Framework of Real-Time Optical Nyquist-WDM Receiver using Matlab & Simulink

By

Adam Vincenzo Crifasi

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

_____
Chairperson Dr. Rongqing Hui

_____
Dr. Shannon Blunt

_____
Dr. Erik Perrins

Date Defended: June 3, 2013

The Thesis Committee for Adam Crifasi

certifies that this is the approved version of the following thesis:

Framework of Real-Time Optical Nyquist-WDM Receiver using Matlab & Simulink

_____

Chairperson Dr. Rongqing Hui

Date approved: June 3, 2013

I investigate an optical Nyquist-WDM Bit Error Rate (BER) detection system. A transmitter and receiver system is simulated, using Matlab and Simulink, to form a working algorithm and to study the effects of the different processes of the data chain. The inherent lack of phase information in the N-WDM scheme presents unique challenges and requires a precise phase recovery system to accurately decode a message. Furthermore, resource constraints are applied by a cost-effective Field Programmable Gate Array (FPGA). To compensate for the speed, gate, and memory constraints of a budget FPGA, several techniques are employed to design the best possible receiver. I study the resource intensive operations and vary their resource utilization to discover the effect on the BER. To conclude, a full VHDL design is delineated, including peripheral initialization, input data sorting and storage, timing synchronization, state machine and control signal implementation, N-WDM demodulation, phase recovery, QAM decoding, and BER calculation.

In loving memory of Kyle

Thank you Lindsey, Mom, and Dad for all the love and support

Thank you Sydney for your encouragement

Table of Contents

Introduction 9
Fiber Optic Communications 9
FPGAs in Communications Systems 10
Background 12
General Properties of Fiber Optic Communications 12
Optical System using Coherent Detection 12
Coherent Detection with Phase Diversity 16
Digital-Subcarrier Multiplexing (DSCM) 17
General Communications and Applicable Digital Signal Processing 22
Anti-Alias Filter 22
Digital Filtering 23
Viterbi-Viterbi Algorithm 27
Symbol Recovery 28
Data Start Sequence 36
Preliminary Simulation, Simulink, and Post-Processing 37
Matlab Simulink Components 37
Normalizer 39
IQ_filter 41
Phase_Recovery 47
QAM_PLL 50
QAMDemod 53
Ninety Degree Shift 55
Barker Checker & Data Checker 60
Simulation Execution 62
ReceiverSimulink.m 63
Data and Clock Timing 68
TX_Func.m 71
DataIn1_maker.m 73
chebysfilter.m 75
Simulink Simulation Results 77
Post-Processing of Data from a Real Fiber-Optic System 87
Post-Processing Results 90
Hardware Implementation Perspective 95
ADC12D1600RB 95
ADC12D1600 97
Xilinx Virtex-4 (XC4VLX25) 99
PLL (LMX2531 LQ1570E) 100
BLP-550+ 102
Hardware Implementation 104
VHDL-only Components 105
dclk_dcm.vhd 105
clk100_dcm.vhd 105
genericcounter.vhd 105
PLL_init.vhd 106
adcDATAinterface.vhd 107
ReceiverFinal.vhd 117
TopLevelDesign.vhd 122
Hardware Results 124
Future Work and Possible Improvements 125
Appendix 126
A. Digital PLL with Proportional plus Integrator loop filter derivation ... 126
B. Bilinear Transformation 127
C. ReceiverSimulink.m 128
D. ReceiverPP.mat 134
E. BER Data 140
Works Cited 144

