Techniques for Low-Cost Spectrum Analysis on Quadrature Demodulation Architectures

A Thesis submitted
to the College of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
University of Saskatchewan
Saskatoon

By

Brendon Fredlund

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Masters degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Electrical and Computer Engineering
University of Saskatchewan
57 Campus Drive
Saskatoon, Saskatchewan, Canada
S7N5A9

Abstract

The Decimator, an SED Systems Ltd. product, is a PCI slot card that performs both time and frequency domain measurements of given input signals. It is essentially a more economical version of a bench spectrum analyzer or oscilloscope, with a PC interface. Several issues limit the speed and accuracy of the results of the Decimator, and the study of these issues is the focus of this thesis. These issues, including but not limited to, are as follows: 1) Imbalances between the received In-phase and Quadrature-phase channels; 2) The FFT and Windowing functions are performed by a microcontroller, but it is desired that they be migrated to an FPGA. While solutions to improve the first issue is being implemented and verified, the second issue is not one of simply reducing a source of error. The second issue requires a cost-benefit analysis on the migration of these signal processing algorithms from an ARM microcontroller to a Xilinx FPGA.

Acknowledgments

I greatly appreciate the guidance my supervisor, Dr. Dinh, has given me during the writing of this thesis. His encouragement and insights were important in helping me find my way through uncharted waters.

I would also like to thank Dr. Salt for his time and consideration. He inspired me to pursue a Masters degree, and he gave me the contacts I needed to find a meaningful and practical thesis topic. He showed me that I did not need to jump into industry in order to meet professionals and work on real-world problems.

My father, Delwyn, has supervised hundreds of graduate students during his tenure at the U of S, and the advice he imparted was invaluable to my journey through the Masters program. By discussing my graduate program with him I was able to see him in a different light than I did growing up. He went from being "Dad" to being "Professor", and through this transformation I was easily able to identify the skills and abilities he has utilized in his decorated career as an Engineer.

The Engineers at SED Systems Ltd. have been more than generous with their time during the course of this thesis. Mr. Akins, Mr. Gunderson, Mr. Armstrong, and Mr. Warkentin were able to provide me with a meaningful thesis project and they helped me greatly along the way. There was no question or request too mundane for them, and for that I am truly grateful.

This thesis was funded by the TRLabs Graduate scholarship and by Professor Dinh. I sincerely appreciate having the financial means necessary to complete my studies.

Table of Contents

Nomenclature

ε	Gain imbalance coefficient
ϕ	Phase imbalance coefficient

Acronyms

ADC	Analog to Digital Converter
BER	Bit Error Rate
BFP	Block Floating Point
BSS	Blind Source Separation
DA	Data Aided
DAC	Digital to Analog Converter
DCR	Direct Conversion Receiver
DIF	Decimation In Frequency
DIT	Decimation In Time
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IC	Interference Cancellation
I	In-phase
IRR	Image Rejection Ratio
NDA	Non-Data Aided
OFDM	Orthogonal Frequency Division Multiplexing
LPF	Low Pass Filter
Q	Quadrature-phase
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
SNR	Signal to Noise Ratio
VHDL	Very high speed integrated circuit Hardware Description Language

List of Figures

CHAPTER 1 : INTRODUCTION

1.1 General

Communication schemes have developed from simple dot and dash Morse code to complex high speed systems where numerous transmitters are simultaneously communicating with numerous receivers. The continual drive to explore new ideas and push known boundaries keeps technology marching steadily forward.

Global communications standards have emerged and are enforced federally in all modern countries. The regulations require wireless communication to adhere to stringent transmission and reception constraints. Power and bandwidth are the two most limited factors that ensure a wide variety of wireless communication systems are able to co-exist without interference. An example of one such highly regulated frequency band is the L-band, which is used for satellite communication, and ranges from about 1 to 2GHz. From the perspective of a designer, a spectrum analyzer may be used for research and development, troubleshooting, and the verification of its functionality as a legal transmitter/receiver device. From the perspective of a federal regulator, a spectrum analyzer may be used to monitor the frequency spectrum to ensure that legal limits are observed.

The complexity of communication systems increases with each advancement in technology, and new methods must be developed for verification and analysis. An example of one such method for analyzing a signal is the Fast Fourier Transform, (FFT), which allows a time based signal to be viewed in the frequency domain (i.e., spectrum). A spectrum analyzer is the hardware realization of the FFT, and it has become a common tool used to monitor communication systems. One such device is the Decimator.

1.2 The Decimator

The Decimator is a Peripheral Component Interconnect, (PCI), slot form factor L-Band spectrum analyzer developed by SED Systems Ltd.. The Decimator is functionally a spectrum analyzer and an oscilloscope with a Personal Computer, (PC), interface. The Decimator receives power from the PCI slot of the PC and communicates with the host computer via an Ethernet connection. As long as the Decimator is powered, any PC with network access can use the Decimator and display its output either in a browser window, or the provided software Application Programming Interface, (API).

The market value of the Decimator comes as a result of its low cost in comparison to equivalent bench spectrum analyzers and oscilloscopes, as well as its small form factor. A device of this nature works well in embedded systems because of its remote access and low power requirements. While the Decimator was initially developed as a low cost test device for a communication system that was being developed, its market value was recognized and has since become one of SED Systems' stand-alone products. The Decimator uses a direct conversion receiver architecture to convert a received Radio Frequency, (RF), signal directly to baseband. This architecture has allowed the Decimator to retain its small form factor and low power requirements. However, it has also led to the introduction of errors that limit its accuracy.

1.3 Known Decimator Issues

1.3.1 I/Q Imbalance

The main issue affecting the Decimator is the introduction of In-Phase (I) and

Quadrature-Phase (Q) imbalances in the received signals from the RF receiver. The gain and phase imbalances are a result of the Direct Conversion Receiver (DCR) architecture that the Decimator employs. In certain applications these imbalances cause a relatively significant error to be present at the output from the receiver. The output of the Decimator is processed and either the frequency or time domain information is displayed on a computer screen. In and of itself these errors may not be significant enough to warrant correction in some applications, but the usefulness of the Decimator is directly linked to the accuracy of its calculations. Since other communication schemes and transmitters can be tested, calibrated, and verified using the Decimator, residual errors may also be transferred, and possibly amplified, in other applications.

1.4 Other Decimator modifications

The majority of known error present in Decimator output signals is due to the issues described above. However, not all changes to the Decimator are being done for the sole purpose of increased accuracy. Speed is also a factor that must be considered. Changes to a major bottleneck in the Decimator's signal processing system will also be studied in an effort to increase its speed. In the existing design, a Xilinx Spartan-3 FPGA and an Analog Devices ARM microcontroller shared the signal processing in the Decimator. The microcontroller currently handles two signal processing algorithms that limit the speed of the Decimator. These two algorithms are “windowing” and the “Fast Fourier Transform,” (FFT). By coding these two algorithms in the FPGA and removing them from the microcontroller, an increase in speed should be achieved at the cost of some accuracy. This trade off comes as a result of the increased streamlining ability of the FPGA and performing the mathematical calculations in a fixed point precision environment. The fixed point precision math of the FPGA will be compared and contrasted with the 32-bit floating point precision math of the microcontroller to help evaluate this migration.

1.4.1 Windowing

A windowing function is a filter that converts a continuous signal into one where the only non-zero values are those within the bandwidth of the window function. The windowing function acts as a buffer and allows a finite length of samples to be analyzed by hardware-implemented signal processing algorithms. In the case of the Decimator, the windowing function buffers the data for the FFT. The windowing function is currently implemented in the microcontroller using floating point precision calculations, but this causes a bottleneck in the signal processing chain. The effect of the fixed point precision on a windowing algorithm will be explored from theoretical and practical viewpoints. The windowing algorithm will be implemented in the FPGA to verify its performance and help conclude whether this migration is economical.

1.4.2 Fast Fourier Transform

The FFT is a practical DSP algorithm that allows the Discrete Fourier Transform, (DFT), of a signal to be calculated in real-world devices such as FPGAs and microcontroller. The FFT converts a signal from the time domain to the frequency domain by calculating the frequency components that are present in a given waveform. Since the FFT is behind much of the functionality of the Decimator, its performance is of utmost important to the overall performance of the Decimator.

Implementing a FFT in a FPGA is not a new endeavor. Xilinx, for example, has patented logic cores that can be dropped into a design and easily configured in a short period of time. By implementing the Xilinx core in the FPGA of the Decimator, it will be possible to perform an economic evaluation of the migration and determine whether the change is feasible. Specifically, the results of the implemented algorithm will be analyzed to confirm whether or not the solution is faster than the current implementation. The degradation in accuracy will also be studied to ensure that it is not beyond acceptable levels.

1.5 Decimator Modification Overview

A basic overview of the Decimator can be seen in Figure 1.1. The incoming L-band signal is received by the analog RF front-end components. The received signal is converted from analog to digital form and is passed to a chain of signal processing algorithms. The signal processing algorithms demodulate the received signal so the samples passed to the time and frequency domain calculations are at baseband.

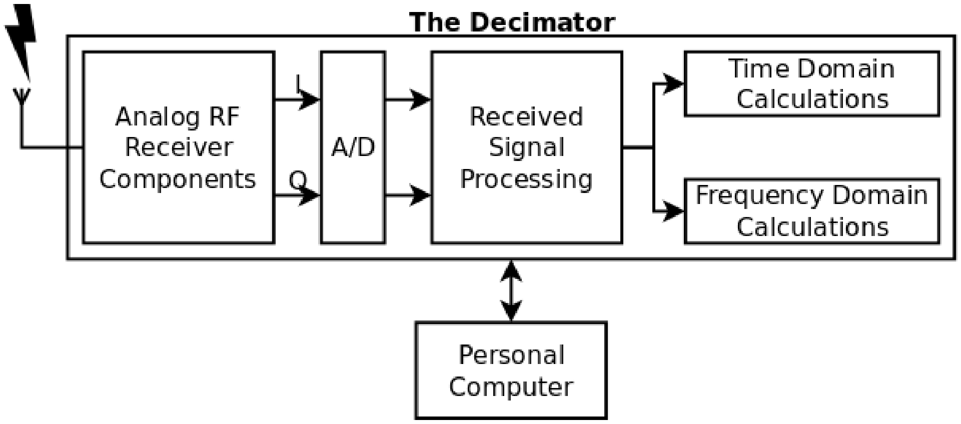
The Decimator
Analog RF Receiver Components
I
Q
A/D
Received Signal Processing
Time Domain Calculations
Frequency Domain Calculations
Personal Computer

Figure 1.1: Decimator block diagram.

An I and Q imbalance correction scheme will be sought that can be implemented in the “Received Signal Processing” block from Figure 1.1. Implementing an I and Q imbalance correction algorithm in the “Received Signal Processing” block is desired because it would not require hardware changes to be made to the Decimator. The correction algorithm would only require modification to the firmware of the Decimator.

The Windowing and FFT algorithms are present in the “Frequency Domain Calculations” block seen in Figure 1.1. Migrating these two algorithms from the microcontroller to the FPGA in the Decimator will not change the block diagram, it will only change the way the Decimator calculates the functions.

1.6 Summary

This thesis will seek Digital Signal Processing (DSP) solutions that should improve the accuracy of the Decimator while not disrupting the current data throughput. The proposed solutions will result in a stand-alone signal processing algorithm that will work with a wide variety of incoming signals, as is expected of a spectrum analyzer. The DSP algorithms should require neither a training sequence nor a calibration signal, (which would require transmitter modification), to help with the correction of the gain and phase imbalances. The DSP algorithms will be implemented between a Spartan-3 FPGA and an ARM microcontroller so that the proposed solutions are compatible with the hardware requirements.

The windowing and FFT algorithms will be theoretically analyzed to show the effect of fixed-precision calculations. Both algorithms will then be migrated from the microcontroller of the Decimator to its FPGA to obtain bit accurate results. The bit accurate simulations, along with the theoretical analysis, will help determine whether the changes are economical and worth implementing in all new Decimators.

1.7 Thesis Outline

This thesis provides detailed descriptions related to the background, concepts, and implementation of the proposed Decimator modifications. Chapter 2 gives a theoretical basis for the proposed changes and reviews literature pertinent to the issues related to the Decimator. Chapter 3 provides details of the changes that are proposed and how each change ought to be simulated for verification. Chapter 4 reviews the simulation results obtained from the proposed methodology. Chapter 5 discusses the results and forms conclusions based on the findings. Future work is also suggested.

CHAPTER 2 : LITERATURE REVIEW AND THEORY

2.1 Introduction

This thesis is not proposing a radically new system; rather, its purpose is to take known solutions to given problems and evaluate whether these solutions can be successfully used to solve the known issues within the Decimator. The issue of I and Q imbalance in DCRs is well documented. Various ways of dealing with I and Q imbalance will be discussed and evaluated to show whether previously proposed solutions can provide an acceptable solution to this problem. The algorithm migrations will be discussed from a theoretical standpoint and simulated to study the implications of the proposed changes.

2.2 Direct Conversion Receivers

The driving motive behind technological advances in communication systems is the desire to make transceivers with higher levels of integration. Bulky off-chip components that are prominent in the popular heterodyne receivers are a limiting factor in system integration because of their high power requirements and larger form factors. This has led to transceiver designs such as the low-IF, (Intermediate Frequency), and zero-IF, or direct conversion, receivers. The low-IF and zero-IF receivers greatly reduce the off-chip hardware requirements, and thus improve efficiency and reduce size. The issues related to direct conversion receivers are therefore the major topic of study in this thesis since the Decimator utilizes the direct conversion receiver architecture.

Direct conversion receivers use quadrature demodulation to split the received signal into real, (In-phase), and imaginary, (Quadrature-phase), components by multiplying incoming signals by orthogonal sine and cosine functions. Figure 2.1 depicts the basic architecture for a direct conversion receiver. Theoretically, quadrature mixing removes the need for anti-alias filtering by infinitely attenuating the image of the signal. Practically, however, there will always be a certain amount of gain and phase imbalances between the I and Q branches of the receiver because of the inability to perfectly match the receiver's analog components [5]. The errors that are introduced prevent the direct conversion receiver architecture from being used in many high-end applications.

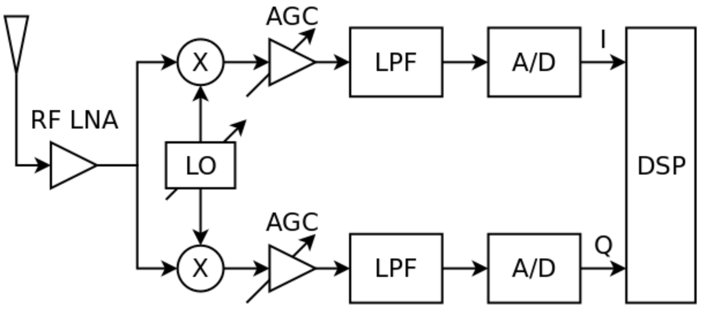

Figure 2.1: Direct conversion receiver architecture.

The error-free Local Oscillator, (LO), can be modeled as,

$$X_{LO}(t)=\cos(w_{LO}t)-j\sin(w_{LO}t) \qquad (2.1)$$

where $\cos(w_{LO}t)$ demodulates the I branch, and $-jsin(w_{LO}t)$ demodulates the Q branch. An arbitrary quadrature incoming signal, $s_M=s_I\cos(wt)+s_Q\sin(wt)$, is split into its real and imaginary branches when it is multiplied by the LO function. The received in-phase signal is mathematically demodulated as shown below.

$$R_I(t)=\cos(w_{LO}t)S_M$$

$$R_I(t)=\cos(w_{LO}t)(S_I\cos(wt)+S_Q\sin(w_{LO}t)) \qquad (2.2)$$

$$R_I(t)=S_I\cos(wt)\cos(w_{LO}t)+S_Q\sin(w_{LO}t)\cos(wt)$$

The LO frequency is tuned to the transmitted signal frequency, so $w=w_{LO}$. Using standard trigonometric identities yields

$$R_I(t)=\frac{1}{2}S_I+\frac{1}{2}S_I\cos(2\mathrm{wt})+\frac{1}{2}S_Q\sin(2\mathrm{wt}) \qquad (2.3)$$

Following the down-conversion is the Automatic Gain Control, (AGC), which equalizes the received signal. The LPF, as seen in Figure 2.1, then removes the high frequency components containing $2\mathrm{w}$. Only the baseband components of the original signal remain. The quadrature-phase branch equation seen in Equation 2.4 can be derived similarly.

$$R_Q(t)=\frac{1}{2}S_Q+\frac{1}{2}S_Q\cos(2\mathrm{wt})+\frac{1}{2}S_I\sin(2\mathrm{wt}) \qquad (2.4)$$

2.3 I/Q Imbalance

The dual path architecture makes the quadrature demodulator prone to gain and phase mismatches between the I and Q branches, and these are called I and Q imbalances. Analog component imperfections alter the received signals differently despite an identical signal processing chain in both branches. The result is a difference in the gain and phase between the I and Q branches of the received signal. This causes the image of the signal to act as interference on top of the desired signal. In theory, a direct conversion receiver can provide infinite attenuation to the received image signal, however, in practice this image cannot be fully removed. Figure 2.2 depicts a RF to baseband conversion and shows the effect that the image has on the received signal at baseband.

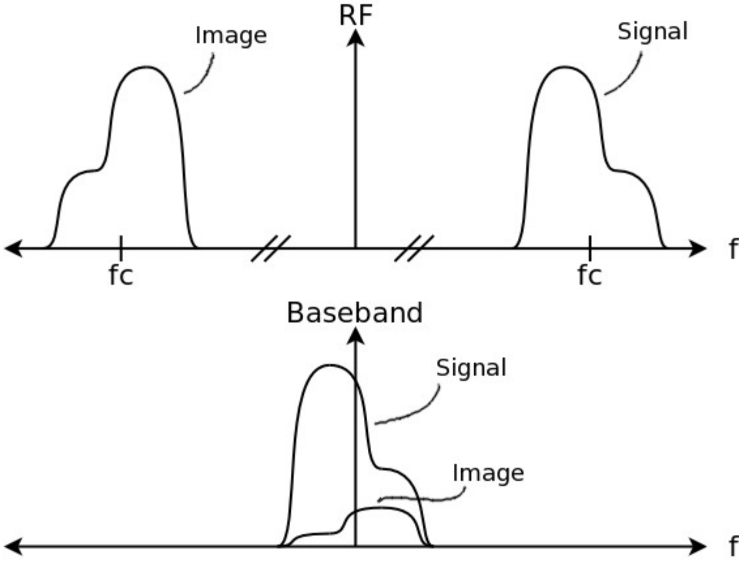

Figure 2.2: Example RF and baseband spectra depicting an I/Q imbalance.

The imperfections in the receiver can be modeled by the complex Local Oscillator (LO) function in the time domain as,

$$X_{LO}(t)=(1-\varepsilon)\cos(w_{LO}t-\phi/2)-j(1+\varepsilon)\sin(w_{LO}t+\phi/2) \qquad (2.5)$$

where ε is the gain imbalance factor and ф is the phase imbalance in radians. The gain imbalance in dB is found by,

$$\beta=20\log((1+\varepsilon)/(1-\varepsilon)) \qquad (2.6)$$

Equation (2.5) shows an equal amount of the imbalances being applied to the I and Q channels. This is an appropriate representation because the difference in gain and phase between the two channels is what is important, and not the absolute values. Therefore, ε and ф are determined by finding the differences between the gain and phase of the two channels. It is possible to model ε and ф by applying half the total errors to each channel. This concept is important, and will be discussed later when the correction architectures are discussed.

Using Euler's formula and some basic mathematical considerations, the unbalanced Local Oscillator (LO) signal can be expressed as,

$$x_{LO}(t)=K_1 e^{-jw_{LO}t}+K_2 e^{jw_{LO}t} \tag{2.7}$$

where K_1 is the desired signal, and K_2 is its image. Mathematically, K_1 and K_2 are,

$$K_1=\frac{(1-\varepsilon)e^{j\frac{\phi}{2}}+(1+\varepsilon)e^{-j\frac{\phi}{2}}}{2} \tag{2.8}$$

$$K_2=\frac{(1-\varepsilon)e^{-j\frac{\phi}{2}}-(1+\varepsilon)e^{j\frac{\phi}{2}}}{2} \tag{2.9}$$

To obtain infinite attenuation of the image, $\varepsilon = 1$ and $\phi = 0$. This would lead to $K_1 = 1$ and $K_2 = 0$, and thus an ideal down-conversion of the RF signal to baseband. It is not currently possible to implement a direct conversion receiver without I and Q imbalances. Therefore the effect of the I and Q imbalances on the received data must be studied to determine the severity of the problem and to understand the nature of the solution.

