam-128qam-256qam.php

[8] Veeresh Taranalli, *Carrier Frequency Offset in Single Carrier and OFDM Systems*, http://veeresht.info/blog/cfo/

[9] Phydas - M.Bellanger, *FBMC Physical Layer: A Primer*, http://www.ict-phydyas.org/teamspace/internal-folder/FBMC-Primer_06-2010.pdf

[10] METIS Project, *Test-bed/demonstration results*, https://www.metis2020.com/wp-content/uploads/deliverables/METIS_D1.3_v1.pdf

[11] Timothy M. Schmidl and Donald C. Cox, *Robust Frequency and Timing Synchronization for OFDM*, http://home.mit.bme.hu/~kollar/papers/Schmidl2.pdf

[12] Hlaing Minn, Vijay K. Bhargava, Khaled Ben Letaief, *A Robust Timing and Frequency Synchronization for OFDM Systems*, http://core.ac.uk/download/pdf/21751016.pdf

[13] Defense Electronics, *Practical Costas loop design*, http://defenseelectronicsmag.com/site-files/defenseelectronicsmag.com/files/archive/rfdesign.com/images/archive/0102Feigin20.pdf

[14] University of Oslo, *CORDIC Figures & Theory*, http://www.uio.no/studier/emner/matnat/ifi/INF5430/v12/undervisningsmateriale/dirk/Lecture_cordic.pdf

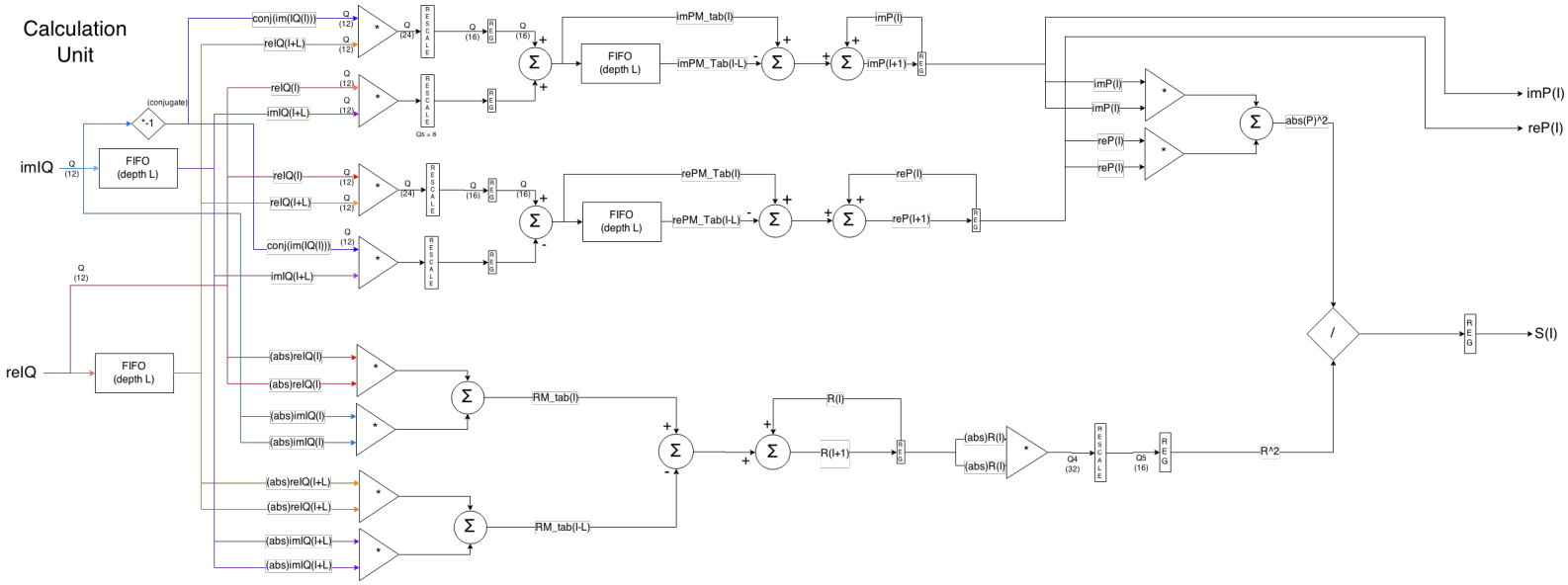

Figure 44: Hardware sketch of the Schmidl-Cox calculation unit.