Figure 1 - Comparison of External Lightwave Modulators (Kikuchi K. , 2010) 12
Figure 2 - Spectrum (left) and time pulse (right) for N-WDM (Gabriella Bosco, 2010) 18
Figure 3 - Spectrum (left) and time pulse (right) for OFDM (Gabriella Bosco, 2010) . 18
Figure 4 - Nyquist filter transfer function and spectrum at the output (Gabriella Bosco, 2010) 19
Figure 5 - Multiplexed N-WDM (Gabriella Bosco, 2010) 21
Figure 6 - Analog Anti-Alias Filter Example 22
Figure 7 - Interpolation Filter Example (8x) – Blue: Magnitude Response – Green: Phase Response 24
Figure 8 - Random Phase Shift Example – Blue symbol sent – Green symbol received ... 27
Figure 9 - Symbol Timing Example (Rice, 2009) 29
Figure 10 - Eye Diagram Derivatives (Rice, 2009) 30
Figure 11 - Symbol Recovery Block Diagram (Rice, 2009) 31
Figure 12 – Linearized Frequency Domain Phase Equivalent PLL (Rice, 2009) 32
Figure 13 - (above) discrete time PLL and DDS; (below) linearized phase equivalent . 33
Figure 14- Phase detector gain for raised cosine pulse 36
Figure 15 – Random signal + length 13 Barker code correlated with length 13 Barker code 37
Figure 16 - Simulink Receiver System Configuration 38
Figure 17 - Normalizer Simulink Model 39
Figure 18 - IQ_Filter Simulink Model 41
Figure 19 - Interpolated Match Filter Responses vs Channel Spacing 44
Figure 20 – Derivative Filter Frequency Response 46
Figure 21 - Phase Recovery Simulink Model 47
Figure 22 - "(a + jb)^4" Simulink Subsystem 47
Figure 23 – Summing Simulink Subsystems 48
Figure 24 - QAM_PLL Simulink Model 50
Figure 25 - Sign Alter Simulink Subsystem 50
Figure 26 - Modulo -1 Simulink Subsystem 51
Figure 27 - QAM Demod Simulink Model 53
Figure 28 - Transmit QAM Constellation 54
Figure 29 - Ninety Degree Shift Simulink Model 55
Figure 30- Constellation Correction Subsystem 57
Figure 31 - QAM Constellation Correction 58
Figure 32 - Barker Checker and Data Checker Simulink Model 60
Figure 33 - Error Counter Subsystem 61
Figure 34 - Integration of Simulink Models into a Matlab Simulation 63
Figure 35 – Measured Analog Anti-Aliasing Filter Response 64
Figure 36 - Data and Clock Timing 68
Figure 37 - Simulink Clock Configuration 70
Figure 38 – Baseband Subchannel Input, Nyquist filter, and Transmitter Output Spectra 73
Figure 39 - Anti-Aliasing Filter Demonstration 76
Figure 40 - Simulink Simulation Spectra Comparison (above: full; below: zoomed in) . 77
Figure 41 - QAM PLL Shift 79
Figure 42 - Scatter Plot Comparison 80
Figure 43 - BER vs Match Filter Order (N) 81
Figure 44 - BER vs N vs Channel Spacing 82
Figure 45 - BER vs % Analog BW vs V&V Ratio 84
Figure 46 - BER vs Input Phase Angle vs V&V Ratio 86
Figure 47 - Test Setup 88
Figure 48 - Resource Savings with Higher Overclocking Rates 89
Figure 49 - Input & Filtered Data vs Time 90
Figure 50 - Filtered and Differentiated Signals vs Time 91
Figure 51 – Input & Filtered Data (dB) vs Frequency (GHz) 91
Figure 52 - QAM PLL Shift Amount 92
Figure 53 - V&V and QAM PLL Constellations vs Input Data (125% Channel Spacing) 93
Figure 54 - V&V and QAM PLL Constellations vs Input Data (115% Channel Spacing) 93
Figure 55 - Comparison of Real and Simulated Data 94