Equation (2.10) shows how the imbalanced LO signal propagates error on the received signal. Using the imbalanced LO signal from Equation (2.5) to demodulate the received signal rather than the perfectly balanced theoretical LO in Eq (2.1) yields,

$$R_I(t)=(1-\varepsilon)\cos(w_{LO}t-\frac{\phi}{2})(S_I\cos(wt)+S_Q\sin(wt)) \tag{2.10}$$

$$R_I(t)=(1-\varepsilon)(S_I\cos(\frac{\phi}{2})-S_Q\sin(\frac{\phi}{2})) \tag{2.11}$$

Following the same procedure, the received quadrature phase branch can be shown as

$$R_Q(t)=(1+\varepsilon)(S_Q\cos(\frac{\phi}{2})-S_I\sin(\frac{\phi}{2})) \tag{2.12}$$

The ratio between K_1 and K_2 gives a measure of the power of the signal versus the power of the image. To represent the attenuation achieved by the analog components in the receiver, or the Image Rejection Ratio, (IRR), the following relationship can be used,

$$IRR_{dB} = 20\log\left(\frac{|K_1|}{|K_2|}\right) \tag{2.13}$$

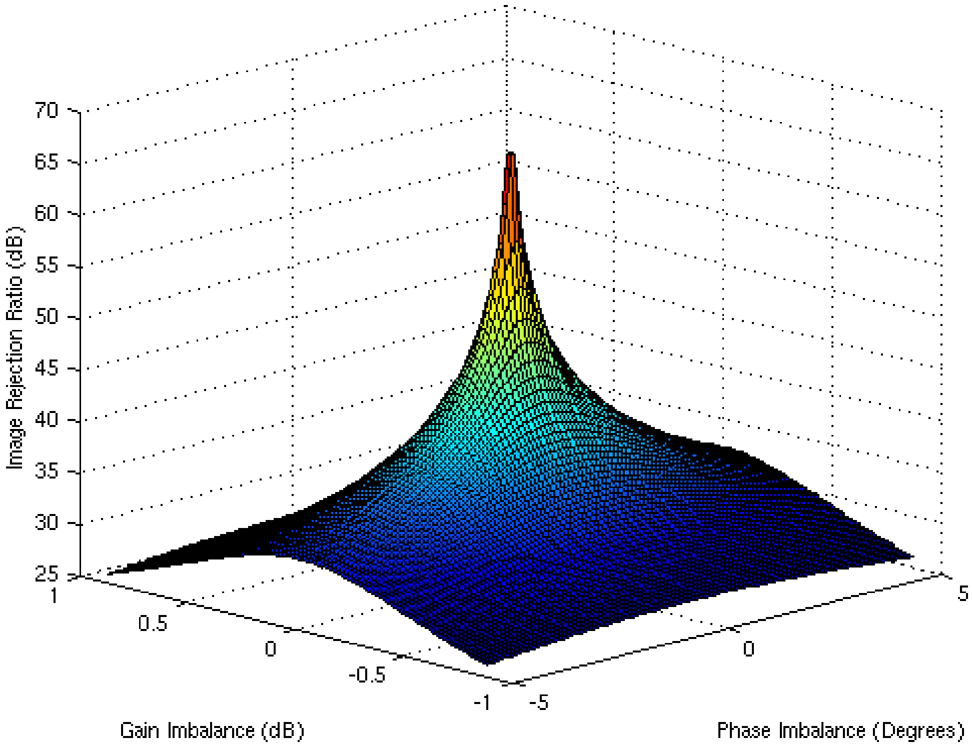

Figure 2.3: Image Rejection Ratio (IRR) with respect to gain and phase imbalances.

Figure 2.3 displays the effect that gain and phase mismatches have on the Image Rejection Ratio, (IRR). The relationship is highly non-linear, so even small errors in gain and phase lead to a significant degradation of the received signal. To achieve at least 50 dB in image attenuation, the gain and phase errors must be held to less than 0.05 dB and 0.2° respectively [6].

The modulation schemes received by the Decimator will almost all be symmetric with a mean of zero. These schemes include, but are not limited to: QAM, PSK, and OFDM. To visualize the effects of gain and phase imbalances in the receiver, Figures 2.4 to 2.6 represent QAM demodulation functions with imbalance errors. Rather than demodulating a signal with a perfectly orthogonal set of functions, an imbalanced set of functions demodulates the signal. Figure 2.4 shows how a perfectly balanced receiver will demodulate a received 4-QAM signal. Both the I and Q branches of the signal will be accurately demodulated, as the constellation depicts. Figure 2.5 illustrates the effect of a gain imbalance in the receiver. The received signal will be demodulated with an imbalance that causes the I branch data to have a higher amplitude than the Q branch data. Figure 2.6 shows the skew associated with a phase imbalanced receiver.

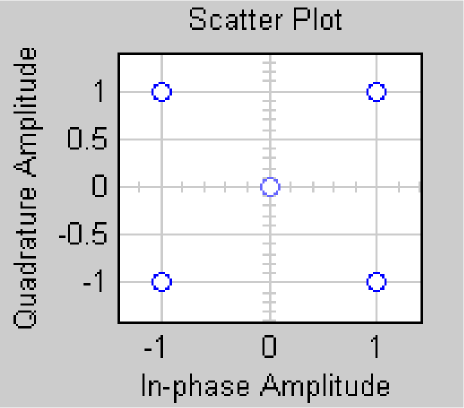

Figure 2.4: 4-QAM original modulation scheme.

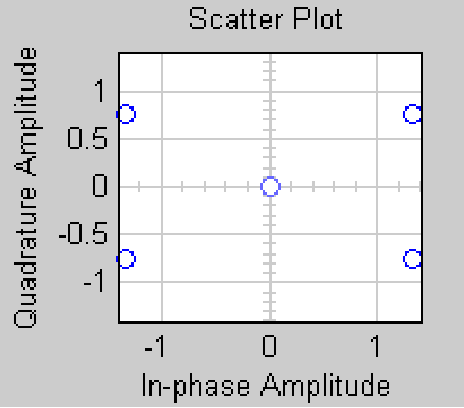

Figure 2.5: 4-QAM gain imbalanced modulation, (2dB).

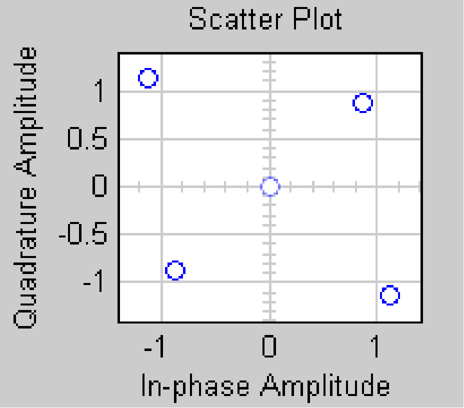

Figure 2.6: 4-QAM phase imbalanced modulation, (10º).

The Decimator receives these signals and displays them graphically to the user. The issue of concern is not one of Bit Error Rates, (BER), and data corruption, but of visual ambiguity. The Decimator is not a part of a larger system that tries to decipher instructions from the received data that is being transmitted; rather, it is simply creating a visual display of the received data for the user. An example of how this is detrimental to the usage of the Decimator becomes obvious when the practical applications of a Spectrum Analyzer are outlined. Spectrum Analyzers are commonly used for testing, debugging, verification, and calibration. From the visual inaccuracies displayed by the Decimator, it is possible to inaccurately calibrate another unit or system. The errors in the Decimator then have the possibility of propagating themselves to other systems, which is why correcting the I and Q imbalance errors is so important.

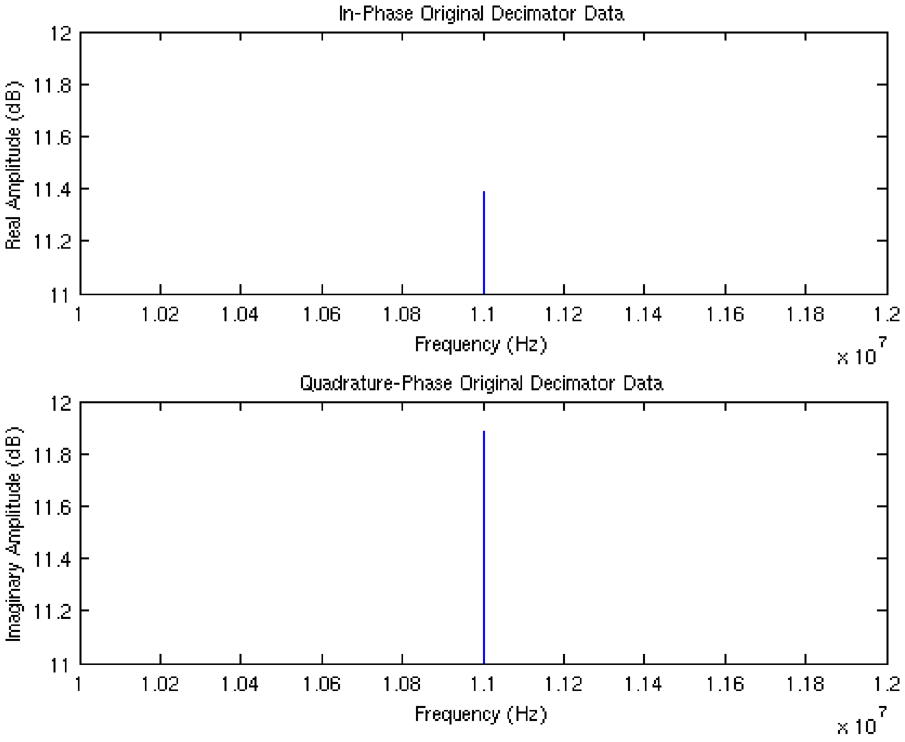

Figure 2.7: Received data corrupted by gain and phase imbalances.

An RF signal with identical I and Q data was generated using a signal generator with arbitrary waveform generation capability. The generated signal was then fed into the Decimator. A zoomed-in result of the Decimator's spectrum analysis can be seen in Figure 2.7. It is clear that the imbalances in the receiver have excessively increased the amplitude of the Q branch. Based on the received data, there is an estimated gain imbalance of 0.4922 dB and a phase imbalance of 2.9303° in the particular Decimator this data capture was obtained from. These errors lead to a 0.4944 dB difference between the peak FFT bins of the two signals. Now the problem of I and Q imbalances has been properly defined, and its effect on received signals has been quantified. The next step is to determine how these errors can be corrected.

2.4 I/Q Imbalance Correction Schemes

The errors introduced by gain and phase imbalances in the receiver should now be apparent. With direct conversion receivers being a viable option for small scale applications, and with imbalances being prevalent in their architecture, a large number of research papers have been published on the subject [5, 7, 8, 14, 16]. Each paper may bring some subtle nuance to a well-known solution but for the most part, these solutions can be categorized into several different types of correction schemes.

2.4.1 Non-Data-Aided (NDA) Correction Schemes

Non-data-aided (NDA), or blind, correction schemes are a popular form of I/Q imbalance solution that do not require knowledge of the modulation scheme and do not utilize training sequences or test tones. NDA correction methods utilize samples of the received data to determine the amount of error that is present, and then a correction is applied to remove the estimated error. The procedures whereby NDA algorithms estimate the error and then apply corrections differ from method to method. However, all methods share the fact that statistical characteristics are utilized to apply a correction to the received signal.

2.4.1.1 Blind Source Separation (BSS)

Blind Source Separation, (BSS), is the process of taking a mixture of N statistically independent signals and recovering all N signals in their original form using no outside knowledge of the source or mixing matrices. In other words, only the signal mixture is used [10]. An imbalanced direct-conversion receiver causes the I and Q channels to mix. Consequently, the signals become correlated and are no longer independent of one another.

BSS makes the implicit assumption that the mixed signals are independent. Therefore, the I and Q imbalance problem is solvable via the BSS method with *N* = 2 independent sources.

One method of BSS is Independent Component Analysis, (ICA), which provides a mathematical approach to solving the BSS problem. While there are other methods to solve the BSS problem, ICA requires that the sources be independent to achieve an applicable solution. ICA reconstructs both the source signals and the mixing matrix by minimizing the statistical dependencies, (i.e., cross-correlation), between the signals. Bell and Sejnowski showed that in signals that have a positive kurtosis, maximizing the amount of information, or entropy, was equivalent to de-correlating the signals [9]. Rather than minimizing the statistical dependencies between the signals, the proposed method attempts to correct the problem by maximizing the signal information.

Figure 2.8 displays the basic structure of the ICA problem where *S(t)* is the original signals, *R(t)* is the received signals after being mixed, and *C(t)* is the corrected output after unmixing has occurred. Both *S(t)* and *A* are unknown, but using only *R(t)* and the ICA technique, *W* can be adapted to remove the error introduced by *A*. From this model it is possible to write,

$$\begin{aligned} R(t) &= \boldsymbol{A} S(t) \\ C(t) &= \boldsymbol{W} R(t) \end{aligned} \tag{2.14}$$

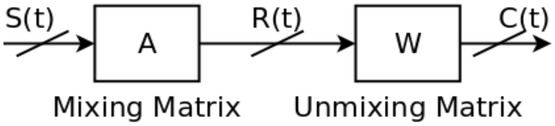

Figure 2.8: Independent Component Analysis problem definition.

The problem becomes one of developing a set of adaptive filters that will undo the effects of the mixing matrix.

A number of applicable learning algorithms have been proposed to update the coefficients in the unmixing matrix. In [11], four of the most prominent learning rules are outlined, and a hybrid learning rule is developed. While all four learning rules have been utilized successfully in other applications, arguably the most important learning rule was developed by Bell and Sejnowski [9]. Their proof of the information maximization rule, (i.e., Infomax), was developed into an effective hardware model using adaptive filters proposed by Torkkola [10]. Torkkola proposed a hardware feedback network that has been at the heart of most adaptive filter techniques for decorrelating independent signals because it provides a structure that can be realized in the receiver hardware. Figure 2.9 shows the Torkkola feedback architecture as it applies to the general case of convolved mixtures.

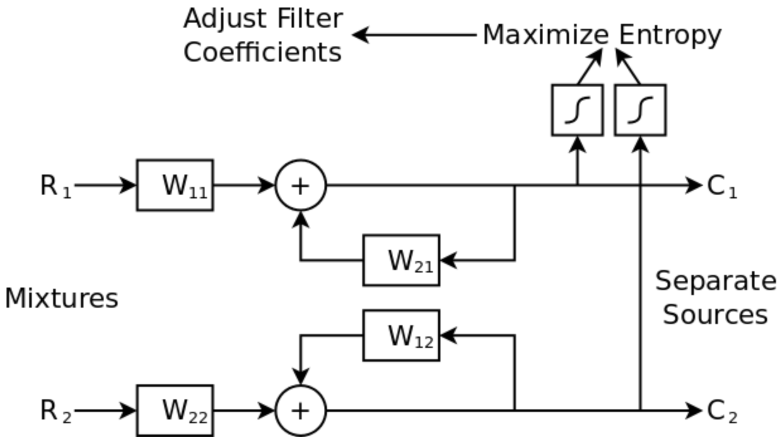

Figure 2.9: Torkkola's feedback network for separating convolved mixtures.

2.4.1.2 Interference Cancellation (IC)

The Interference Cancellation, (IC), based technique attempts to create an interference signal that, when subtracted from the received signal, will remove the erroneous component from the desired signal. This method also requires no knowledge of the received signal, but makes the fundamental assumption that the desired signal and its interfering component are uncorrelated. Since in the case of I and Q imbalance the erroneous component of the signal is from the other branch of the receiver, this relationship holds and the IC method can be utilized to solve the I and Q imbalance problem.

The basic architecture behind the IC method is depicted in Figure 2.10. The adaptive filter modifies the reference signal such that it correlates with the erroneous component of the signal but not the desired portion. The modified reference signal is then subtracted from the incoming signal in an attempt to remove the noise, or in the case of I and Q imbalance, the cross-talk. Similar to the BSS solution, IC based methods rely heavily on the learning rules employed to modify the adaptive filter.

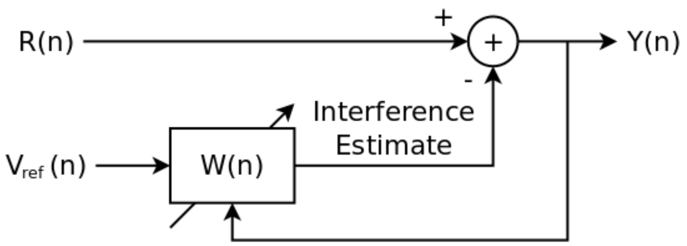

Figure 2.10: Adaptive interference canceler (IC) architecture.

2.4.1.3 Adaptive Methodologies Summary

Valkama et. al. analyzed several BSS and IC based methods and determined that both provide feasible solutions to the problem of I and Q imbalances in low-IF receivers [5]. While the Decimator uses a zero-IF receiver architecture, the methodologies are still applicable and have been applied to direct conversion receivers in other cases [6, 11].

Valkama concluded that the BSS based solutions are more robust and can correct a wider range of receiver imbalances at a variety of received signal levels. BSS also avoids the signal leakage problems that inhibit the IC based methods. On the other hand, IC based methods are more capable of handling the effects of additive noise and symbol timing errors. The IC based method can also be less sensitive to the type of modulation scheme.

Both methods track changes in I and Q imbalances with time, as is expected of the adaptive techniques. The speed and accuracy of convergence of these methods vary depending on the learning rule used to update the filter coefficients. Both methods are comparable in this respect with the proper update rule selection.

Valkama shows that both methods would be successful in various situations; however, these solutions are not the most promising when considering the nature of the Decimator [5]. The most obvious shortcoming is the fact that the number of filter coefficients needed to obtain acceptable results may easily be in the range of 60 to 100. To keep the Decimator operating at 65 MHz on the Spartan 3 FPGA, this would require a large number of dedicated multipliers. These added multipliers would also cause a considerable amount of added latency in the DSP. The adaptive filtering techniques, while promising in other applications with different hardware, would most likely not provide the best results for the Decimator.

2.4.1.4 Statistical Correction Method ("Stat")

Figures 2.4 to 2.6 show the changes that occur to a modulation scheme in the presence of I and Q imbalances. These changes alter the shape of the incoming signals such that the statistical characteristics of the received signals are also modified. There are a number of I and Q correction schemes that take advantage of these known signal alterations to estimate the error that was introduced into the signal. Error estimates can be made and then used to reverse the effect of the errors on the signals. Several statistical methods will be evaluated to determine their viability as a solution to the I and Q imbalance problem in the Decimator.

Statistical methods for solving the receiver I and Q imbalance problem are relatively new. One of the earliest methods that is applicable to the Decimator architecture is presented in [12]. The problems encountered with direct conversion receivers were discussed at length, and several variations of statistical methods for correcting gain and phase imbalances were proposed. Unfortunately, the paper does not provide any verification of the proposed methodologies. Consequently, the proposed techniques requires further study.

Around the same time as [12] was published, a methodology was proposed (and presumably independently developed) by Kocic et al. [13]. The proposed methodologies offered simple hardware implementation and promising error correction results. Shortly after the Kocic et al. [13] paper, an essentially identical methodology was proposed by Rykaczewski et al. [14]. Their proposed methodology offered good Bit Error Rate (BER) improvements while using only received data to formulate the correction scheme. The performances of both [13] and [14] have been verified, and their methodologies are similar in many respects to [12]. Therefore, it appears that the methodologies proposed in [13] and [14] should be pursued as a solution rather than [12].

The statistical-based correction scheme proposed in [13] and [14], known henceforth as "*Stat*", makes several basic assumptions about the form of the signal being received. The first assumption is that the real and imaginary portions of the received signal be statistically independent. This assumption differs from that of the BSS based solutions which make the assumption that the two original signals are uncorrelated. For the two signals to be independent, the following relationship must hold [4]

$$E[R_I R_Q] = E[R_I] E[R_Q] \tag{2.15}$$

The second assumption was that the real and imaginary portions of the received signal are of equal power. That is, the following relationship must hold.

$$E[R_I^2] = E[R_Q^2] \tag{2.16}$$

While the second assumption is not required by the BSS solutions, it is not an assumption that limits the practicality of the *Stat* solution. These assumptions will still cover approximately 98% of all incoming transmitted signals. One notable modulation scheme that does not meet the requirement in (2.16) is BPSK. BPSK transmits data that is modulated on the I branch only which means the real branch power will be much larger than the imaginary branch power.

The signal model in Equations 2.11 and 2.12 show how the desired signal is interfered with by the gain and phase imbalance components present in the receiver. The *Stat* method proposes an estimation of the variables ε and ф, and then use those estimates to reverse the effects of the imbalance error. From Equation (2.6) it is evident that the scaling factor between the two channels is $(\varepsilon+1)/(\varepsilon-1)$. Using mean-squared calculations on both channels, [14] proposes the gain imbalance estimate to be,

$$\varepsilon = \frac{\sqrt{E[R_Q^2]} - \sqrt{E[R_I^2]}}{\sqrt{E[R_Q^2]} + \sqrt{E[R_I^2]}} \tag{2.17}$$

The gain imbalance can easily be removed by multiplying each branch by the estimated imbalance.

Once the gain imbalance has been corrected, the phase imbalance can be addressed. Squaring the I and Q branches yields

$$S_I^2 = R_I^2 \cos^2(\frac{\phi}{2}) + R_Q^2 \sin^2(\frac{\phi}{2}) - R_I R_Q \sin(\phi) \tag{2.18}$$

$$S_Q^2 = R_I^2 \sin^2(\frac{\phi}{2}) + R_Q^2 \cos^2(\frac{\phi}{2}) - R_I R_Q \sin(\phi) \tag{2.19}$$

After the assumption in Equation (2.16) is acknowledged, it can be seen that,

$$E[S_I^2] + E[S_Q^2] = E[R_I^2] + E[R_Q^2] \tag{2.20}$$

And since $R_I R_Q = S_I S_Q - 1/2 \sin(\phi)(S_I^2 + S_Q^2)$, it follows that,

$$\phi = -\arcsin\left(\frac{2E[R_I E_Q]}{E[R_I^2] + E[R_Q^2]}\right) \tag{2.21}$$

The calculations required to find the phase estimate are quite simple to compute in an FPGA, with the exception of the Arcsin function. According to [13] the following simplification can be made,

$$\phi = -\left(\frac{2E[R_I E_Q]}{E[R_I^2] + E[R_Q^2]}\right) \tag{2.22}$$

Kocic et al. justify the removal of the *Arcsin* function by noting that in real world applications phase errors are typically less than 20° [13]. With small phase values, the Arcsin function does not significantly change the estimated value. Therefore, it can be removed to make the algorithm easier to implement in hardware without a substantial loss of precision to the phase estimate.

Once estimates of the error parameters have been calculated, a method is needed for applying these estimates in a way that removes the I and Q imbalance error. In [12], several time domain models for applying the correction coefficients are presented that look quite similar to the feedback network solution proposed by Torkkola [10]. Figure 2.11 depicts the proposed gain and phase error correction block diagram. The gain is corrected first, and then the phase, as was outlined in [12]. Mathematically, it is clear that mixing the coefficients as shown in Figure 2.11 cancels the extra components in the received signal, as shown in Eqs. 2.11 and 2.12.

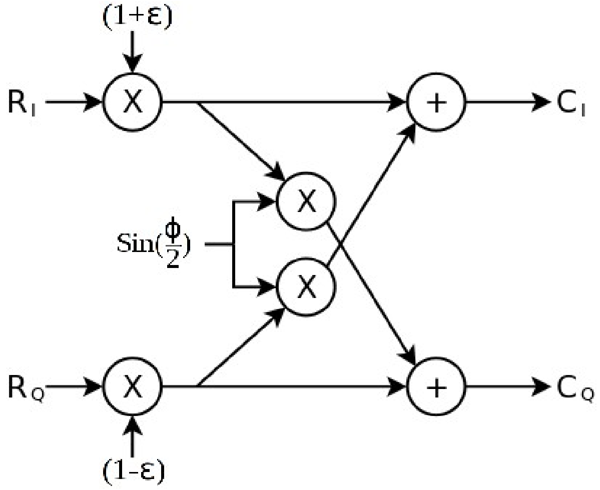

Figure 2.11: I and Q Imbalance Correction Block Diagram.

The block diagram in Figure 2.11 is a valid method of applying the correction coefficients; however, the method requires the use of four multipliers and two adders. The impaired signal was shown to have half the errors applied to the I channel, and the other half of the errors applied to the Q channel. Applying the I and Q errors to only one branch of the signal is well documented [5, 16, 19]. Rather than spreading the gain and phase corrections between the I and Q channels, the corrections can be applied to just one of the channels. This causes the Q branch of the receiver to be equalized to the I branch, and since the desire to make the spectral powers of the I and Q channels equal, this is a valid modification. The proposed modification is depicted in Figure 2.12. The simplification to the hardware correction scheme reduces the dedicated FPGA hardware required to two multipliers and one adder.

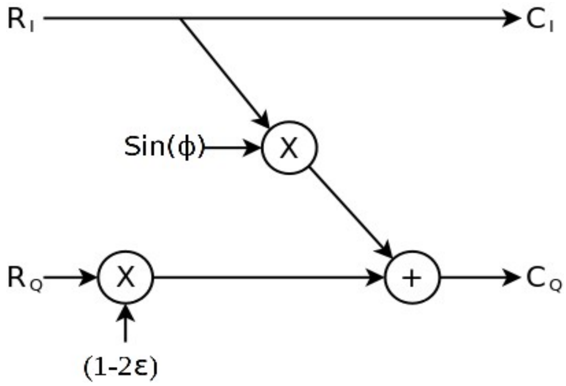

Figure 2.12: Simplified I and Q Imbalance Correction Block Diagram.

2.4.1.5 Other Statistical Correction Schemes

Stat is not the only statistical correction scheme that has been proposed and as a result, several other promising methods will also be presented and discussed. Moseley and Slump presented a novel method that uses only data from the received signal to correct subsequent incoming samples [18]. Figure 2.13 shows the proposed correction architecture. Three estimators are adapted in real time to determine the I and Q imbalance compensation coefficients. Windows of data anywhere from 32 to 256 samples are captured and basic averaging is performed. A Low Pass Filter (LPF) is used with each estimator to smooth the data that is generated by each window of samples. The output of the estimators is then used to generate the correction coefficients. Once the coefficients are computed, their application requires only two multipliers and one addition. This hardware requirement is the same for *Stat*.

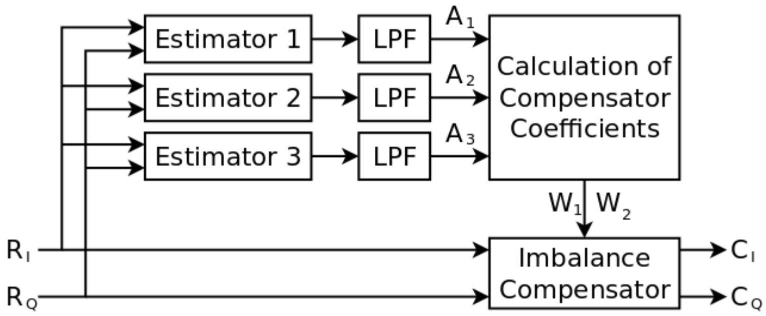

Figure 2.13: Moseley and Slump's I and Q Imbalance Compensation Block Diagram.

The Moseley and Slump method [18] provides an alternative to *Stat*. The performance of both methods cannot be compared based solely on the respective performances because one method contains IRR simulations, and the other method provides BER simulations. Without implementing both methods in Matlab to obtain numerical results, the decision can be made based on the ease of implementation in the Decimator.

The hardware requirements of [18] are only mildly greater than *Stat*. The only major difference is that three additional multipliers are needed for the LPFs. The main drawback comes from the flow of the correction algorithm. *Stat* calculates several running sums in the background and then uses the sums to calculate the error coefficients. Rather than calculating the coefficients in the FPGA, the sums can be passed to the microcontroller for processing. The results can then be passed back to the FPGA for use. Conversely, [18] would require full implementation in the FPGA because of its real time nature. The Mosely and Slump [18] method cannot wait for values to be passed back and forth from the microcontroller. For this reason, *Stat* still appears to be the more promising algorithm for implementation in the Decimator.

Another novel method for removing I and Q imbalances in both the transmitter and receiver was proposed in [29]. Reference was made to *Stat* in the paper but no reasons were given as to why it should not be used were provided. The main benefit to the proposed scheme is that it provides a correction in the transmitter as well as in the receiver. The Bit Error Rate (BER) graphs provided in [29] indicates that the proposed method does not perform as well as *Stat* while correcting 16-QAM and 64-QAM signals under similar conditions. Since the Decimator is a receiver, and no modifications are being made to any transmitters, choosing *Stat* over the method proposed in [29] seems quite justified.

Another novel statistical correction technique was proposed by Anttila et al. [16, 17] which used second order statistics of a signal to obtain correction parameters. The basic assumptions of the methodology are that the received signals are zero-mean, circularly symmetric, mutually uncorrelated, and of equal power. This may seem like quite a number of assumptions to make, but these assumptions apply for the vast majority of modern communication systems. The drawback in the case of the Decimator is that the calculated estimates must be implemented with adaptive filters. Adaptive filters are not feasible in the case of the Decimator. While the Anttila et al. technique may prove invaluable in other receivers, it would probably not be of much value to the Decimator [16, 17].

2.4.2 Data-Aided (DA) Correction Schemes

Data-aided correction schemes are less popular than their counterpart schemes because training sequences must be injected into the signal at the transmitter. Introducing a training sequence requires an increase of complexity in both the transmitter and the receiver due to the increased channel equalization and frequency synchronization. While the performance of DA correction schemes has been shown to be quite good, the usefulness of such correction schemes is still questionable. For these reasons, many practical applications opt for a NDA type solution, rather than for a DA type solution [5]. DA solutions are not feasible in the case of the Decimator because the modulation scheme is unknown to the receiver and altering the transmitter is not an option.

2.4.3 I and Q Imbalance Conclusions

The simplicity of *Stat* and its error improvements in other application make it a prime candidate for integration into the Decimator. It does not require calculations that are beyond the scope of an FPGA and it does not require any knowledge of the incoming signal. The signal is assumed to be symmetric about the origin and to have equal channel powers. For these reasons, *Stat* will be further evaluated as a solution to the I and Q imbalance problems present in the Decimator.

2.5 Windowing

Windowing is a term used to refer to a filter that passes a selected group of samples and sets all others to zero. In contrast with the other types of filters that pass data based on frequency content, a window filter passes data based on its time domain position. The result is a finite sequence of non-zero samples that may be processed further by subsequent DSP algorithms. In the case of the Decimator, the processing that follows the windowing is the FFT.

Windowing has become a common practice in applications that perform the FFT. The reasons for its necessity stem from issues that arise out of the calculation of the FFT. The FFT assumes it is calculating a periodic sequence of data, however, only a small sampling of the incoming data is used to make the calculation. The starting point and ending point of the window frame have to exactly line up to provide a seamless periodic representation of the captured signal. In practice, a set number of samples are used to represent the incoming signal but its starting and ending points do not line up. This results in the energy of the signal being spread across a number of frequency bins, rather than being isolated from each other. This phenomenon is known as spectral leakage.

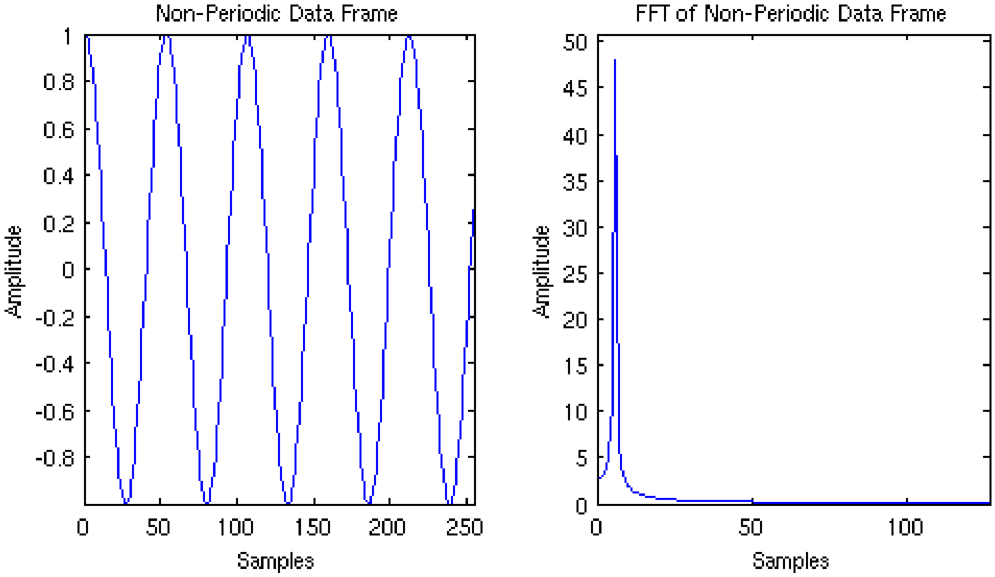

Figure 2.14: Non-periodic frame of data from a periodic sinusoid.

Figure 2.15 shows the 256 point FFT of a simple sinusoidal wave whose starting and ending samples line up to make the data frame periodic. The results in Figure 2.15 can be compared with the result seen in Figure 2.14, where the same sinusoid was processed and the frame of data did not line up. The differences between these two representations are evident. Firstly, the peak FFT bin value is lower because of the spectral leakage. Secondly, spectral leakage causes the base of the FFT spectrum to grow when it should be narrow, as shown in Figure 2.15. Obviously this introduction of error into the FFT calculation should be mitigated. This is why windowing has become an important process in the DSP chain of the Decimator.

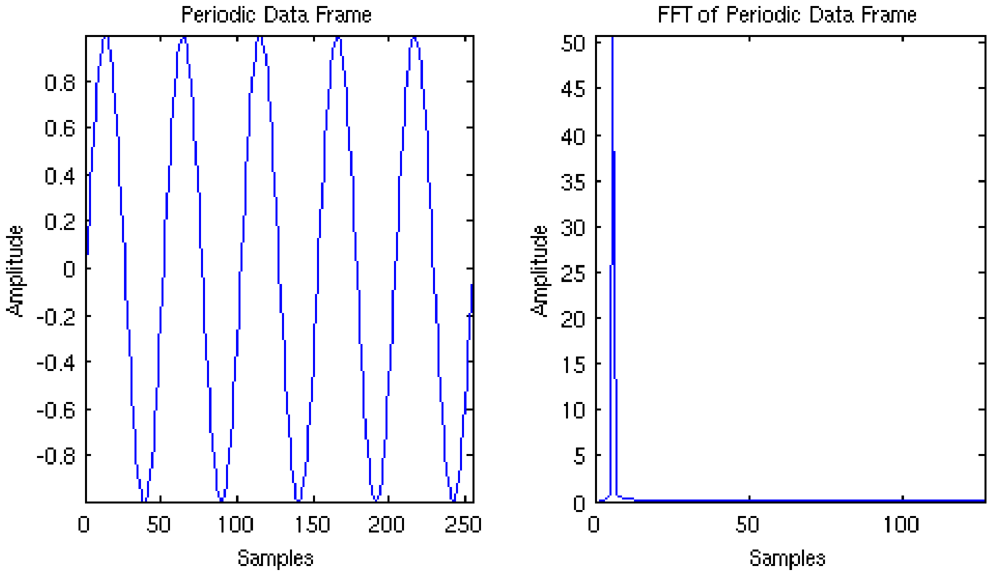

Figure 2.15: Periodic frame of data from a periodic sinusoid.

It should be noted that although windowing was not explicitly performed on the sinusoids from Figure 2.15 and Figure 2.14, the act of limiting the input to the 256 point FFT to 256 samples is itself an implicit application of a rectangular window. The rectangular window is what caused the sharp cut offs at each end of the data frame. There are a wide variety of window designs that round the corners of the data frame to reduce spectral leakage. Figure 2.16 shows a few of the most common windows in the time domain.

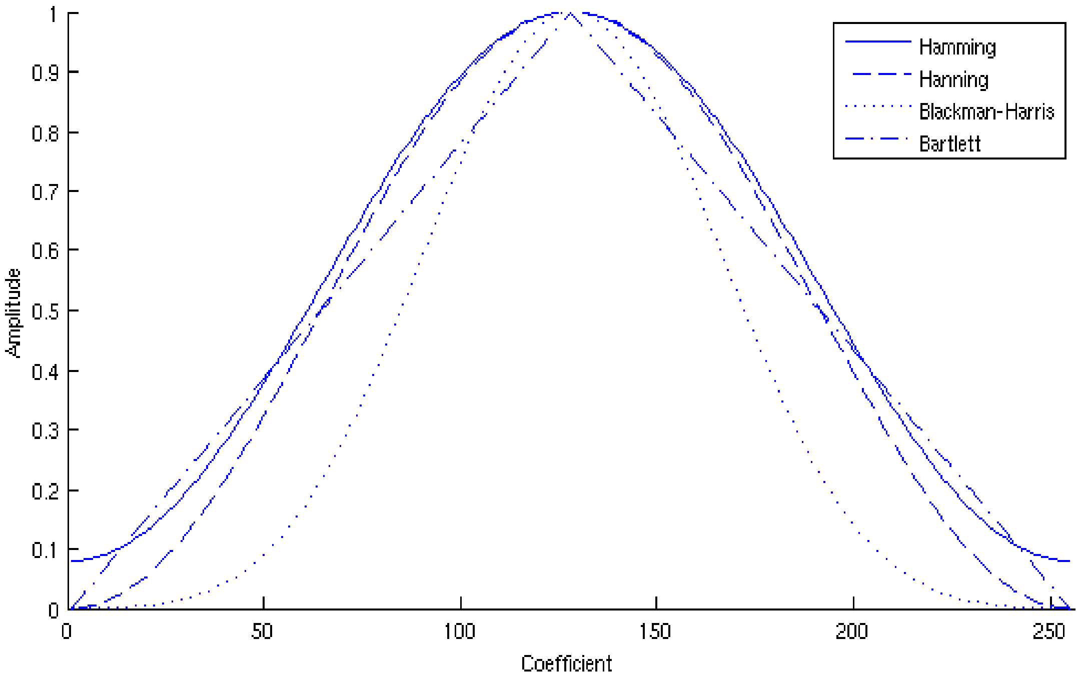

Figure 2.16: Several common window functions.

Taking the example from Figure 2.14 and applying a Hamming window prior to transformation yields improved results. Figure 2.17 shows these improved results, along with the Hamming window coefficients that were applied. Although the new frequency spectrum does not have the spectral leakage that was previously present, the amplitude of the spectrum is significantly lower. The lowered spectral gain is due to the windowing function which removed much of the signal power when it tapered the edges of the data frame. The amount by which the amplitude decreases is known for each type of window and can be corrected by applying a gain factor to the spectrum after the transformation.

Another significant difference can be seen by the overall width of the frequency component. While the frequency component no longer contains the leakage around the base, it is now wider than it was in Figure 2.15. The frequency component is now wider and it has lost some of its spectral resolution. Where two spectral components that are very close to each other may have been distinguishable before the windowing operation, they may overlap each other and cause their spectral components to interfere with each other. Herein lies the trade-off that takes place with the application of a window function: spectral leakage versus frequency resolution.

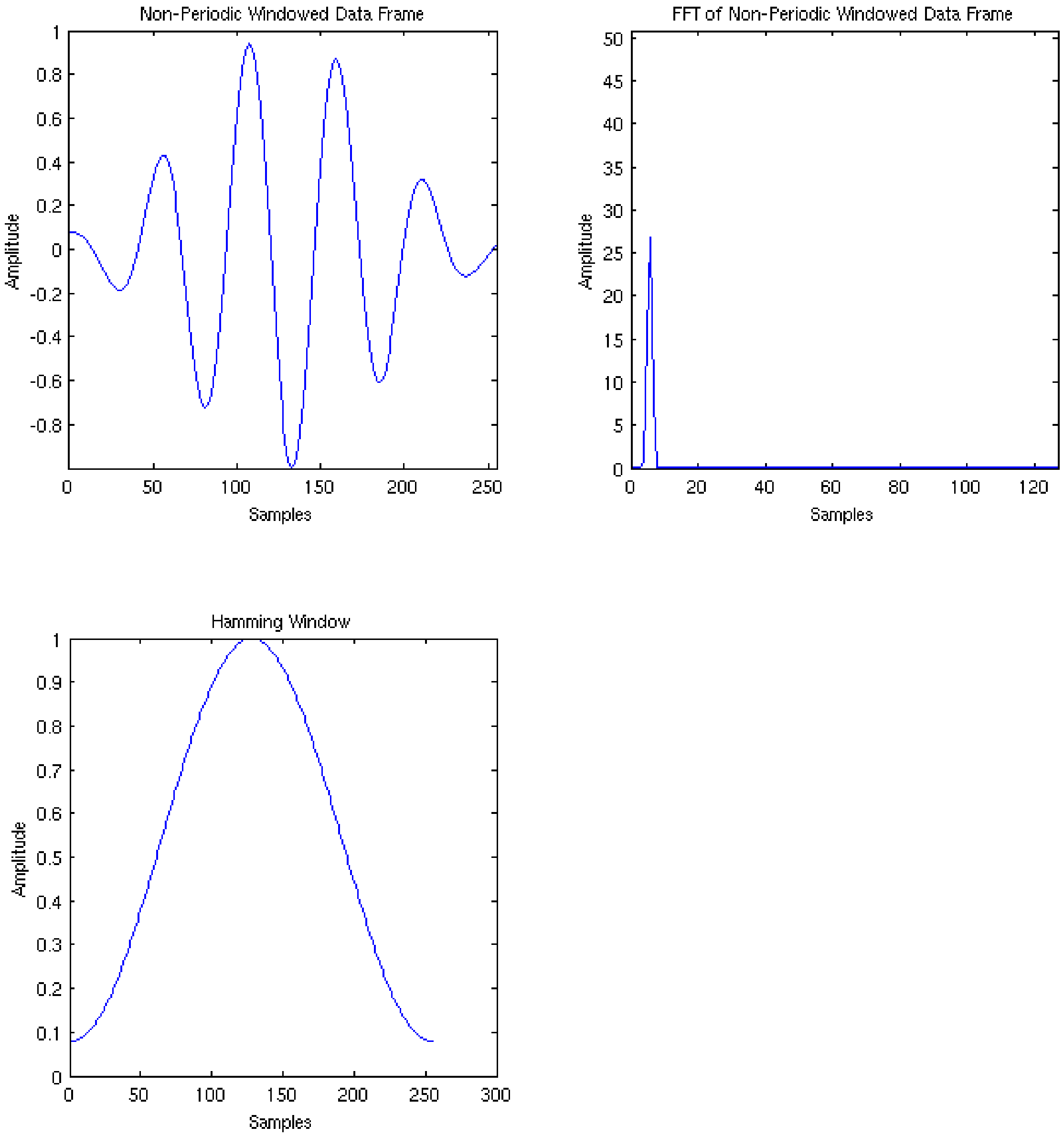

Figure 2.17: Effect of the Hamming window on a periodic signal capture.

2.5.1 Finite Register Length

Finite register lengths degrade the precision of an algorithm in two ways. First, by limiting the precision of calculated values (coefficients), and second, by truncating the results of multiplication operations that overflow. Windowing coefficients are calculated by the Decimator in the microcontroller and may then be passed to the FPGA for storage and use. This process will require rounding operations to take place such that the 32-bit coefficients can be represented by fewer bits in the FPGA. This will be the first loss of precision in the window migration process. The second source of error will not be an issue as it was in the FFT algorithm. Each incoming sample is multiplied by its corresponding window coefficient, but each coefficient is less than 1. There will be no errors introduced by overflows in the windowing function. Only one of the two main sources of error are applicable to the windowing operation. Migration of the windowing to the FPGA should not induce as much error in this system as migration of the FFT algorithm.

2.6 Fast Fourier Transform (FFT)

Advances in mathematics have brought about new ways of viewing data. Various transforms such as the Hilbert, Cosine, and Fourier Transforms have become commonplace in a variety of signal processing applications. The Fourier Transform changes a signal so that rather than viewing a signal as an amplitude versus time function, the signal may be viewed as an amplitude versus frequency function. While the Fourier Transform is a theoretical transformation, the Fast Fourier Transform (FFT) is its practical realization [1, 2, 3]. The FFT has become one of the most important transforms in signal processing applications and much work has been done to implement the FFT on a variety of hardware platforms.

The FFT is currently performed in the ARM microcontroller on board the Decimator. However, there are several reasons why migrating the FFT to the FPGA is desired. The primary reason relates to speed; the microcontroller is much slower at processing data than the FPGA. The advantage of the microcontroller is that it has a 32-bit floating point mathematical operator, whereas the FPGA is inherently fixed point. Keeping as much of the signal processing chain on the FPGA makes for a much more maintainable product. As a side benefit, a logic core that performs the FFT adequately on the Decimator may be an asset to other related projects.

2.6.1 FFT Background

The Discrete Fourier Transform (DFT) provides a way for digital systems to realize the Fourier Transform of a function. However, it is a time consuming transform that is not practical in most systems. Exploiting some of the key properties of the DFT, such as periodicity and symmetry, a variety of more efficient algorithms have been developed to make the implementation of the DFT in hardware a practical reality. These more efficient algorithms fall under the blanketed term “Fast Fourier Transform” because of increased speed with which the transform is calculated [1, 2].