Figure 56 - ADC12D1600RB Top Level Design 95
Figure 57 - ADC12D1600 Functional Block Diagram 97
Figure 58 – Single Channel Analog Input Configuration 98
Figure 59 - LMX2351LQ1570E Functional Block Diagram 100
Figure 60 - Top Level Block Diagram 104
Figure 61 - Serial Interface Diagram 107
Figure 62 - Single Channel adcDATAinterface.vhd Block Diagram 107
Figure 63 - Single Channel demux_fifo_sort.vhd Block Diagram 108
Figure 64 - Dual Channel DataSort.vhd Block Diagram 111
Figure 65 - DataSort.vhd State Machine 113
Figure 66 - ReceiverFinal.vhd Block Diagram 117
Figure 67 - ReceiverFinal.vhd State Machine 119
Figure 68 - TopLevelDesign.vhd Block Diagram 122

Table 1 - Interpolated Match Filter Specifications vs Channel Spacing..... 43
Table 2 - Length 11 Barker Code..... 56
Table 3 - Barker code and Filter Coefficient Comparison..... 56
Table 4 - Phase Shift MUX's Alterations..... 58
Table 5 - Triggers vs MUX Selections..... 58
Table 6 - Length 13 Barker Code..... 60
Table 7 - Key Specifications of the ADC12D1600..... 98
Table 8 - Virtex 4 LX Model Comparison..... 99
Table 9 - Key Specifications of the BLP-550+..... 102
Table 10 - Data Sort Order..... 110
Table 11 - DataSort.vhd State Machine Events..... 113
Table 12 - DataSort.vhd Control Signals..... 115
Table 13 - ReceiverFinal.vhd Block Diagram..... 119
Table 14 - ReceiverFinal.vhd Control Signals 1..... 120
Table 15 - ReceiverFinal.vhd Control Signals 2..... 120
Table 16 - FDRSE_reset Logic Sequence..... 123

Introduction

Fiber Optic Communications

The world of networking relies on a mixture of wireless, copper, and fiber optics communication media to transfer data from anywhere to everywhere. No one medium can conquer the whole task and each provides advantages and disadvantages. Wireless links provide a means to communicate without the physical infrastructure needed with copper or fiber, but are restricted by the structured frequency bands and suffer from extra signal degradation. Copper provides a direct electrical link with ease of transmission and receiving, but has mediocre relative bandwidth capabilities and suffers from electromagnetic interference. Fiber optics boast the most advantageous potential of them all, but also has its faults.

Fiber optics are a necessity in this modern world of communications. Their extreme bandwidth capabilities and electrical isolation represent their most important features. Telecom providers continue to push the data rate envelope to reduce the number of fibers needed, utilize modern networking equipment, and deliver their users the bandwidth they crave. Power utilities, among others, enjoy the safety of a communication link that does not carry fault current.

The process of creating and demodulating a light signal is similar, yet unique, when using a light source. The complex modulation schemes require expensive equipment and precise considerations. Variances in the forces laying on a fiber, for example, can alter the lightwave within the fiber. Chromatic and modal dispersion inherently smear the signal

inside the core. But companies around the world will continue to utilize fiber above the other options due to the bandwidth available.

FPGAs in Communications Systems

The efficiency of communication systems has rapidly improved over the years, but with the price of increased system more complexity. More data can be packed into smaller bandwidth by clever signal processing. Higher order systems need precision equipment to sample and decode these signals. Technology is continuously striving to chase the requirements of the internet, cell phone usage, and other data demand.

Networking hardware has to process a lot of data while minimizing latency. This requires many repetitive and parallel processes. All computers can accomplish repetitive tasks; that's what they were made for. But many computer architectures are not suited for parallel tasks. For instance, microcontrollers (MCUs) can execute multiplication functions in milliseconds, but can only process one operation at a time. Two multiplication operations take approximately twice as long as one multiplication. This is where FPGAs excel.