The general DFT equation can be written as,

$$X[k]=\sum_{n=0}^{N-1} x[n]W_N^{kn} \tag{2.23}$$

where X[k] represents the frequency bins found in the sequence x[n]. N represents the number of samples in the given frame of data, or window, and W_N^{kn} represents the complex exponential $e^{-j(2\pi/N)kn}$. By multiplying discrete samples by the complex vector $e^{-j(2\pi/N)kn}$, the frequency content of the signal at various angles can be summed and transformed into a set of frequency bins [1, 2]. However, the way in which the DFT algorithm is implemented in hardware is a topic of much study, and a number of methods have been proposed to reduce its computational complexity.

2.6.1.1 Decimation-in-Time (DIT) Algorithms

There are two main types of FFT algorithms; decimation-in-time (DIT) and decimation-in-frequency (DIF) algorithms. The DIT algorithms break an incoming frame of samples into successively smaller sub-sequences before performing the transform to only a small number of samples. The rest of the transform coefficients can then be deduced using the periodicity and symmetry principles of the Fourier Transform [1]. Figure 2.18 depicts the flow of a DIT algorithm. While this example flow chart shows an 8-point DIT FFT, the method can be expanded or contracted to apply to all FFT point sizes that are factors of 2^N. The decomposition scheme leads to a reduction in the number of multiplication operations needed by more than a factor of 100 [1, 2].

The values of W_N^{kn}, referred to as "twiddle factors," need to be calculated for each FFT point size. They are reusable, however, so they only need to be calculated once per point size. As is seen in Figure 2.18, some of the twiddle factors may be simplified to either a "1" or "-1". This is a valid simplification since the angular frequencies at even multiples of π are $W_N^{N/2}=e^{-j(2\pi/N)N/2}=e^{-j\pi}=-1$ and $W_N^0=e^{-j0}=1$.

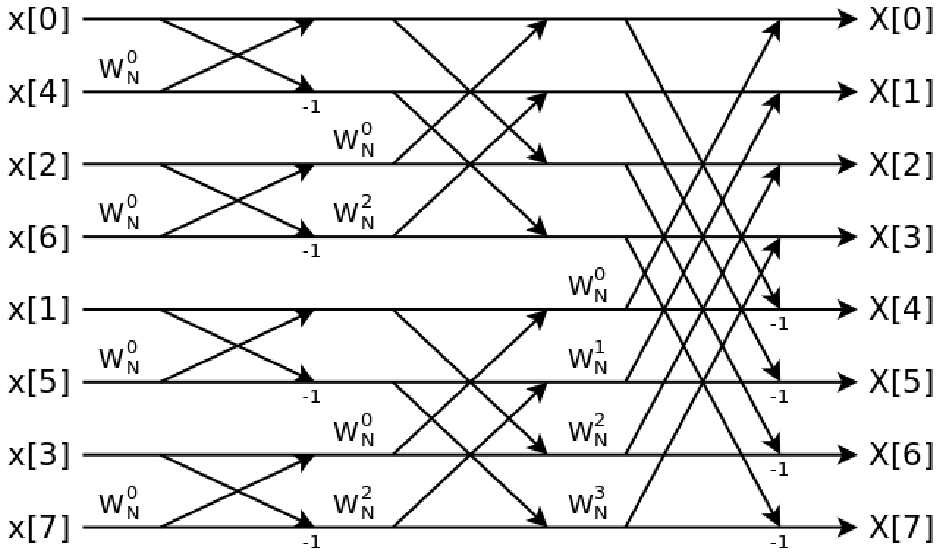

Figure 2.18: Flow graph of an 8-point DIT decomposition.

2.6.1.2 Decimation-in-Frequency (DIF) Algorithms

The most notable difference with the DIF algorithms is that it takes the input data in its natural order and performs the FFTs starting with the 2^N (for radix-2) FFT first. This is in contrast to the DIT algorithm which rearranges the input data window in order to perform the smallest FFT calculation first before moving up to the higher values. Both algorithms have strengths and weaknesses when compared to one another, so the right algorithm needs to be chosen to fit the system it is being implemented in. Figure 2.19 depicts the flow graph of a basic DIF implementation.

Comparing Figure 2.18 and Figure 2.19, one design consideration is immediately apparent. A choice between ordered inputs, or ordered outputs must be made. The DIT FFT must buffer the input data in order to rearrange it and apply the butterfly calculations. On the other hand, the DIF takes data in order and generates Fourier coefficients that are out of order. Because of this, the data at the output must either be buffered and rearranged, or the system using the generated Fourier coefficients must know that they are not in their natural order (i.e. the coefficients are not ordered 0, 1,2,etc.).

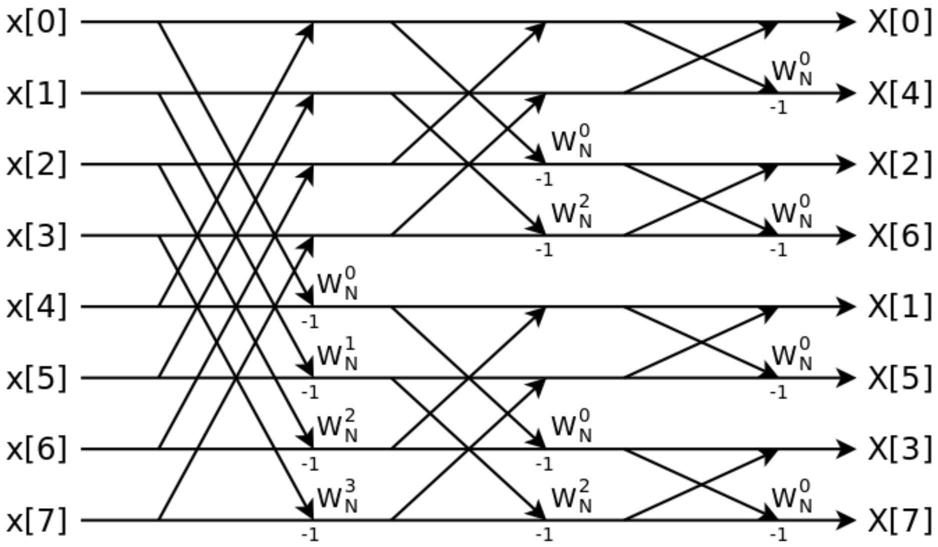

Figure 2.19: Flow Graph of an 8-point DIF decomposition.

2.6.1.3 FFT Radix Size

In the previous examples of DIT and DIF FFTs, a radix size of $N = 2$ has been assumed. That is, the FFT length is broken down by a factor of 2 at each step of the decomposition. Radix-2, where $N = 2$, is one of the most common FFT settings because it greatly reduces the number of multiplications needed. The down side to decomposing the FFT this much is that it takes longer to calculate. When implemented in hardware, this equates to an increase in algorithm latency. Rather than decomposing the calculations down to a minimum of 2, as in the radix-2 calculation, setting $N = 4$ simply causes the minimum FFT calculation to include 4 samples. This is known as radix-4, and it offers an alternative to radix-2 in that it performs the same FFT calculation in less time but with more multiplications. Unfortunately, it is less robust than radix-2 because it can only calculate FFT lengths that are a power of 4.

2.6.2 Finite Register Lengths

The most important issue when discussing the migration from floating point hardware to a fixed point platform is the loss of data that can be attributed to truncation and rounding. One way these errors creep into the FFT calculation is through twiddle factor multiplications. Twiddle factors must be stored in memory with a fixed number of bits. The number of bits used determines how precisely the actual twiddle factor value is represented. Bit growth due to the multiplications that take place in each butterfly of the FFT is the other dominant source of error due to finite register lengths. Analyzing these two issues varies based on the FFT architecture and input signal model.

FFT algorithms contain a large number of multiplications, and twiddle factor quantization errors propagate with each multiplication. The way quantization noise is manifested in twiddle factors depends on the implementation of the complex multiplier (i.e. the architecture that is instantiated). The Chang and Nguyen model is based on the Radix-2 FFT, which limits their results to be applicable only to Radix-2 FFTs [27]. Comprehensively evaluating the effects of finite register lengths with respect to the Xilinx FFT core is a fairly in-depth task. Oppenheim and Weinstein believe the twiddle factor quantization errors are not a major source of error [22]. Rather, it was concluded that the quantization error varies directly with *N*, the number of bits used, which means doubling the number of bits used to represent the twiddle factors would produce only a small improvement in the noise-to-signal ratio of the FFT. It is important to note that more experimental verification is required since their hypothesis is based on an equation meant to give a rough estimate of quantization error.

Each butterfly in a Radix-2 FFT has the potential to increase the number of bits required to represent the result by a factor of two [1]. In a Radix-4 FFT, an increase up to a factor of four is possible. Floating point arithmetic is able to handle these bit growths because of its superior dynamic range performance [22], however, fixed point arithmetic requires some form of intervention to keep the result manageable. The Xillinx FFT core comes with several configurable options to address this problem.

2.6.2.1 Full Precision Unscaled

The Full Precision Unscaled settings introduces the least noise into the system. The number of bits at the output is determined by the worst case scenario formula, as seen in Equation (2.24).

$$\mathit{input\ width} + \log_2(\mathit{FFT\ length}) + 1 \tag{2.24}$$

Therefore, 12-bit input samples will yield 22-bit FFT coefficients. While this setting does not introduce any truncation or rounding noise to the data, the data will not be usable by the rest of the system. As a result, the output will most likely require scaling to bring it back to a usable size. This is an elementary way of dealing with the bit growth problem, and it will introduce a substantial amount of error into the system.

2.6.2.2 Scaled Fixed Point

The next built-in function to deal with bit growth after each butterfly calculation is called Scaled Fixed Point. Rather than scaling the result at the output of the FFT, the scaled fixed point setting scales by a user-defined value at each butterfly calculation. This technique is quite common, and has been shown to be superior to having one large scaling factor at the input of the FFT [1, 22]. The down side to this procedure is that scaling is not necessarily required with each butterfly, but it is still applied anyway. Data that has a higher average value and comes close to saturation quite frequently will not notice much of an issue with this solution. Data that is close to saturation requires scaling at most butterfly operations anyway. Conversely, data that has a lower amplitude will become much noisier with this setting since scaling is applied unnecessarily at most FFT stages.

2.6.2.3 Block Floating Point (BFP)

The final built-in FFT core setting implements Block Floating Point (BFP) arithmetic. BFP may be considered a special case of the floating point format where non-overlapping groups of data are joined together by a common scaling factor. The scaling factor acts as the mantissa in a floating point number, except the scaling factor is chosen to represent the largest samples within the group [23]. When the BFP arithmetic option is selected, the output of each butterfly is checked to determine whether an overflow has occurred. If an overflow has occurred, a scaling factor of two (for Radix-2) or four (for Radix-4) is used to bring the data back to the desired number of bits. The number of overflows and the stage at which they occur affect the SNR. The setting of this variable can greatly vary the SNR of the input data [1].

2.6.3 Dynamic Range

Dynamic range refers to the smallest and largest values that can be represented of a given variable. As it pertains to spectral analysis, dynamic range determines the ability of the FFT to distinguish between small and large spectral peaks. Measuring the dynamic range of an FFT system is not a trivial task. This is primarily because dynamic range has no specific definition as it pertains to FFT systems [24]. There are several common methods of measuring the dynamic range of an FFT system that are widely accepted. The first method is to find the ratio between a full scale sinusoid and the noise floor. This is called a “two-toned” measurement and it provides an accurate theoretical measurement of dynamic range. The “two-toned” measurement does not take into consideration the fact that real-world signals contain numerous sinusoids, so it fails to provide an accurate practical measurement of dynamic range [25].

The second method for determining the dynamic range of an FFT system is called the “noise slot test” [21, 24] White Gaussian noise is created and passed through either a notch filter or a bandpass filter. The remaining data is then scaled and quantified to use the full range available given the number of bits in the system. This signal is then passed to the input of the FFT system. The difference between the average signal power and the average noise power is the dynamic range of the FFT system. There is a third method of determining dynamic range called the “mean-squared error technique” but it can be shown to be equivalent to the “noise slot test” [24].

Xilinx simulated a number of “noise slot tests” in Matlab and documented the results [21]. Figure 2.20 shows an overview of the results that were obtained. The slope of the graph is 6.06 dB/bit, which means that each additional bit used to represent the input and output increases the dynamic range of the FFT by 6.06 dB. The change of about 6 dB of dynamic range per bit is not an accurate representation of actual dynamic range, but of theoretical range [24, 25]. In reality, there are other factors that cause noise and prevent the 6 dB per bit rule to be followed explicitly.

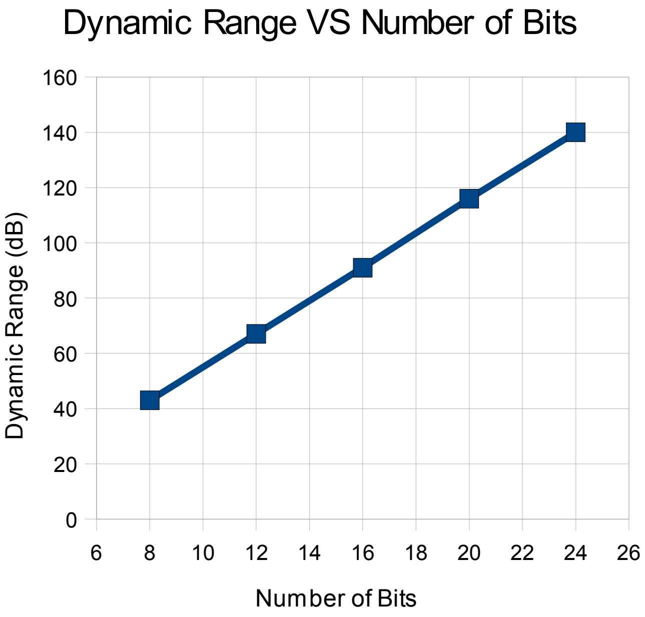

Figure 2.20: Xilinx dynamic range results.

Xilinx documented several other “noise slot test” result that are of interest to this thesis. Xilinx calculated the dynamic ranges of the full precision unscaled arithmetic, the scaled (1/N) arithmetic, and the BFP arithmetic using a bit accurate Matlab model of the Xilinx FFT Core [21]. All simulations were run using 1024 point, Radix-4 Burst I/O transforms with 16-bit input data, 16-bit phase factors, and convergent rounding. The full precision unscaled simulation yielded a benchmark result of 91 dB. The BFP simulation calculated a dynamic range of 73 dB. The smallest dynamic range was the scaled arithmetic simulation, which found the dynamic range to be 64 dB. Therefore, moving from full precision unscaled to BFP lead to a loss of 18 dB. A further loss of 9 dB was sustained by going from the BFP to the scaled arithmetic.

A more practical look at the effective dynamic range of an FFT system was performed in [26]. Other factors that affect the dynamic range of an FFT system are the FFT algorithm that is used (i.e. DIT or DIF), the FFT length, and the window filter that is used. Several simulations were run using a variety of settings, but one simulation in particular had similar settings to those used in [21]. Simulation 3 in [26] was run using the 1024 point DIT FFT, a rectangular window, 15-bit input data, 15-bit phase factors, and 15-bit output samples. The only notable difference from one of the Xilinx simulations is the use of 15-bits as opposed to 16-bits. The Dynamic range was found to be 70 dB in [26], which is very close to the 73 dB value found by Xilinx. The 3 dB difference may be attributed to the slight differences between the simulations.

CHAPTER 3 : RESEARCH PROGRAM / METHODOLOGY

3.1 Introduction

The research program is designed to study the effectiveness of the proposed changes to the Decimator. A methodology for simulating and verifying the chosen I and Q imbalance correction scheme will be outlined. Similarly, a plan for the algorithm migrations will be outlined that can be implemented in VHDL and simulated using the Xilinx ISE Simulator. The steps taken to study each of the issues in this thesis will be clearly defined. The results of the described simulations will be presented and discussed in Chapter 4.

3.2 I and Q Imbalance

Chapter 2 presented a variety of solutions to the I and Q imbalance problem. There are only a few that can be practically implemented in the Decimator. Statistical methods of correction showed promising results while using minimal resources and for these reasons, a statistical method was studied. The statistical method studied is called *Stat*, as discussed in Chapter 2. *Stat* will be implemented in VHDL and simulated using Xilinx ISE to obtain bit accurate data and hardware usage estimates. *Stat* will also be implemented in Matlab to give the bit accurate VHDL simulations 32-bit floating point reference results. The method of applying the correction coefficients will be implemented as outlined in Figure 2.12.

3.2.1 Stat Design Overview

The overall design of *Stat* was outlined in Chapter 2, but a more detailed discussion regarding its implementation in VHDL is presented herein. The discussion describes the source of the simulation results. Figure 3.1 shows the VHDL system level design of *Stat*.

The first point to note is that the actual calculation of the correction coefficients is done outside the FPGA in the microcontroller. The reasons for using this approach were outlined in Chapter 2. The low level calculations required to find the correction coefficients (i.e., the sum of squares and sum of products) are performed in the FPGA. These calculations are kept in the FPGA because their hardware requirements are quite low and it is possible to keep up with the speed of the incoming data (65 MHz). Sending all the incoming data to the microcontroller for the sum of squares operations would more than likely require buffering. Exporting the pre-summed quotients should work well because it utilizes the speed of the FPGA and the precision of the microcontroller to find the correction coefficients.

The microcontroller controls the I and Q imbalance correction block. The *Enable_Corr* signal instructs the block whether to correct the incoming data or to simply

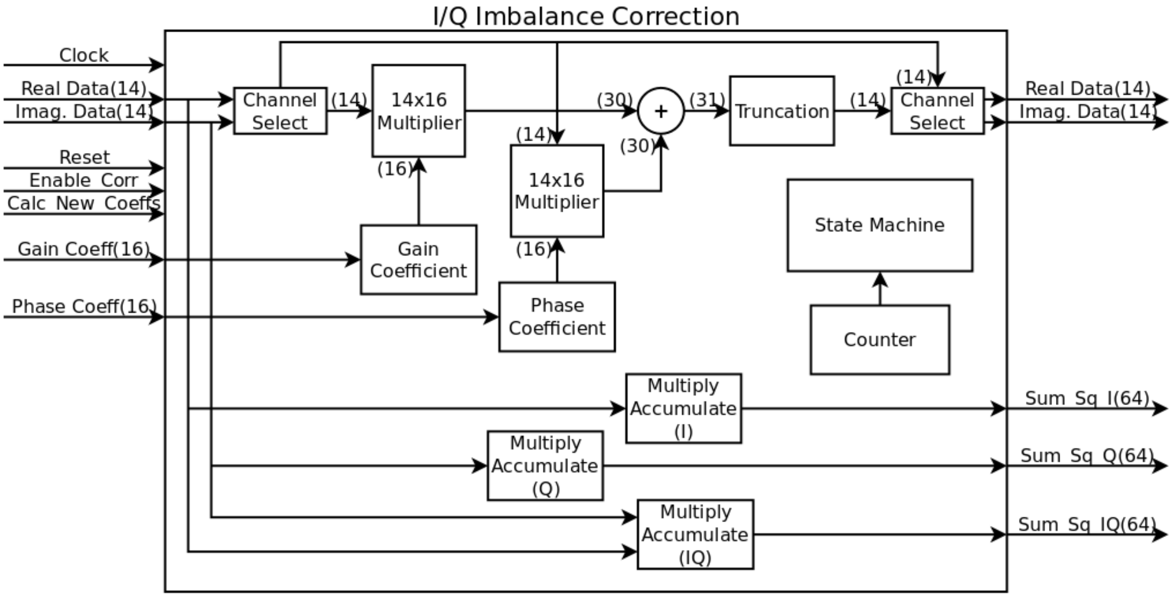

Figure 3.1: System level design of I/Q imbalance correction implementation

pass it through to the output without correction. The *Calc_New_Coeffs* signal begins the process of calculating data for the microcontroller to use in formulating new correction coefficients. Once the data has been summed, it is passed to the microcontroller for further calculations. The gain coefficient is calculated first and is updated in the FPGA. Using that new gain coefficient, the incoming data is gain-corrected and the phase correction coefficient is calculated. Once enough data has been summed, the phase data is sent to the microcontroller for coefficient calculation. The phase coefficient is then passed back to the FPGA to correct phase errors in the incoming data.

A state machine will control the calculation of the gain coefficient and subsequently the phase coefficient to ensure that each step in finding the new coefficients is handled sequentially. The sequencing relies on the incoming control signals from the microcontroller to begin the process for finding new coefficients as well as advancing the process through its cycle. The counter is also a critical part of advancing the state machine. The counter ensures the proper number of samples are summed as well as letting the state machine know when to pass the collected data to the microcontroller.

The design displayed in Figure 3.1 utilizes five 18 x 18 multipliers. As was discussed in Chapter 2, correcting only one of the channels to make it match the other channel eliminates the need for two of the multipliers. The *Channel_Select* modules are switches that pass the real data to the gain correction block when the gain correction coefficient is negative, and passes the imaginary data to the gain correction block when the gain correction coefficient is positive. The channel that is not passed to the correction blocks is passed to the phase correction block. Simply stated, the channel that has more gain is multiplied by a correction coefficient that is between -1 and 1 to reduce its amplitude to that of the other channel. The *Channel_Select* facilitates this process.

3.2.2 *Stat* Sources of Error

The implementations of *Stat* in VHDL and Matlab will help locate the sources of error that are associated with the correction scheme. The *Stat* implementations will also help determine the error correction that it provides. A number of simulations will be run that focus on specific areas of *Stat*. By testing individual sections of *Stat*, it should be possible to pinpoint any deficiencies in the correction scheme. Figure 3.1 shows that the error introduced in the correction scheme come from quantization of the gain and phase coefficients, and from truncating the result of the correction back to 14-bits. The focus of this thesis will be on the overall performance of *Stat* and the quantity of error associated with each of its sources of error.

3.2.2.1 Coefficient Estimate Accuracy

There are two key questions that need to be answered regarding the ability of *Stat* to estimate correction coefficients; namely, 1) how accurate are the estimates? and 2) how many samples are necessary for sufficiently accurate estimates? The remainder of this section will discuss these two issues and describe methodologies for obtaining quantifiable results that help answer these questions.

There is no closed-form expression for determining the correction coefficients. The correction coefficients must be statistically determined from incoming data. Estimating the correction coefficients introduces error into the correction scheme because the number of samples used will affect the result of a statistical estimate. The number of data samples used to calculate the correction coefficients must be analyzed to determine whether there is an optimal number of samples that should be collected for subsequent coefficient estimations. Searching for an optimal number of samples will also show whether or not the accuracy of the correction coefficients converges as the number of samples gets extremely large.

Figure 3.2 outlines the following data generation process description. The first step in determining the affect the number of samples has on the accuracy of the estimate is to generate some random data. Generating random data yields a full spectrum of frequencies. The simulations will involve modulation and demodulation of the generated data and both these operations include anti-aliasing Low Pass Filters (LPFs). LPFs remove the frequency components of the originally generated data that are above ½ the sampling frequency. This cutoff is called the Nyquist Frequency [1, 2]. Filtering the randomly generated data with a 5th order Butterworth LPF that has a cutoff frequency of ¼ the sampling frequency will eliminate this problem. The random samples are generated with a mean of zero, however, there is always a possibility that the generated data will have a slight DC offset. Any minor DC offset that may be present in the generated data will be removed to isolate the sources of error in the simulation. The data is ready for simulation once these factors have been taken into consideration.

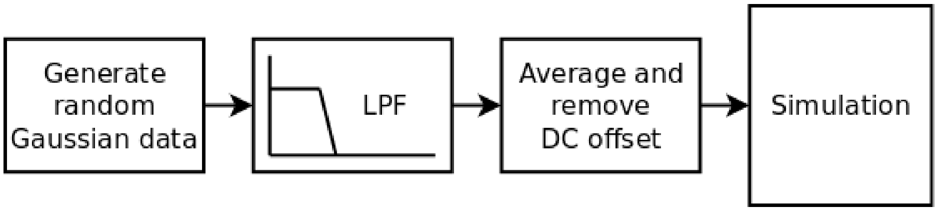

Figure 3.2: Data generation for coefficient estimate accuracy simulation.

Either a filter must be used or a number of simulations need to be averaged to smooth the results of the simulation. Averaging a number of simulations is a better way of performing the smoothing because it reduces the statistical anomalies that may be present in any one generated group of samples. The coefficient estimation errors are averaged over 10 different trials, each using unique randomly generated data. There are 20 different sample values used to calculate the gain and phase coefficients which range linearly from 50 to 418876. A range of samples of this size should yield the desired convergence information. The gain and phase errors that are introduced are 0.608 dB and 4° respectively. These are above-average I and Q imbalances than are typically found in Decimators.

The appropriate number of samples that must be used to obtain an accurate coefficient estimate should be clear once these simulations have been run. The next step will be to choose a number of samples to use that is far larger than the determined minimum number of samples to find how accurate the coefficient estimations become as the number of samples effectively approaches infinity. This result should show what the estimators are capable of and how they can operate under ideal conditions.

3.2.2.2 32-Bit Floating Point Stat Performance

The performance of *Stat* in the 32-bit floating point environment of Matlab will determine its ability to correct unknown errors in the actual Decimator data. The captured Decimator data will come directly from the output of the Analog to Digital Converter (ADC), which bypasses the DSP chain. The input signals for the Decimator have been generated using a variety of Arbitrary Wave Generators (AWG) that are capable of producing signals up to at least 2.15GHz. The Agilent E4438C, Agilent 8267D, and Agilent N5181A are all examples of such AWGs. The generated input signal was split into two identical components in order to have the I and Q channels of the Decimator identical.

By having identical simulated received signals, any differences between the two post processed signals from the receiver can be classified as front end receiver error. The amount of error introduced will be evaluated and *Stat* will be used to correct the data. Correcting the data will show the effectiveness of *Stat* without including fixed point precision error in the results.

A number of data captures have been obtained from the Decimator. The captures will be used to verify the Matlab simulation of *Stat*. The captures to be analyzed are discussed below to show the ability of *Stat* to handle some standard signals that are routinely encountered by the Decimator. The captures contain a large number of samples but only a finite number will be used for *Stat* coefficient estimation. The number of samples that should be used to obtain realistic results will be determined prior to running this simulation.

3.2.2.3 Fixed Point Precision Affect on Stat

Another important factor to consider is the amount of error that will be introduced into the *Stat* correction scheme from the fixed-point precision arithmetic in the FPGA. The implementation of *Stat* in the FPGA will be functionally identical to its implementation in Matlab. The only difference is that the fixed point hardware is used to run the calculations and correction. This simulation will illustrate the error that is introduced into the correction scheme from the truncation and rounding associated with fixed point mathematics. The simulation will also serve as a verification that *Stat* has been successfully implemented in VHDL.

3.2.2.4 Arcsin Affect on Phase Estimates

As was discussed in Section 2.4.1.4, [13] concluded that removing the Arcsin function from the phase coefficient estimate, (as seen in Equation (2.22)), would not contribute greatly to the error in the estimate for phase imbalances under 20°. The reason 20° was chosen as the maximum phase the estimator equation could handle is unclear since no data is presented to qualify this claim. A simulation will be performed to determine the affect the Arcsin function has on the precision of the phase coefficient estimation. The results will indicate whether the reduction in hardware from the Arcsin function is worth the loss in the coefficient precision. The results will also indicate whether the 20° error point recommended in [13] was identified for an obvious reason.

The simulation will be set up in a manner similar to the one described in Section 3.2.2.1. Phase errors between 0° and 30° will be introduced in five randomly generated signals through a demodulator. The five randomly generated sets of data will help smooth statistical anomalies that may be present in any one of the randomly generated sets of data. Each of the five sets of data will be run through two *Stat* functions; one using the phase estimate seen in Equation (2.21) that contains the Arcsin function, and one using the phase estimate in Equation (2.22) that does not have the Arcsin function.

3.2.3 *Stat* Resource Usage

The hardware resources required by *Stat* will be individually evaluated and discussed once the VHDL implementation of *Stat* has been synthesized. The required resources will be apparent and the available resources on the Spartan 3 1500 are known, so concluding whether the VHDL implementation of *Stat* will work in the FPGA should be straight forward. The current Decimator implementation has also been previously synthesized and the results of the synthesis will be compared with the added resources that S*tat* will require.

3.3 Windowing

A window filter will be designed to operate in the FPGA of the Decimator based on the currently implemented window filter that operates in the microcontroller. A similarly functioning window filter will also be designed in Matlab. Bit accurate simulations using Xilinx ISE will then be run to obtain results that ought to be identical to those that would be obtained if the algorithm were running on the FPGA in the Decimator. The results of the VHDL simulation can then be compared to those of the Matlab simulation. The differences between the two simulations will characterize the affects of migrating the window filter from the microcontroller to the FPGA.

The design of the window filter is based on the original design that was implemented in the microcontroller of the FPGA in C. The code required to implement the windowing

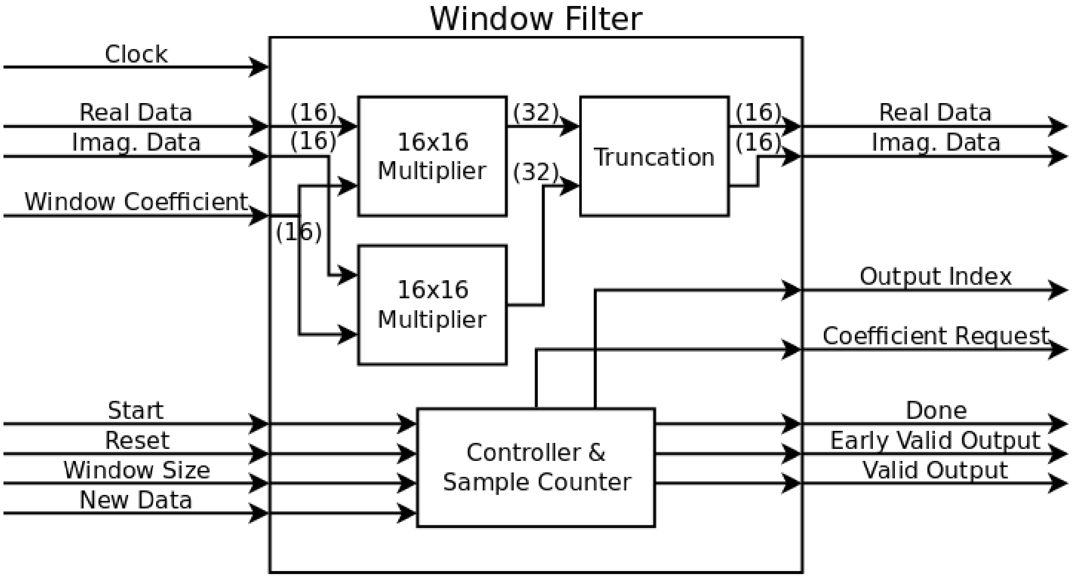

Figure 3.3: Window filter VHDL implementation block diagram.

in VHDL is substantially more complex than the original C code due to the nature of VHDL. Figure 3.3 displays a system-level overview of the implemented VHDL window filter.

Data enters the window filter as 16-bit fixed point real and imaginary samples. The windowing coefficients will be generated in the microcontroller, truncated to 16-bits, and then passed to the shared RAM that the microcontroller and the FPGA both have access to. The Controller & Sample Counter in the window filter keeps track of the number of samples that have been received, which coefficient is required next, and when the filter has completed its cycle. The incoming real and imaginary samples are multiplied by the recalled coefficients stored in the shared RAM. The two 16-bit multipliers yield 32-bit fixed point results which are then truncated to 16-bit values and passed out of the block.

3.4 Fast Fourier Transform (FFT)

Since the Decimator uses a Xilinx Spartan-3 FPGA as one of its main computational units, using a Xilinx FFT core would appear to be an obvious solution to pursue. The various features of the Xilinx FFT core, along with a discussion of why it would most likely function satisfactorily if used in the Decimator. In this section, the Xilinx FFT core will be reviewed, and a number of simulations will be devised to test various features of the core. The results of these simulations should show whether the core does in fact meet the requirements of the Decimator, or whether another solution will be required.

3.4.1 Xilinx FFT Core

There are four basic architectures available for the Xilinx FFT core: Pipelined streaming I/O; Radix-4 burst I/O; Radix-2 burst I/O; and Radix-2 Lite burst I/O [21]. Each architecture calculates the FFT differently, offering a tradeoff between resource usage and throughput. The details of the architectures are not as important to this thesis as their resulting performances. These four basic architectures will be discussed along with pertinent design considerations.

Figure 3.4 is a graph provided by Xilinx to illustrate the relationship between the four architectures with respect to their performances. By far the most powerful architecture is the Pipelined Streaming I/O FFT, but its resource usage is also the highest. There is no added latency associated with loading and unloading the data frame with the Streaming I/O architecture because it is capable of processing data in real time. It is also the only architecture to use the DIF algorithm, but the output can either be in natural order or bit reversed order. The processing engines used are Radix-2, so the point sizes supported range from 8 to 65536 at multiples of 2^N.

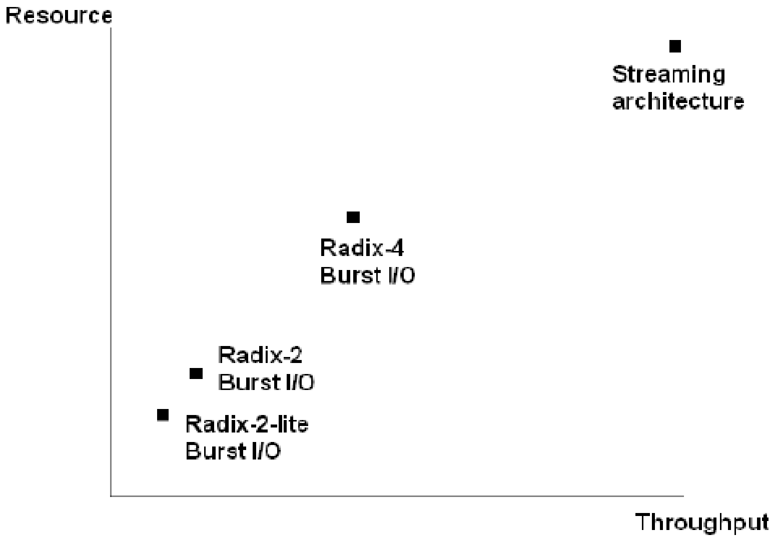

Figure 3.4: Resource usage V.S. throughput for Xilinx architecture options [21].

Radix-4 Burst I/O, along with both the Radix-2 and Radix-2 Lite architectures, use the DIT FFT algorithm to process data. These architectures are labeled as "burst" architectures because loading, calculating, and unloading the data happens separately. Incoming data cannot be processed in real time as in the case of the Pipelined Streaming architecture. The exception to this rule is that data load and unload operations can be overlapped when the output is in bit reversed order. The processing engine is Radix-4 based which allows for computation of point sizes that are multiples of 4^N. A Radix-2 engine is added to the architecture and this allows for point sizes of 64 to 65536 to be processed by the Radix-4 Burst architecture in multiples of 2^N.

The Radix-2 Burst I/O architecture and Radix-2 Lite Burst I/O both use one Radix-2 butterfly engine and both support point sizes from 8 to 65536. The difference in performance is a result of the Lite architecture only having one input to the butterfly. The real and imaginary samples are alternately fed into the butterfly of the Lite architecture, which adds an extra clock cycle to complete each calculation.

The first consideration when implementing the FFT core is determining which architecture should be used. While the Pipelined, Streaming I/O architecture is not necessary and is most likely too large for the Spartan-3 FPGA, one of the Radix-4, Radix-2, or the Radix-2 Lite architectures may operate satisfactorily and use an acceptable number of resources. Each of the four architectures will be instantiated and their hardware requirements and performances will be evaluated to determine the optimal solution.

One of the key features of the Xilinx FFT core is its ability to perform BFP arithmetic. A Radix-2 1024-point FFT can grow by 10 bits, which is a factor of 1024 times the size of the original data. BFP arithmetic allows the data to be scaled only when there is an overflow from one of the butterfly operations, which prevents truncation from occurring unnecessarily. Unnecessary truncation is a major source of error with fixed-point pre-determined scheduled scaling. BFP also keeps track of the number of times the data is scaled so the output can be re-scaled once the output of the FFT has been passed to the microcontroller. Simulations will be run to determine the feasibility of implementing the FFT using BFP arithmetic. Simulations will also determine whether there is in fact a reduction in truncation error.

The performance improvement of BFP will be determined by implementing the FFT core using a scheduled scaling routine. After each butterfly, the data will be scaled by 1-bit to ensure there are no overflows. The spectrum will then be compared to both the Matlab simulation and the block floating point VHDL simulation. Measures of accuracy will involve spectral peak error and Mean Squared Error (MSE) using the Matlab simulation output as the benchmark. The loss of dynamic range that occurs with scaled fixed point arithmetic may become apparent through these simulations. If it is, dynamic range measurement will also be taken to quantify the results of the system.

3.5 Windowing and FFT VHDL Simulation

Testing the window filter and the FFT in an independent manner should help isolate the exact locations of introduced errors. Independently testing the windowing and FFT modules may not be necessary because they will never function independently from one another in the Decimator. Testing the two systems together will yield the overall performance of the two blocks as they will be operating when implemented within the Decimator.

There will be two measures of accuracy to determine the performance of the windowing filter and the FFT algorithm as they operate together in the FPGA. The first measure is the spectral peak error. The signal that will be analyzed is a complex wave with a carrier at +15 MHz. This means there will be one large spectral peak carrier signal. The ratio between the error of the VHDL and Matlab implementations and the power of the transmitted signal will show the amount of error at the peak of the transmitted signal. This is an important measurement since many of the measurements taken from a spectrum analyzer involve spectral peak values. The second measure of accuracy is mean error, where the average difference between the VHDL implementation and the Matlab implementation is calculated. The ratio between the power of the transmitted signal and the average error of the VHDL implementation is then calculated to present the results in Decibels. These two measurements will be performed at window and FFT lengths of 512, 2048, and 8192 to determine the affect of point size on these algorithm migrations. The point sizes will be used to represent the entire range of point sizes the Decimator is capable of handling.

Figure 3.5 shows the system level layout of the two blocks implemented in VHDL. The input data and windowing coefficients must be supplied when simulating the system. The input data will be obtained in the simulation by reading captured Decimator data from a text file. The window coefficients will be generated in Matlab and truncated to 16-bit values. The window coefficients will then be loaded into a block RAM that simulates the shared RAM on the Decimator. The VHDL testbench can then obtain input samples of real and imaginary data in real time while the window filter and FFT process the correct samples of data based on their control signals.

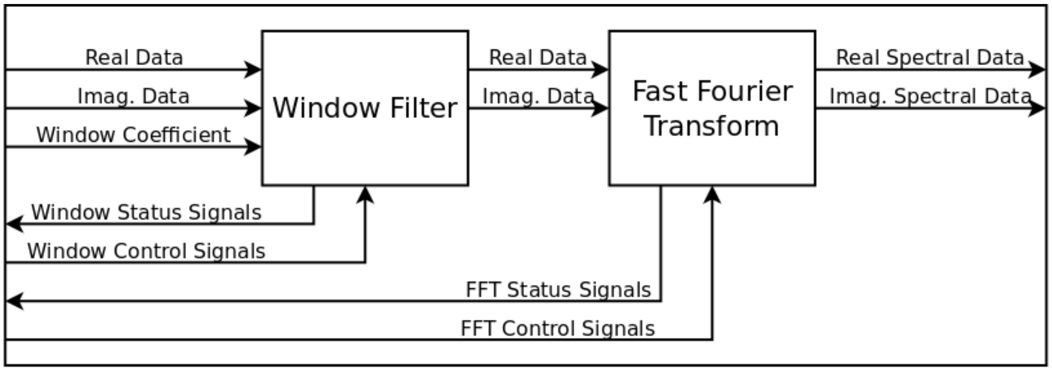

Figure 3.5: Window filter and FFT system level layout.

Spectral output of the FFT will be written to a text file that can be analyzed in Matlab. The Decimator displays the magnitude of the spectral data, so the real and imaginary output must be processed by Equation (3.1) before it can be displayed. Calculating the magnitude can be done on the microcontroller to save FPGA hardware resources. This choice also brings about two major benefits; namely, the output of the FFT can be scaled back to its actual level, and the required calculations to obtain the magnitude of I and Q samples can both be performed using 32-bit floating point precision.

$$Magnitude=\sqrt{I^2+Q^2} \qquad (3.1)$$

3.5.1 Block Floating Point (BFP) Versus 32-bit Floating Point

The microcontroller is simulated using Matlab since both operate using 32-bit floating point arithmetic. The output of the FFT core must be represented using 16-bits. The blk_exp variable is incremented each time there is an overflow in one of the butterfly calculations. The results of the Xilinx FFT core will be passed to the microcontroller for further processing, which achieves benefits that were mentioned earlier. The FFT results in the microcontroller may be expanded from their 16-bit representation to their actual representation by multiplying each spectral bin by $2^{\text{blk\_exp}}$. This expansion makes the results of the Xilinx FFT core comparable with the Matlab FFT results since they have the same scale.

3.5.2 16-bit Fixed Point Versus 32-bit Floating Point

A fully fixed point FFT simulation will be run for comparison with the previously mentioned BFP FFT implementation. Implementing the fixed point FFT core requires a pre-determined scaling schedule to be set in order to prevent butterfly overflows. The scaling schedule that will be used is one bit (1-bit) per stage (butterfly). This is a fairly aggressive setting to mitigate overflows, but it is necessary unless more advanced knowledge about the input data is known. The results of the 16-bit fixed point FFT implementation will be compared with the BFP results and the 32-bit floating point results. The hardware usage for the fixed point FFT core will also be presented in the hardware usage section of the results.

CHAPTER 4 : PRESENTATION of the RESULTS

4.1 Introduction

The system level designs for each part of this thesis were outlined in Chapter 3. Testing methodologies were also outlined to verify the functionality and performance of the outlined systems. The focus of this chapter is to present the results that were obtained from the previously described simulations and discuss whether the results are in line with the system requirements of the Decimator. At the conclusion of this chapter, there should be a full understanding of the designs that were implemented, how the designs were tested, and how well each design performed.

4.1.1 *Stat* Sources of Error

4.1.1.1 Coefficient Estimate Accuracy

There is always some error when calculating averages based on a finite data set. Performing the simulation described in Section 3.2.2.1 allowed the determination of the number of samples necessary to obtain accurate coefficient estimates. Figures 4.1 and 4.2 depict the number of samples used to calculate the coefficients versus the error in the estimates. It is clear that using more samples yields a more accurate estimate. The gain and phase estimates converge similarly, and it does not appear that more data is required by one method than another to obtain an accurate result.

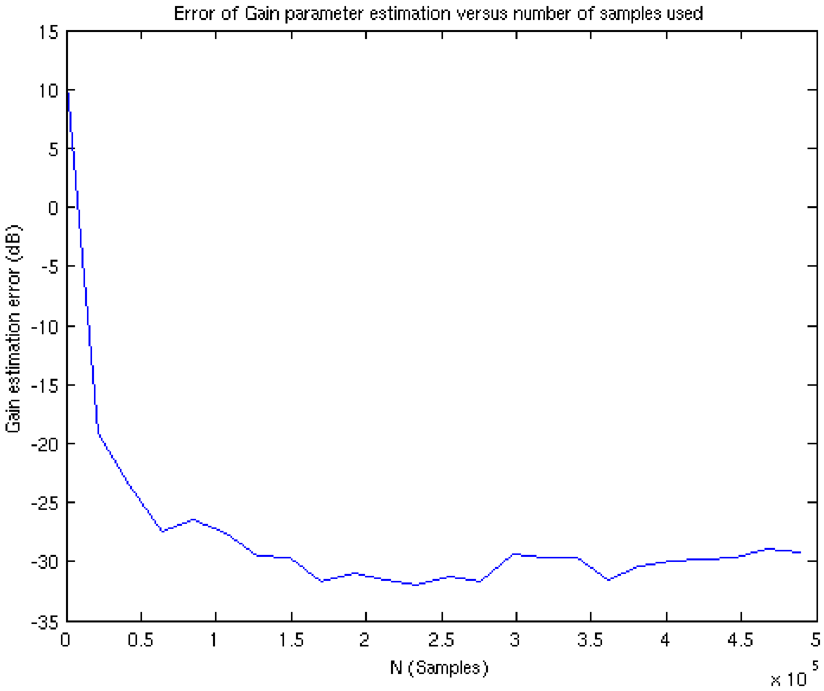

Figure 4.1: Error of gain coefficient estimate with respect to number of samples used.