Commercial communication equipment usually employ application-specific integrated circuit (ASIC) for high reliability, low power consumption and miniature footprints. However, the cost of ASIC development is high and the design process is long, and therefore it is usually not used for prototyping. In comparison, a Field Programmable Gate Array (FPGA) offers increased flexibility, as its a chip containing massive amounts of gates which can be arranged in any fashion. Along with the embedded hardware are

hundreds of pins which can be programmed as inputs or outputs. Whereas all operations have to process through a common core in a MCU, the logic on an FPGA can take any form, be completely independent of one another, and even belong to different clock domains. Modern FPGAs contain more than programmable AND and OR gates, built in modules help ease the amount of programming and generate a better product. Most FPGAs, for instance, have clock controllers to ensure the sequential logic performs as expected.

Digital signal processing (DSP) almost always consists of repetitive tasks. Many times, there are separate channels which require the exact same processing. The FPGA hardware and programming architecture thrive at parallel processing, and thus parallel pipelines can execute the required process on several channels at once, depending on gate availability. Each parallel process can be seen as a multiplier to the system's clock speed. The FPGA's ability to run at already high clock speeds coupled with the parallel multiplier means the chip can provide a developer with the right tool to create the fastest, most robust systems.

Background

General Properties of Fiber Optic Communications

Optical System using Coherent Detection

To convert an electrical signal into the optical domain and send into fiber optic systems, the laser itself can be modulated directly or an external modulator can act upon the launched light. Only amplitude modulation (AM) can come of modulating the laser itself. An external modulator can modulate a light signal a number of ways: AM, phase modulation (PM), or more complex modulation schemes (Kikuchi K. , 2010).

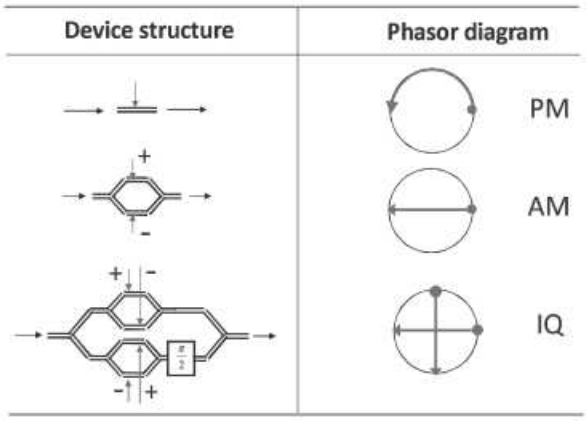


Figure 1 - Comparison of External Lightwave Modulators (Kikuchi K. , 2010)

An external modulator with a Mach-Zehnder configuration, driven in push-pull mode, can perform AM of a light source. By placing two Mach-Zehnder push-pull modulators in parallel with a 90 degree phase shift between the arms, a complex IQ signal is generated (S. Shimotsu, 2001). The in-phase and quadrature signals now exist independently of each other. With no higher order modulation, the signal information is

doubled within the same bandwidth. Any kind of higher order modulation is possible with this complex modulation setup, just as it is in RF or wireless communications (eg, 16 QAM).

An optical receiver system performing coherent detection in the correct configuration can demodulate a complex signal. The signal must be downshifted from the transmitter carrier frequency. This is done, fundamentally, by taking the product of the electric fields of the signal and the local oscillator. The local oscillator (LO) is a continuous wavelength (CW) light source. Define the modulated optical signal and the local oscillator as:

$$\vec{E}_s(t) = \vec{A}_s(t)e^{j\omega_s t + j\emptyset_s(t)} \quad \text{Equation 1}$$

$$\vec{E}_{LO}(t) = \vec{A}_{LO}e^{j\omega_{LO} t + j\emptyset_{LO}} \quad \text{Equation 2}$$

where A_S and A_{LO} are the complex amplitudes, and ω_s and ω_{LO} are the angular frequencies. Note that the complex amplitude of the LO is constant, whereas the signal's complex amplitude changing with time (Kikuchi K. , 2010). Both the amplitude and the angle of the signal vary with time.