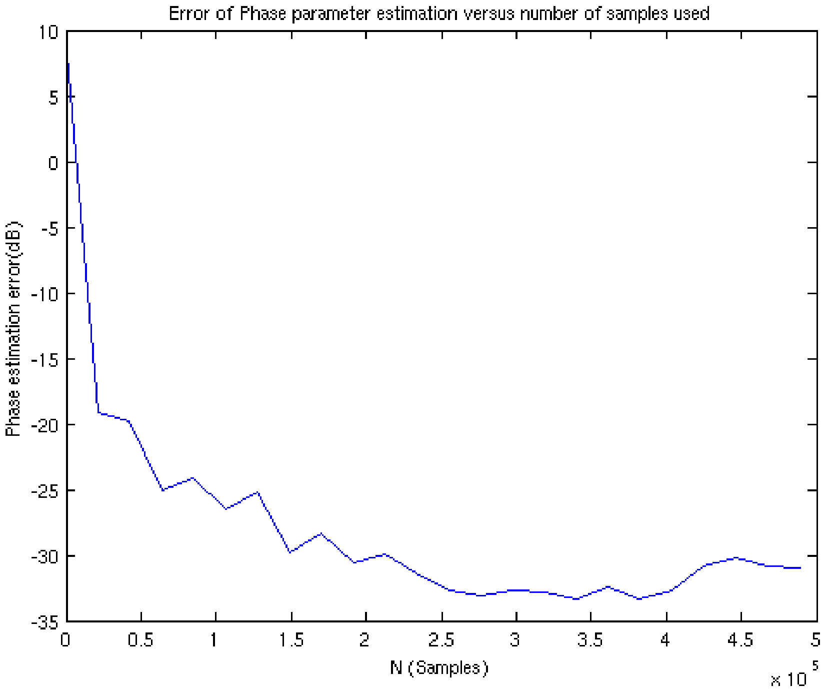

Figure 4.2: Error of phase coefficient estimate with respect to number of samples used.

One of the important observations the simulation is that the results do not converge to zero. There is always a small error present in the estimate regardless of the number of samples used to calculate the estimates. There is little to no benefit using more than 150,000 samples for both gain and phase estimates. The simulations show acceptable estimates are produced by using 100,000 samples. Choosing a point close to the corner of the exponential curve is not prudent. Doing so could possibly compromise the accuracy of the estimates because real transmitted signals may not converge quite as quickly as the random data that was generated. Using 154,354 samples, the gain estimate average is off by 0.029532 dB and the phase estimate average is off by 0.148969°.

4.1.1.2 32-Bit Floating Point Stat Performance

CAPTURE 1) Decimator_Time_Uncal The first data capture that was analyzed was taken from an early version of the Decimator. It exhibits more gain error than is found in the current version of the Decimator. This section discusses the performance of *Stat*, not

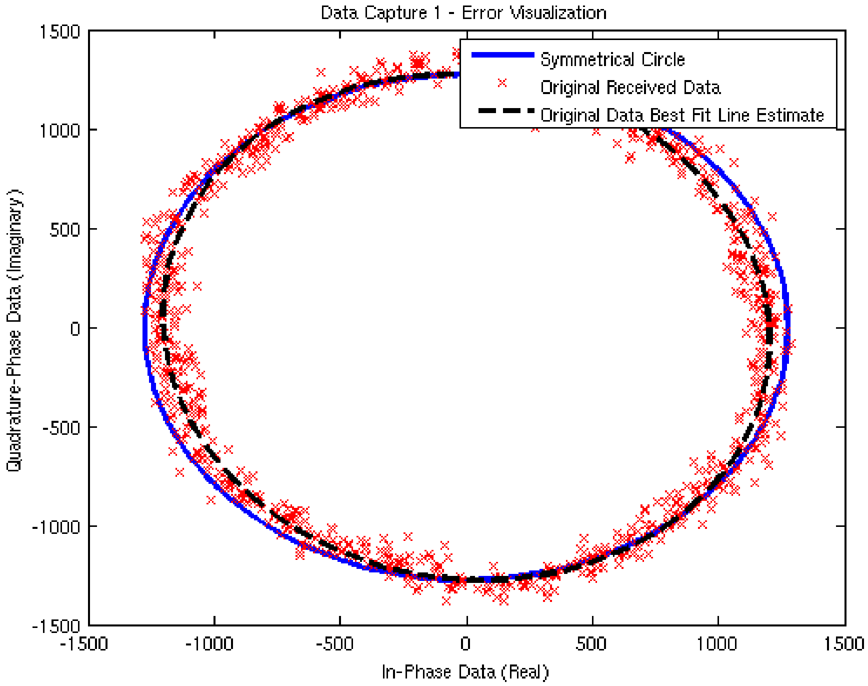

Figure 4.3: Capture 1 – Error visualized using best fit line estimate.

the Decimator, so the data still provides meaningful results. An x-y scatter plot of 1000 samples of the capture is seen in Figure 4.3. The 'x' markers indicate data points from the original received data. The circle serves as a reference point since it is perfectly round. Ideally, the received samples should be symmetrical about the circle.

It is clear that the received samples deviate from the circle because of both noise and I and Q imbalances. The dashed line in Figure 4.3 is a best-fit estimate of the received samples. The spectrum of the received data is seen in Figure 4.4. This zoomed view of the spectrum of the received signal shows that there is an imbalance between the two channels. Using 200,000 samples, *Stat* estimates the gain error to be 0.4940 dB, and the phase error to be 2.9586° in this capture.

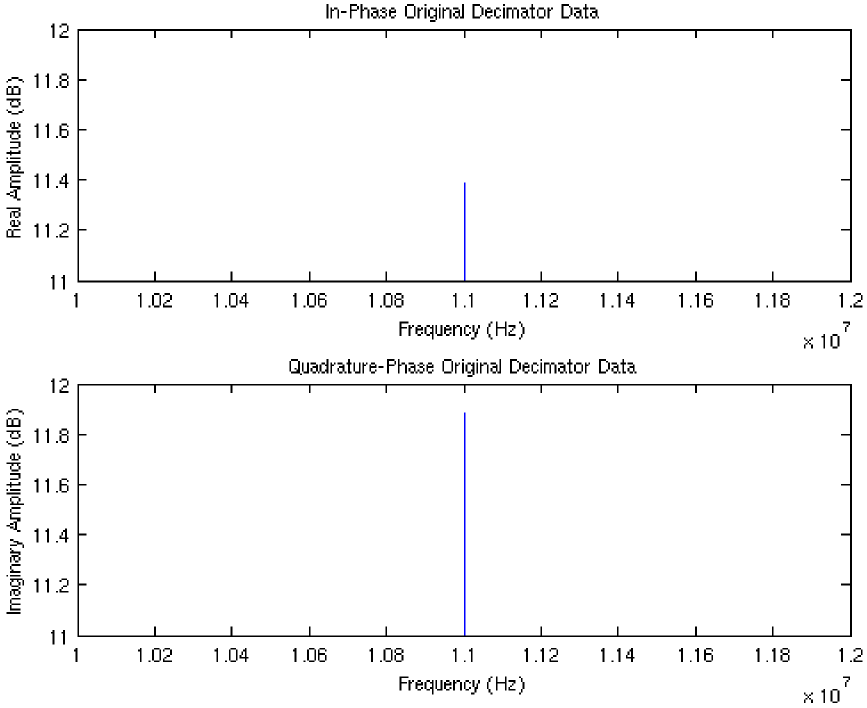

Figure 4.4: Capture 1 – Spectral peaks before correction.

Figure 4.5 shows the scatter plot of the received data before and after the correction coefficients have been applied. The corrected data matches with the reference circle whereas the original data showed a deviation from the reference circle, as shown in Figure 4.3. To visualize the gain imbalance, a zoomed spectral view of the capture is shown in Figure 4.6. Using the Root Mean Squared (RMS) values of the two channels yields an imbalance estimate of 0.4940 dB, which is the same as the estimate yielded by *Stat*.

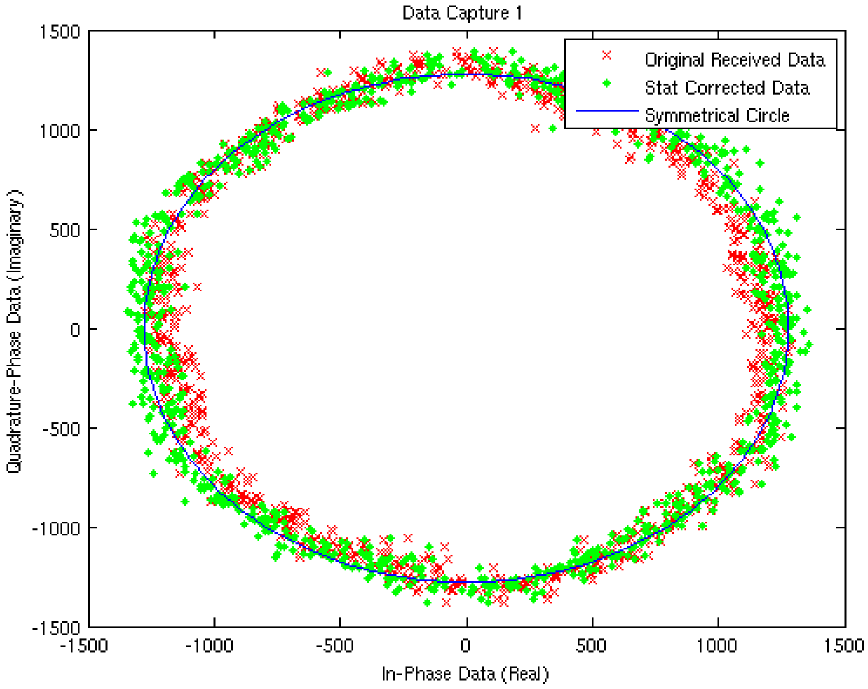

Figure 4.5: Capture 1 - PSK modulated data with significant imbalances.

The correction coefficients estimated by *Stat* were quite accurate When the correction coefficients were applied to the data the gain and phase errors were greatly reduced. The imbalance between the corrected signals is 0.0253 dB, which is an improvement of 0.4687 dB. *Stat* estimates the imbalance between the corrected signals to be 0.0253 dB as well. The phase imbalance of the received data is 2.9586° according to *Stat*, and it becomes -0.0000813° after correction for an improvement of 2.95852°. This Matlab simulation shows that *Stat* functioned effectively for this captured data.

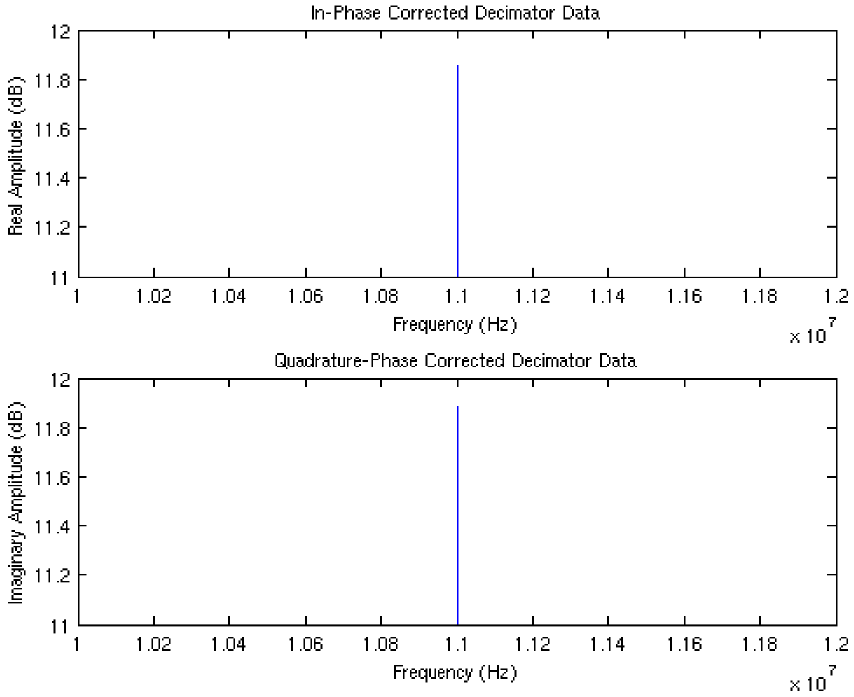

Figure 4.6: Capture 1 - Spectral peaks after correction.

CAPTURE 2) CW512Samples+5 The second capture was obtained from a current version of the Decimator, which has significantly reduced gain imbalances as compared to the older version that was used for the first capture. Figure 4.7 shows a plot of the original received data, along with the corrected *Stat* estimation and a reference circle. The capture is of PSK modulated sine waves at 5 MHz above the center frequency. Figure 4.7 shows that there is less imbalance present in this capture than there was in the first capture. *Stat* estimates a gain imbalance of 0.0702 dB and a phase imbalance of -2.1949°. The gain is considerably more accurate in this capture than it was with capture 1.

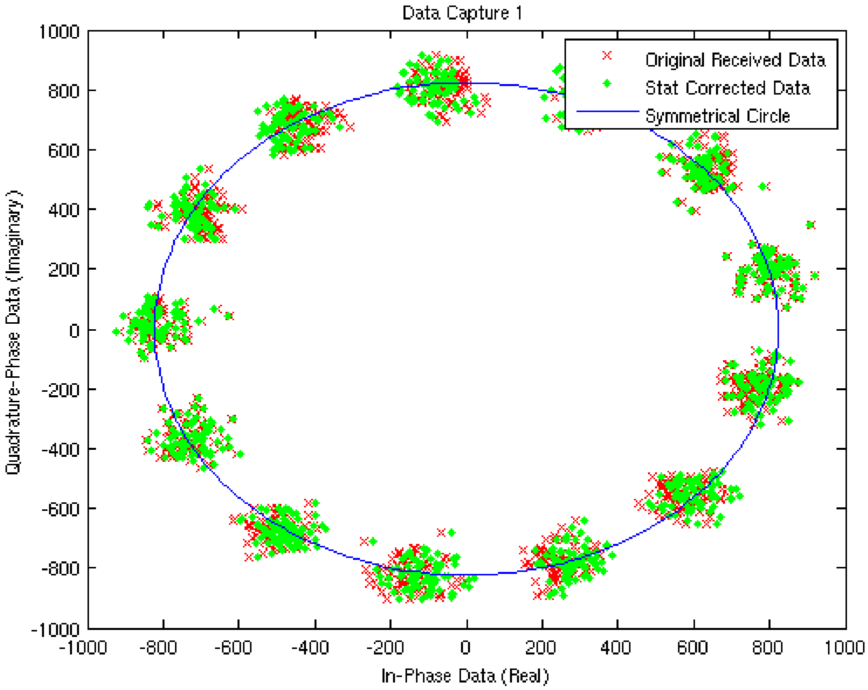

Figure 4.7: Capture 2 - PSK modulated received data.

Correcting the captured data and re-evaluating the errors yielded improved results. The new gain imbalance is 0.0067 dB and the phase imbalance is 0.00007136°. These are improvements of 0.0635 dB and 2.1948°, respectively. Once again, *Stat* corrected much of the error that was introduced by the RF receiver.

CAPTURE 3) CW+15 The third capture was also obtained from a current version of the Decimator. Figure 4.8 shows a plot of the original received data, along with the corrected *Stat* estimation and a reference circle. The capture is of PSK modulated sine waves at 15 MHz above the center frequency. The gain imbalance of the capture is 0.0985 dB and the phase imbalance is -0.3843°.

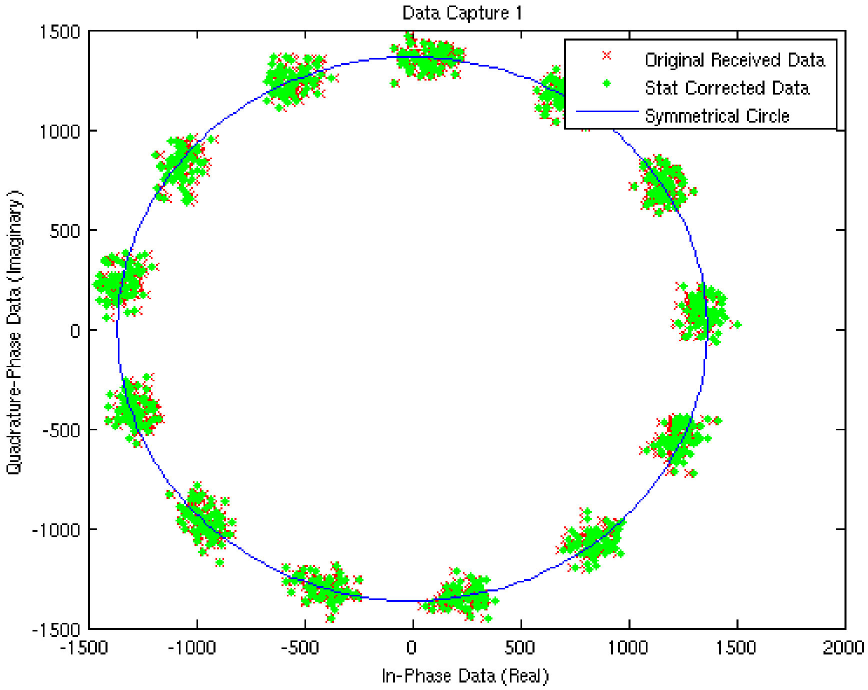

Figure 4.8: Capture 3 - PSK modulated received data.

After applying the correction coefficients, the gain imbalance improved to 0.00075114 dB and the phase imbalance improved to 0.00002459°. These are improvements of 0.09775 dB and 0.38432°, respectively.

Several other data captures were analyzed similarly to the ones described above. The results of these other captures functioned acceptably and no notable deviations in the performance of *Stat* were observed.

In conclusion, Stat operates well in a 32-bit floating point environment and is able to improve imbalanced data by at least a factor of 10. In several cases Stat improved the data by much more than a factor of 10. Figure 4.9 outlines the results obtained from the various Decimator data capture simulations.

Capture Number	Gain Improvement (dB)	Phase Improvement (°)
1	0.4687	2.9585
2	0.0635	2.1948
3	0.0978	0.3844

Figure 4.9: Data capture results summary.

4.1.1.3 Fixed Point Precision Affect on Stat

The VHDL and Matlab implementations of S*tat* were simulated using the same data to determine the performance degradation that will occur by implementing S*tat* in the FPGA. The correction coefficients were varied while the same data set was passed through the two S*tat* implementations. The arrangement allows for the direct comparison of the two simulations and indicates whether the FPGA correction differs from the Matlab correction simulations.

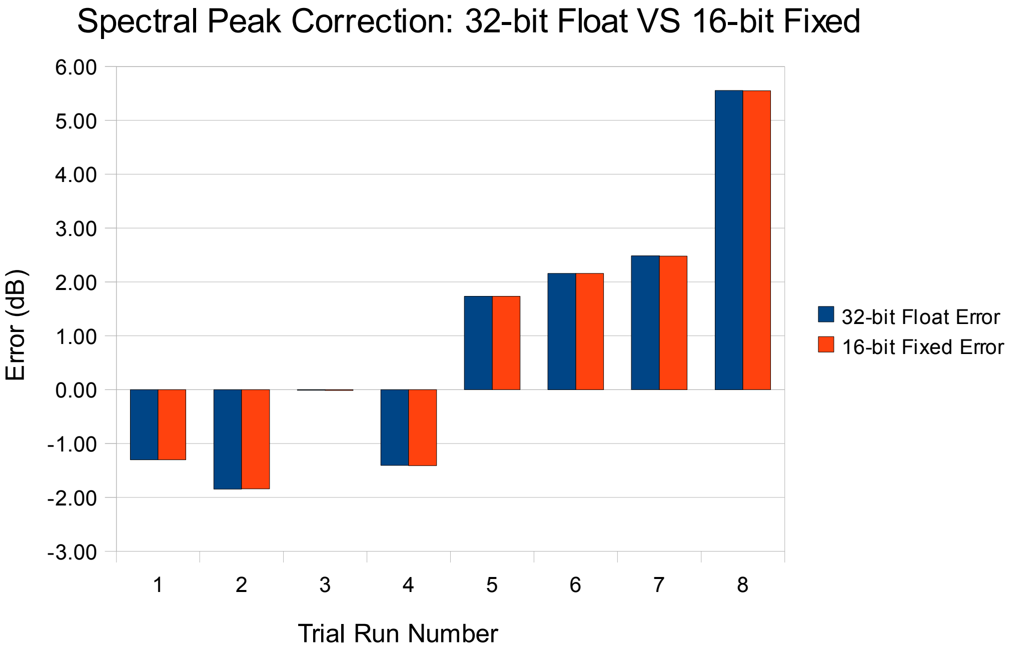

Figure 4.10: Spectral peak error comparison between Matlab and Xilinx ISE simulation.

Figure 4.10 displays the results obtained from the Xilinx ISE and Matlab simulations. The generated data was corrupted with a different set of correction coefficients in each trial and the spectral peak error was used as a measure of how well the corrections performed. Clearly the S*tat* algorithm is not adversely affected by the implementation hardware. While there is some error present with each trial, the important point to note is that the Matlab and Xilinx ISE simulations performed nearly the same. Empirically, the results deviate from each other by an average of 0.002656 dB. The amount of error introduced into the system from the fixed point arithmetic of the FPGA must therefore be negligible because the differences between the two simulations are so minimal. It can be concluded that the Matlab simulations presented previously provide a fairly accurate representation of how well *Stat* will function in the FPGA.

4.1.1.4 Arcsin Affect on Phase Estimates

The phase errors typically found in the Decimator are well under +/- 4°. Even in

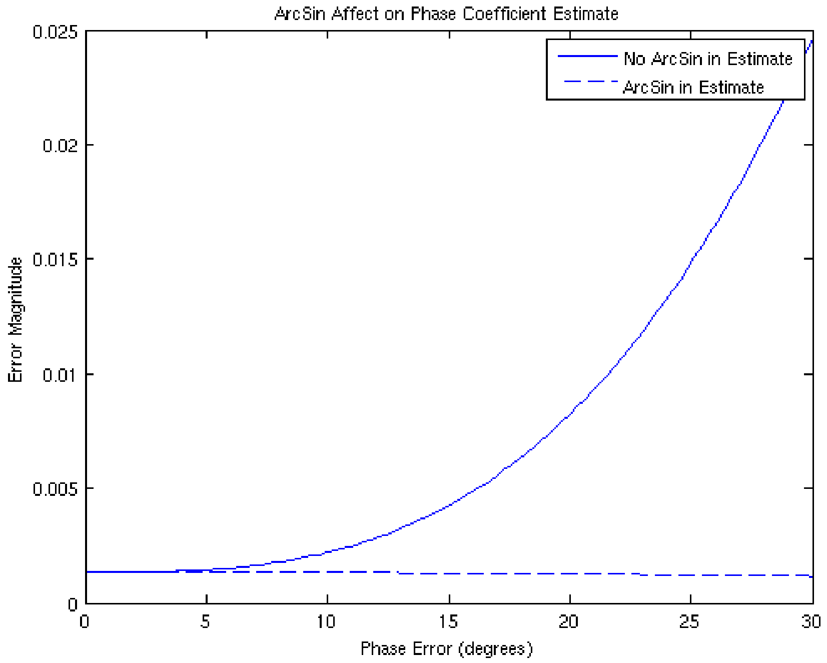

Figure 4.11: Arcsin Affect on Phase Coefficient Estimate

Capture 1, where the data was obtained from an older version of the Decimator, the phase error is less than 3°. The results of this simulation are shown in Figs. 4.11 and 4.12. For phase errors less than 4°, the difference in phase estimates is negligible. The *Arcsin* function begins to alter the results of the estimator equations when the phase error is above 4°. At 5° the difference between the two estimates is about 1 dB, and at 10° the difference is 4.57 dB. These results show that the error of the phase estimate increases exponentially when the *Arcsin* is not used. The difference between the two estimators is quite minimal when kept under 5°. The results of this simulation show that phase errors of 20° introduce a considerable amount of error in the phase coefficient estimate. It is not clear from these results how the 20° level was chosen in [13] to be the appropriate cutoff for Eq. (2.22).

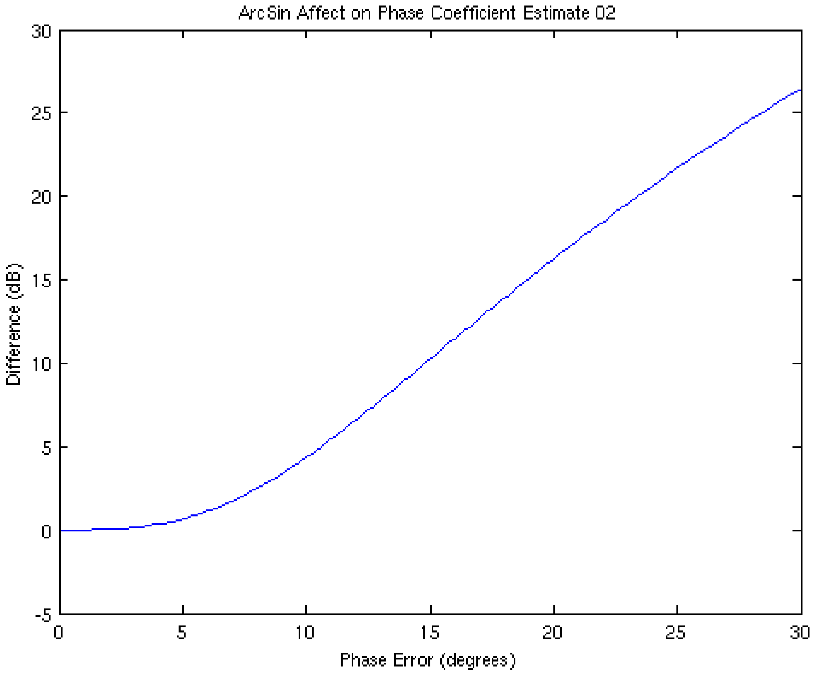

Figure 4.12: Error introduced in phase coefficient from not using Arcsin.

The simulations confirm the suggestion in [13] to remove the *Arcsin* from the phase coefficient estimation equation in order to conserve hardware resources. This conclusion can be drawn based on the fact that phase errors in the front end receiver components in the Decimator are typically much less than 5°, and errors under 5° contribute very little error to the coefficient estimate.

4.1.2 VHDL Resource Requirements

The resource requirements of the VHDL programmed S*tat* algorithms are quite low for all areas except one; 18 x 18 multipliers. Figure 4.13 depicts the resource requirements for the current Decimator implementation as well as the additional resources that would be used by the S*tat* I and Q imbalance correction module. It is apparent that the I and Q imbalance module would fit in the Spartan 3 1500 alongside the current Decimator code.

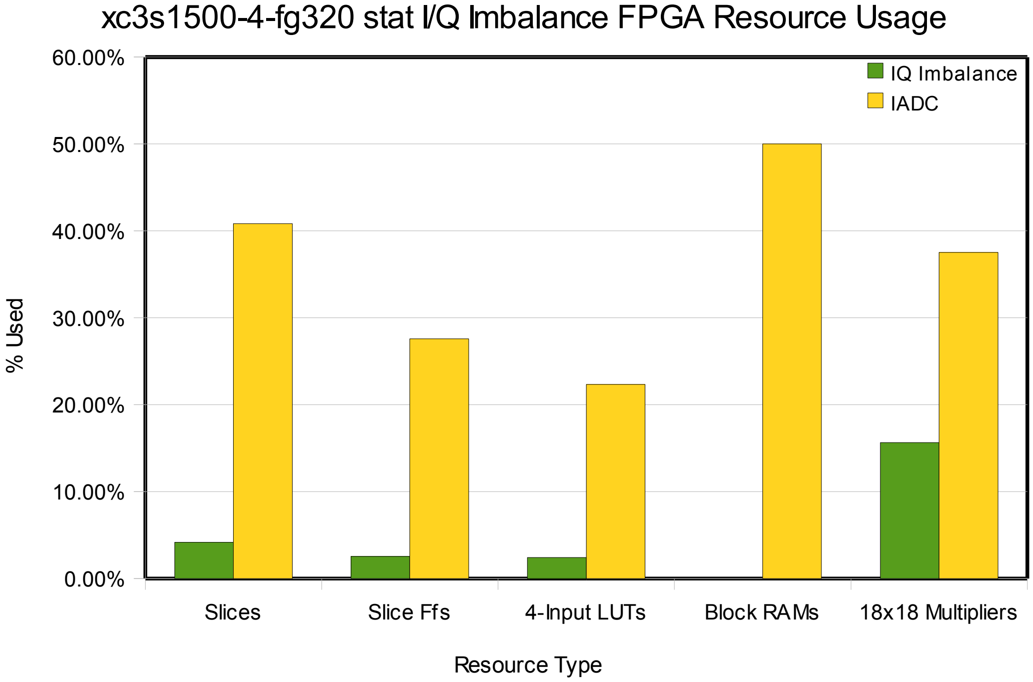

Figure 4.13: Stat I and Q Imbalance VHDL Resource Requirements.

The only concern with the S*tat* correction module is that it uses 5 of the total 32 18 x 18 multipliers on the FPGA. This means there would be 11 18 x 18 multipliers available on the FPGA for other algorithms and future development.

4.1.3 Speed Requirements

The Decimator operates at 65 MHz near the front end of the Decimator where the *Stat* module would be implemented. This means *Stat* must operate at least at 65 MHz to be implemented seamlessly into the received data processing chain. The synthesis report for the *Stat* module shows its maximum operating speed to be 64.574 MHz at a speed grade of -4. This is close to the required speed, but it is not sufficiently close. The present design of the I and Q correction module is not fast enough and will require modification to make it viable for the Decimator.

Figure 3.1 should be reviewed in an attempt to locate the bottleneck in the I and Q imbalance VHDL model. The "Channel Select" block acts as a switch and has a fairly simple design. The subsequent gain and phase multiplications are straight forward and are capable of operating at much higher speeds than 65 MHz, so the problem is most likely not due to any of these operations. The "Multiply Accumulate" block could possibly be the problem, but given its low level nature, this is unlikely. The most likely culprit is the "State Machine" because there are a number of signals that drive the case statements in it.

Reviewing the HDL Synthesis report confirms these suspicions. The report indicates that there is a latch in the "Multiply Accumulate" block and that it may be causing timing problems. Even more critical are the warnings regarding the Synthesis of the "State Machine." According to the Xilinx ISE Synthesis report:

> "INFO:Xst:2371 - HDL ADVISOR - Logic functions respectively driving the data and gate enable inputs of this latch share common terms. This situation will potentially lead to setup/hold violations and, as a result, to simulation

> problems. This situation may come from an incomplete case statement (all selector values are not covered). You should carefully review if it was in your intentions to describe such a latch."

This statement is an obvious red flag that indicates a flaw in the VHDL code. Although the I and Q imbalance code simulated fine, it does not synthesize properly because the timing of the system has been so adversely affected by the inferred latch. Correcting the "State Maching" block is necessary before proceeding with further system integration.

4.2 Windowing and the Fast Fourier Transform (FFT)

4.2.1 Simulation Results

A number of simulations were performed using the same data capture that has a carrier wave at 15 MHz above the baseband. Results were obtained from both the Xilinx ISE Simulator and from Matlab. The simulations varied the type of window that was used and the length of the window and FFT calculation that was performed. These results display the error that can be attributed directly to the migration of these algorithms from the floating point microcontroller to the fixed point FPGA. The microcontroller is equivalent to the Matlab 32-bit floating point results, and the FPGA is equivalent to the Xilinx ISE 16-bit fixed point results.

4.2.1.1 Block Floating Point (BFP) Versus Floating Point

The first set of simulations compares the error introduced by the window filter and FFT core when the FFT core is implemented using BFP arithmetic. Figure 4.14 displays the results of the simulation where the error is the average deviation in Decibels between the fixed and floating point simulation results.

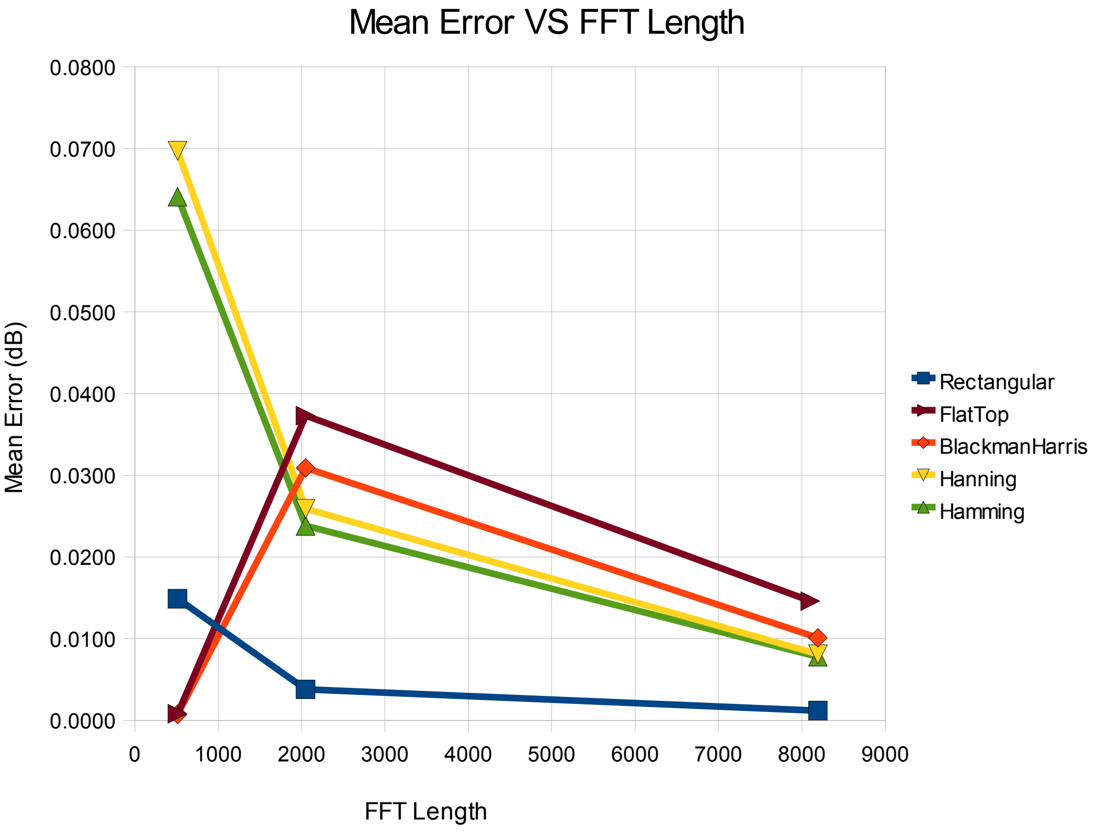

Figure 4.14: Average FFT bin error VS length of FFT.

Figure 4.15 presents a similar comparison to that shown in Figure 4.14, however, only the error of the peak spectral bin is evaluated. It is important to evaluate the peak spectral bin because many of the measurements taken by the Decimator are of the peaks of transmitted signals. It is clear that Figs. 4.14 and 4.15 appear to be similar, but it is also important to note the magnitude of the error associated with each point on the graphs. The spectral peak errors are much larger than the average spectral error which means there is an above average error present at the peaks generated by the fixed point hardware. This is a drawback associated with fixed point hardware but is still relatively insignificant.

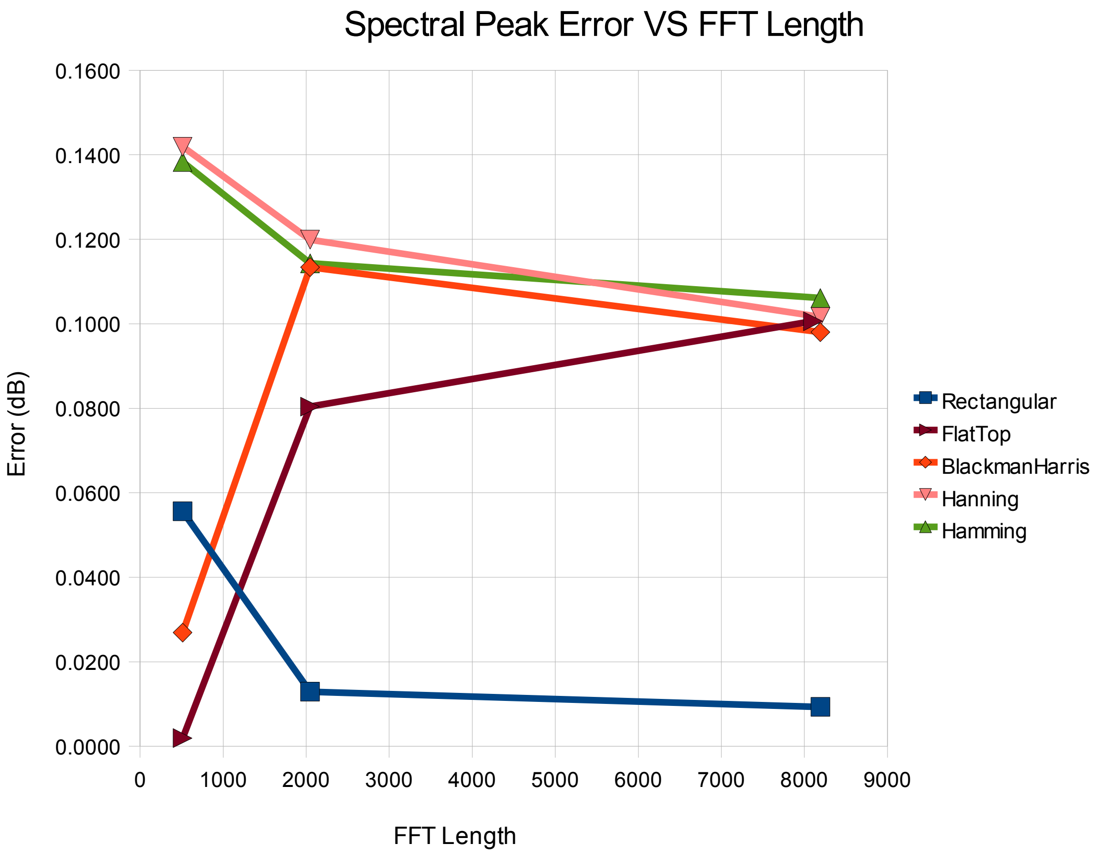

Figure 4.15: Carrier peak error VS length of FFT.

Figures 4.14 and 4.15 have similar shapes to the graphs. An initial conclusion might be to assume that the FlatTop and BlackmanHarris 512 point simulations are erroneous, and that the others are correct. Upon further study of the spectrums of the simulation results, it is apparent that the FlatTop and BlackmanHarris 512 point simulations are most likely the correct results and the others are erroneous. Figure 4.16 shows the results of the 2048 point FlatTop windowed and FFTed data produced by the Xilinx and Matlab simulations. While the results of the FlatTop and BlackmanHarris 512 point Xilinx simulations compare quite accurately to their Matlab counterparts, the other simulations look similar to those shown in Figure 4.16. The Xilinx results contain an added erroneous signal that creates a spectral arc near the 15 MHz carrier wave. The results for the other simulations are similar, however the arc becomes narrower with higher point sizes.

The spectral error as displayed in Figure 4.16 is either caused by the windowing module that was developed, or by the Xilinx FFT core that was used for the simulations. To determine where the error was originating, the output of the Xilinx window filter was put through the Matlab FFT. The results are similar to those in Figure 4.16 which means the error is caused by the window filter designed for the FPGA. It is not known why this error is created only with certain windows and at certain point lengths. Reviewing the Synthesis report did not yield any indications of an ill-designed block. Troubleshooting the VHDL window filter will have to be a topic of further study.

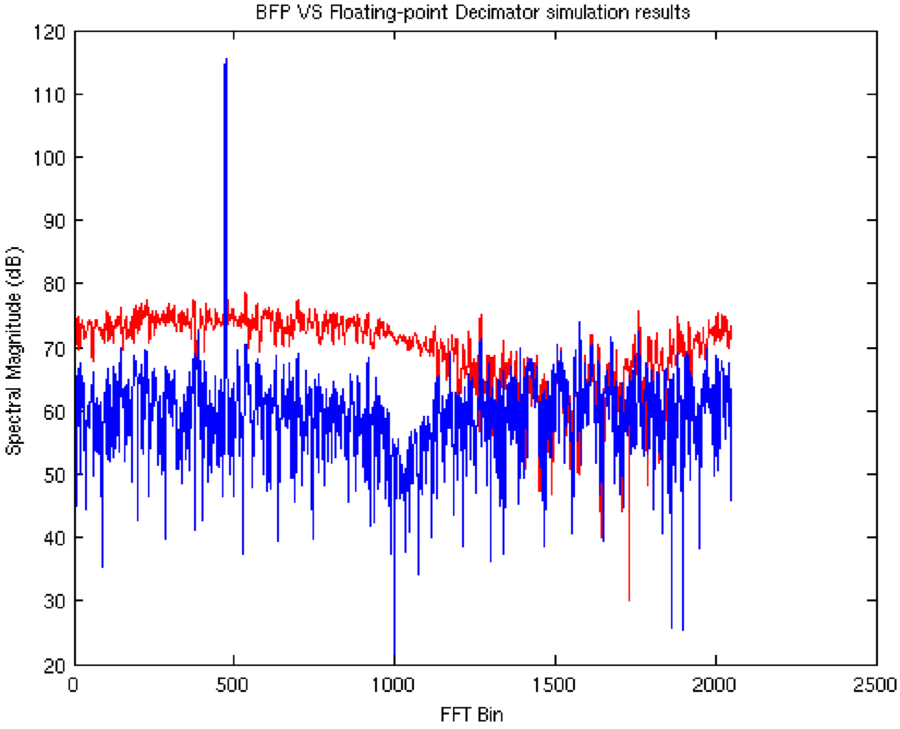

Figure 4.16: Window and FFT calculated results: Matlab vs Xillinx.

The rectangular window in both Figs. 4.14 and 4.15 displays significantly lower error values than their Hamming and Hanning windowed counterparts while maintaining a similar shape. Although a large error has been discovered in the results of the window filter block, the differences between the rectangular window and the Hamming and Hanning windows ought to provide some insight into the performance of the Xilinx ISE simulations. Incoming data samples are multiplied by one and passed to the FFT because the rectangular window contains only ones as coefficients. This results is little to no rounding error being introduced by the window filter.

The error that is present in the rectangular window simulations is entirely due to the FFT calculation and the newly discovered error in the design of the window block. Comparing the values of the rectangular window simulations with their Hamming and Hanning counterparts, it is quite possible that the majority of the increased error in the simulation was introduced by the window filter, and not the FFT. As an example, the Hamming 2048 point simulation had on average 6.263 times the amount of error of the rectangular 2048 point simulation. This means the average amount of error introduced by the window filter stage of the hardware is over six times as much as the FFT stage of the hardware. Therefore, the dominant source of error by migrating the windowing and FFT algorithms to the FPGA will most likely come from the window filter. Of course, the design flaw in the window filter means that this conclusion cannot be formally verified.

4.2.1.2 Fixed Point Versus BFP and Floating Point

The simulation comparing fixed point versus BFP and floating point is identical to the one discussed previously in Section 4.2.1.1 except it includes results obtained from a fixed point FFT implementation in Xilinx ISE. By comparing the BFP results to the fixed-point results, the loss in precision by using fixed-point arithmetic over BFP were apparent.

Figure 4.17 displays the results of the simulations with the measure of error being mean deviation from the 32-bit floating point arithmetic results in Decibels. As predicted the fixed-point introduces a significant amount of extra error into the results of the windowed and FFT calculated data. The 8192 length FFTs display some telling results. In both cases the error introduced by the fixed point FFT arithmetic increased the total system error by a factor of about three. This means that the error attributed only to the fixed point FFT could potentially be twice that of the error introduced by the BFP FFT.

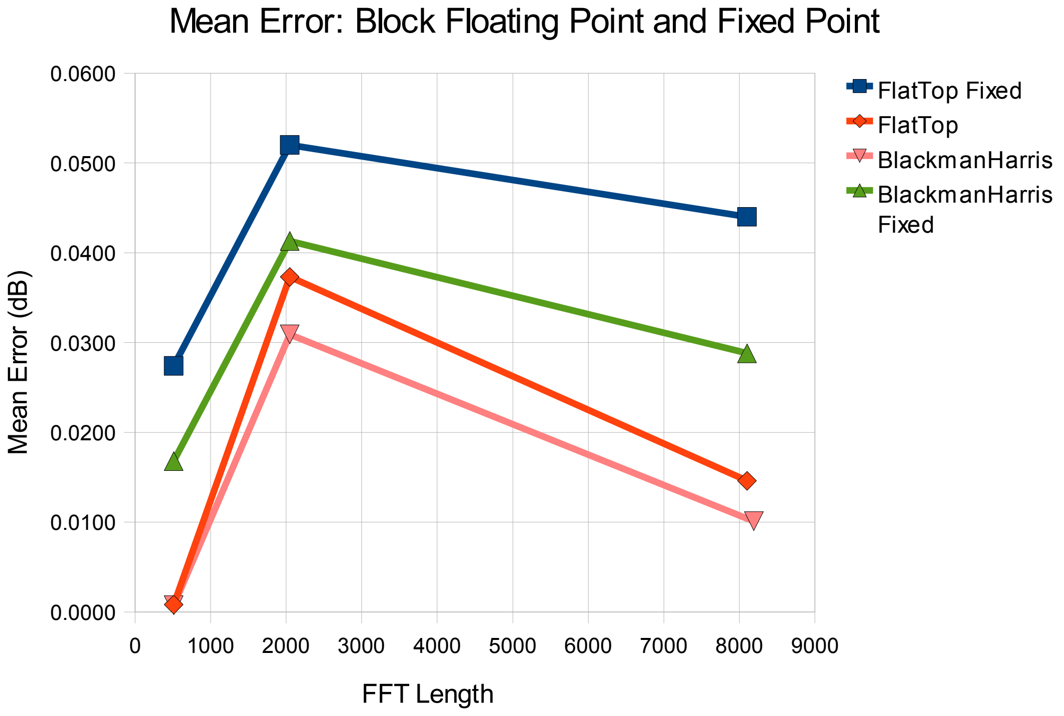

Figure 4.17: Loss of precision caused by BFP and fixed point arithmetic in the FFT.

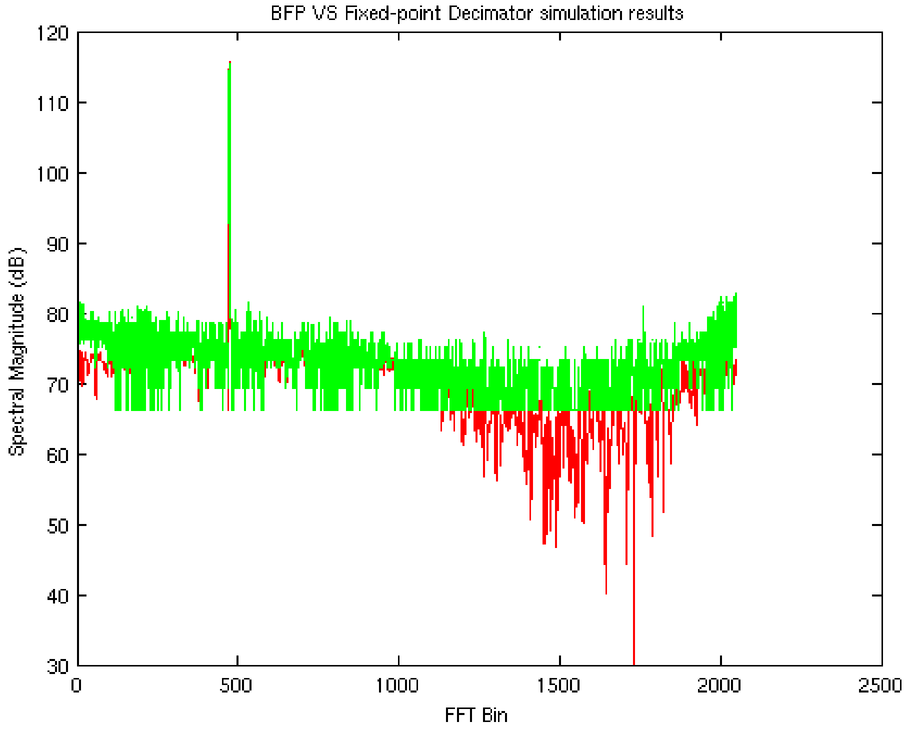

Figure 4.18: Block floating point VS fixed point FFT implementation. 2048-point FlatTop window.

Figure 4.18 displays the 2048-point FlatTop window results of the BFP and Fixed-point FFT core. The noise that is added to the system from the fixed-point scaling schedule in the FFT is quite evident. Most notable is the loss in dynamic range of the FFT when using fixed-point arithmetic. The fixed-point results bottom out at about 74 dB, whereas the BFP results are able to resolve spectral data down to 30 dB. This is a substantial loss in precision. The scaling schedule should help to quantify the reason for this loss in precision and dynamic range. For a 2048 fixed-point FFT, such as what was processed in Figure 4.18, there is a possible bit growth of 11-bits. The bit growth will occur if there is an overflow caused by each butterfly in the FFT. The scaling schedule that was used in this simulation was one bit per stage. The results of the FFT then need to be scaled by 2^{11} to be comparable to floating-point results. By contrast, the BFP FFT is sufficiently intelligent to only scale after a butterfly when an overflow has occurred. At the end of the FFT, the *blk_exp* indicates how many bits were lost during the FFT processing. The *blk_exp* variable was 5 for this simulation, which means using the fixed point pre-determined scaling schedule truncated 6-bits of data unnecessarily.

It was discussed previously that each extra bit used to represent the input and output data of an FFT theoretically increases the dynamic range by about 6 dB. A rough estimate of the loss of dynamic range experienced by the fixed point scaling result seen in Figure 4.18 is found by multiplying the unnecessary loss of bits by the affect each bit has on dynamic range. Therefore, $6\,dB/bit*6\,bits=36\,dB$. Figure 4.18 shows a loss of 36 dB between the smallest signal in the BFP simulation and the smallest signal in the scaled fixed point simulation. Although the actual and theoretical values are equal, it cannot be concluded that the theoretical prediction is 100% accurate at determining dynamic range [25]. The estimate has provided a good indication of the loss of dynamic range in this case.

The performance improvement by using BFP is certainly apparent from the simulations that were performed. Of course, the tradeoff in using BFP arithmetic in the FFT is an increase in hardware resource usage. The tradeoff will be discussed in the hardware usage section that follows.

4.2.2 Hardware Usage

Migrating the window filter and FFT algorithms has proved to be functionally viable, however, limited FPGA resources make these changes impossible to implement. Figure 4.19 shows the percentage of total FPGA resources needed for each FFT architecture type along with the developed window filter. The "IADC" bar at the far right of each category indicates the percentage of FPGA resources that are currently used by the Decimator. Successfully migrating the windowing and FFT algorithms to the FPGA requires adding one of the first four bars in each column with the "IADC" bar without exceeding 100% resource usage in any column.

The "Block RAMs" column of Figure 4.19 shows the limiting factor of migrating these algorithms to the FPGA. The current Decimator implementation utilizes 50% of the available block RAMs on the Spartan 3 1500 but all FFT implementations require at least 56% of the block RAMs on the FPGA. As such, migrating the FFT algorithm to the FPGA is impossible.

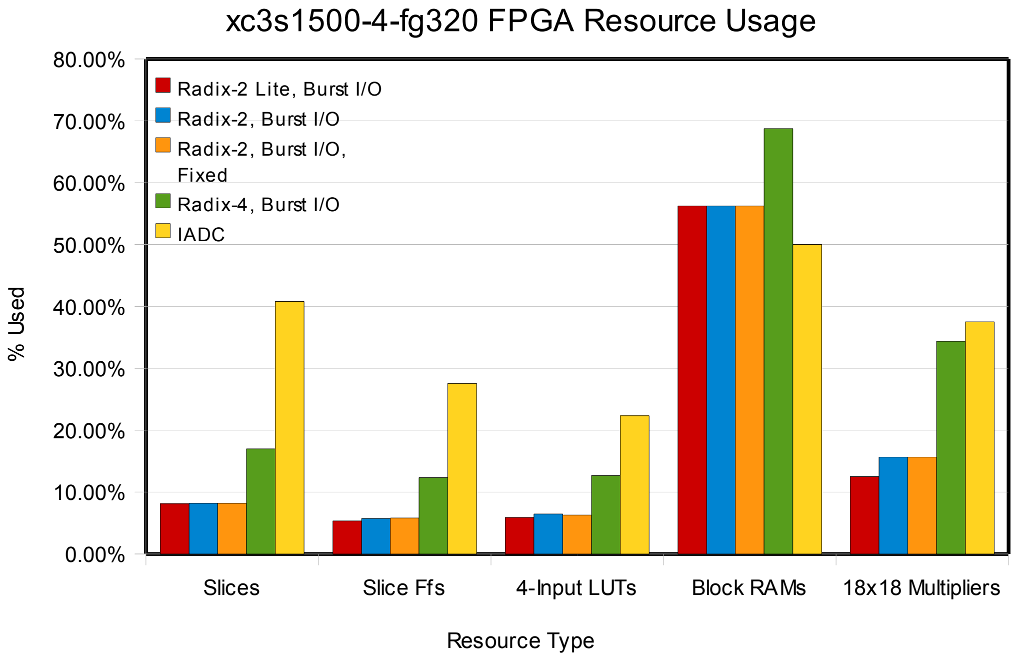

Figure 4.19: Windowing and FFT Resource Usage

A possible solution to this problem is to reduce the maximum point size of the FFT core to 4096. The window filter and FFT core were again instantiated and synthesized using 4098 as the maximum point size supported in order to provide verification of this point. While most of the resource requirements stayed near previous synthesis reports, the number of block RAMs dropped to 9. This reduction of block RAMs by half would allow the FFT to fit in the current Xilinx Spartan 3 chip. Unfortunately the loss of the 8192 window and FFT calculation would be required for the windowing and FFT operations to be implemented in the FPGA.

The additional hardware requirements of the BFP over the fixed-point arithmetic in the FFT are quite minimal. As Figure 4.19 shows, the only resource that the BFP core uses more of than the fixed-point core is 4-input LUTs. The increase in 4-input LUTs is minimal. The number of slices, block RAMs, and 18 x 18 multipliers all either remain the same or increases a negligible amount. Slice flip flop usage is actually less in the BFP FFT core implementation. Given the increase in precision that was shown previously in Section 4.2.1.2 and the minimal change in FPGA resource requirements, it is concluded that BFP ought to be used when implementing the FFT core in the FPGA.

4.2.3 Speed Requirements

The signal processing in the Decimator operates at 65 MHz, therefore the window filter and FFT core must operate at least at this frequency to avoid a more complex integration into the Decimator. According to the synthesis results, the window and FFT blocks can operate at a maximum frequency of 99.463 MHz at a speed grade of -4. This maximum speed remained constant despite varying the FFT architecture and various other FFT settings. The synthesis report for the FFT core that was reduced to a 4096 maximum point size also reported this same speed. Therefore the window filter and FFT core should be more than capable of handling the 65 MHz clock in the signal processing section of the Decimator.

CHAPTER 5 : CONCLUSIONS and RECOMMENDATIONS

5.1 Introduction

This thesis has focused on several key issues relating to a SED Systems product called the Decimator. The first issue that was evaluated was whether or not it is possible to correct the I and Q imbalance that is present in the analog RF receiver. The I and Q imbalance error is one of the main sources of error in the Decimator, and as such, a method of improving this error was sought. A general overview of the problem was presented, and a suitable solution was found. The statistical I and Q imbalance correction algorithm known as "*Stat*" was implemented in VHDL and in Matlab, and was simulated to verify its performance in both of these architectures.

The second issue this thesis dealt with was the migration of two key signal processing algorithms from the on-board microcontroller to the on-board FPGA. The first signal processing algorithm that was migrated was the window filter. The window filter currently implemented in the microcontroller was used as a reference design and a similar instantiation of it was coded in VHDL. The second algorithm migration that was reviewed was the FFT. Like the window filter, it was desired for the FFT processing to occur in the FPGA rather than the microcontroller. While migrating these algorithms was desired for speed and future consideration reasons, the microcontroller provides a 32-bit floating point environment in which to perform these calculations. The FPGA, on the other hand, is inherently fixed point in its arithmetic. Both algorithms were implemented in VHDL and were simulated to help quantify the loss in precision that would occur if these algorithms were migrated to the FPGA.

5.2 I and Q Imbalance

The I and Q correction scheme that was evaluated is called *Stat*. The results of the simulations all showed that *Stat* is a viable option for correcting front-end receiver error. The main issue that was discovered during the simulations is that the current VHDL implementation of *Stat* cannot operate at the target speed of 65 MHz. The reasons for this behavior is most likely due to the implementation of the "State Machine" block. According to the synthesis report there were unintended latches in the design. Latches often adversely affect the timing of a block. Removing the latches in the design of the "State Machine" block would undoubtedly improve the maximum speed to well above the 65 MHz target speed.

An issue for further study is the response of *Stat* to transmitter I and Q imbalances. This thesis has only exposed the correction algorithm to imbalances originating in the RF receiver. *Stat* was not implemented and tested with the intent to correct transmitter I and Q imbalances. There will most likely be instances where the Decimator is exposed to I and Q imbalances that have originated in the transmitter. Therefore, quantifying the correction ability of *Stat* with imbalances that originated in the transmitter would certainly be beneficial.

5.3 Windowing

The basic window filter design in the microcontroller was used to develop a VHDL window filter. The design performs most of the processing in the FPGA, but still obtains its coefficients from the microcontroller. The window filter that was designed seemed to operate properly, but later turned out to be introducing a substantial error into the processed signals. The cause of this error was not determined. Without a fully operational window block in VHDL, comparing the Matlab and Xilinx simulations is inconsequential. One suggestion is that rounding be implemented rather than truncating, which may be adding a bias to the resultant windowed signal.

5.4 Fast Fourier Transform (FFT)

The Xilinx FFT core was evaluated as a possible solution to the FFT migration to the FPGA. The Xilinx FFT core offers a variety of settings to make it viable in a number of different applications. Each architecture was implemented and simulated. The results of these simulations showed that there would be a minute change in the fixed-point precision error introduced into the system from the FFT. From a performance standpoint, the Xilinx FFT core would operate fast enough and accurately enough to perform well in the Decimator.

The number of block RAMs required for the lightest FFT architecture, Radix-2 Lite, exceeds the number that are available in the Spartan 3 1500 FPGA. The only way to reduce the number of required block RAMs is to reduce the maximum point size of the FFT to 4096. The Decimator currently handles point sizes up to 8192, so reducing the maximum available point size in order to migrate the FFT to the FPGA is an undesired consequence. An area of future study is whether there are any other available FFT cores that are free and would fit in the Spartan 3 1500. If there are no available FFT cores that meet the block RAM constraints in the Spartan 3 1500, it may be possible to design and implement a fully custom FFT that fits the requirements of the Decimator.

5.5 Conclusions

This thesis has focused on several issues related to the performance and implementation of a device called the Decimator. From theory to bit accurate simulations, these issues have been thoroughly investigated and meaningful conclusions have been drawn from their results. The results of this thesis should provide SED Systems Ltd with meaningful data to be of assistance with decision processes related to changes to the next revision of the Decimator.

As is the case with all research, many venues were not fully explored and many new questions came to light as a result of the research. While this thesis fell short of fully integrating the described changes in the Decimator, many key questions regarding the implementation of these changes have been answered. This thesis would undoubtedly be of value to anyone working to improve the described issues in either the Decimator or in any other similar system.

References

[1] A. V. Oppenheim and R. W. Schafer with J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1999.

[2] A. V. Oppenheim and A. S. Willsky with S. H. Nawab, *Signals & Systems*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1997.

[3] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, 1st ed. Upper Saddle River, NJ: Prentice Hall, 1993.

[4] A. Papoulis, S. U. Pillai, *Probability, Random Variables and Stochastic Processes,* 4th ed. New York, NY: McGraw-Hill, 2002.

[5] M. Valkama and M. Renfors and V. Koivunen, "Advanced Methods for I/Q Imbalance Compensation in Communication Receivers," *IEEE Transaction on Signal Processing*, Vol. 49, No. 10, pp. 2335-2344, October 2001.

[6] E. Cetin, I. Kale, and R. C. S. Morling, "Adaptive Digital Receivers for Analog Front-End Mismatch Correction," *IEEE VTS 54th Vehicular Tech. Conf. (VTC 2001 Fall)*, vol. 4, pp. 2519-2522, 2001.

[7] E. Cetin, S. Demirsoy, I. Kale, and R. Morling, "Efficient FPGA Implementation of an Adaptive IQ-Imbalance Corrector for Communication Receivers Using Reduced Range Multipliers," *Proc. European Signal Processing Conference* (EUSIPCO '05). Sept. 2005.

[8] P. Rykaczewski and F. Jondral, "Blind I/Q Imbalance Compensation in Multipath Environments," *Proc. Int. Symp. Circuits and Systems,* 2007. (ISCAS'07), New Orleans, LA, pp. 29-32, May 2007.

[9] A. J. Bell and T. J. Sejnowski, "An Information-Maximisation Approach to Blind Separation and Blind Deconvolution," *Neural Computation*, Vol. 7, No. 6, pp. 1129-1159, 1995.

[10] K. Torkkola, "Blind Separation of Convolved Sources Based on Information

Maximization," *IEEE Workshop Neural Networks for Signal Processing*, Kyoto, Japan, pp. 423-432, Sept 4-6, 1996.

[11] A. Celik, M. Stanacevic, and G. Cauwenberghs, "Mixed-Signal Real-Time Adaptive Blind Source Separation," *Proc. Int. Symp. Circuits and Systems* (ISCAS'04), Vancouver, Vol 5, pp. 760-763, May 2004.

[12] F. Harris, S. Parekh, I. Gurantz, "I-Q Balancing Techniques for Broadband Receivers," in *2005 Software Defined Radio Technical Conference (SDR), Proceedings,* Hyatt Regency, CA, USA, Nov. 2005.

[13] M. Kocic, L. Martinot, Z. Zvonar, "Signal Processing Techniques for EDGE Wireless Modem," *The Int'l Conference on "Computer as a Tool"* (EUROCON '05), Belgrade, Vol 1, pp. 131-134, Nov. 2005.

[14] P. Rykaczewski, M. Valkama, M. Renfors, and F. Jondral, "Non-Data-Aided I/Q Imbalance Compensation Using Measured Receiver Front-End Signals," in *Proc. 17th Annual IEEE Int'l Symp. On Personal Indoor and Mobile Radio Communications* (PIMRC)*,* Helsinki, Finland, pp. 1-5, Sept. 2006.

[15] G. Gil, "Nondata-Aided I/Q Mismatch and DC Offset Compensation for Direct-Conversion Receivers," *IEEE Trans. On Signal Proc.*, Vol 56, No. 7, pp. 2662-2668, July 2008.

[16] L. Anttila and M. Valkama and M Renfors, "Blind Moment Estimation Techniques for I/Q Imbalance Compensation in Quadrature Receivers," *IEEE Int. Symp. on Personal, Indoor and Mobile Radio Comm.* (PIMRC'06)*,* Helsinki, pp. 1-5, Sept. 2006.

[17] L. Anttila and M. Valkama and M Renfors, "Circularity-Based I/Q Imbalance Compensation in Wideband Direct-Conversion Receivers," IEEE Transactions on Vehicular Technology, Vol. 57, No. 4, pp. 2099-2113, July 2008.

[18] N. Moseley, and C. Slump, "A Low-Complexity Feed-Forward I/Q Imbalance Compensation Aalgorithm." In *17th Annual Workshop on Circuits*, Veldhoven, The Netherlands. pp. 158-164, Nov. 23-24 2006.

[19] M. Windisch and G. Fettweis, "On the Performance of Standard-Independent I/Q

Imbalance Compensation in OFDM Direct-Conversion Receivers," *Proceedings of 9th International OFDM Workshop* (InOWo'04), Dresden, pp.57-61, Sept. 2004.

[20] M. Windisch and G. Fettweis, "Blind Estimation and Compensation of I/Q Imbalance in OFDM Receivers with Enhancements through Kalman Filtering," *IEEE/SP 14th Workshop on Statistical Signal Processing*, pp.754-758, August 2007.

[21] Xilinx Inc., "Fast Fourier Transform v5.0 LogiCORE: Product Specification," document DS260, October 10, 2007.

[22] A. Oppenheim, and C. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," *Proceedings of the IEEE*, Vol. 60, No. 8, pp. 957-976, 1972.

[23] A. Mitra, "On Finite Wordlength Properties of Block-Floating-Point Arithmetic," *Int. Journal of Signal Processing*, Vol. 2, No. 2, pp. 120-125, 2006.

[24] E. O. Bringham, and L. R. Cecchini, "A Nomogram for Determining FFT System Dynamic Range," IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP '77), Vol. 21977, pp. 623-627, 1977.

[25] D. Formenti, "S&V Questions and Answers: Dynamic Range," *Sound and Vibration*, pp. 2-3, May 1999.

[26] Q. H. Nguyen, and I. Kollar, "Limited Dynamic Range of Spectrum Analysis Due to Roundoff Errors of the FFT," *IEEE Instrumentation and Measurement Technology Conference* (IMTC '93), Irvine, CA, pp. 47-50 May 18-20, 1993.

[27] W. Chang, and T.Q. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms," *IEEE Trans. On Signal* Processing, Vol. 56, No. 10, pp. 4673-4682, Oct. 2008.

[28] X. Huang, "On Transmitter Gain/Phase Imbalance Compensation at Receiver," *IEEE Communications on Letters*, Vol. 4, No. 11, pp. 363-365, November 2000.

[29] J. J. de Witt and G. van Rooyen, "A Blind I/Q Imbalance Compensation Technique for Direct-Conversion Digital Radio Transceivers," *IEEE Transactions on Vehicular Technology*, Vol. 58, Issue 4, pp. 2077-2082, May 2009.