The two outputs of a 2x2 optical coupler mixing the source and local oscillator light waves is given by the following transfer function:

$$\begin{bmatrix} \vec{E}_1 \\ \vec{E}_2 \end{bmatrix} = \begin{bmatrix} \sqrt{1-\varepsilon} & j\sqrt{\varepsilon} \\ j\sqrt{\varepsilon} & \sqrt{1-\varepsilon} \end{bmatrix} \begin{bmatrix} \vec{E}_s(t) \\ \vec{E}_{LO} \end{bmatrix} \quad \text{Equation 3}$$

The square roots describe an energy mixing equation and ε is the power-coupling coefficient (Hui & O'Sullivan, 2009). In a simple setup, one output is terminated and the signal contained in the other fiber is

$$\vec{E}_1(t) = \sqrt{1-\varepsilon}\vec{E}_s(t) + j\sqrt{\varepsilon}\vec{E}_{LO} \quad \text{Equation 4}$$

A photodiode placed at the end of the output produces the current

$$i(t) = R\left|\vec{E}_1(t)\right|^2 \qquad \text{Equation 5}$$
$$= R\left\{(1-\varepsilon)\left|\vec{A}_s(t)\right|^2 + \varepsilon\left|\vec{A}_{LO}\right|^2 + 2\sqrt{\varepsilon(1-\varepsilon)}\vec{A}_s(t)\cdot\vec{A}_{LO}\cos(\omega_{IF}t + \Delta\emptyset(t))\right\}$$

where $\omega_{IF} = \omega_s - \omega_{LO}$, $\Delta\emptyset(t) = \emptyset_s(t) - \emptyset_{LO}$, and R is the responsivity of the photodiode (Hui & O'Sullivan, 2009). The term ω_{IF}, or intermediate frequency, is the difference between the two light source's frequencies and $\Delta\emptyset(t)$ is the time dependent relative phase of the two light waves. Another term, the sum of the two light sources, exists but is ignored due to the fact that the RF circuitry will not couple this signal.

To reduce this equation to a compact form, a few assumptions shall be made. Firstly, the LO is a continuous light source with little to no intensity noise, leaving $\vec{A}_{LO} = \sqrt{P_{LO}}$, a constant. A DC block after the photo diode will remove this component of the output current. The LO light source is assumed to have a much larger amplitude than the incoming signal. Therefore, the $\left|\vec{A}_s(t)\right|^2$ can also be ignored as $\left|\vec{A}_s(t)\right|^2 \ll \vec{A}_s(t)\cdot\vec{A}_{LO}$ (Hui & O'Sullivan, 2009). A 3dB coupler should be used in the coherent receiver since the maximum power output is achieved when $\varepsilon = 1/2$ (Hui & O'Sullivan, 2009). These assumptions produce a simplified coherent detection photocurrent of

$$i(t) \approx R\sqrt{P_s(t)\cdot P_{LO}}\cos(\omega_{IF}t + \Delta\emptyset) \qquad \text{Equation 6}$$

When the LO and signal center frequencies are equal, the system is said to be in homodyne configuration. Therefore, since $\omega_{IF} = \omega_s - \omega_{LO} = 0$, the equation further reduces to

$$i(t) \approx R\vec{A}_s(t)\cdot\vec{A}_{LO}\cos(\Delta\emptyset) \qquad \text{Equation 7}$$

Set up in a homodyne detection scheme, the desired electrical signal from the photodiode exists in baseband, ready to be demodulated by an external system. The

incoming light signal is often very weak. As expressed in Equation 7, the resulting photocurrent benefits from a strong LO to amplify the electrical signal. The relative phase of the LO and the incoming signal will shift, causing the $\Delta\emptyset$ term to be time dependent. Demodulation techniques further down the signal chain must compensate for this fluctuation.

Coherent Detection with Phase Diversity

A coherent receiver in certain arrangements can detect the complex amplitude information from a modulated signal. The setup and theory is very similar to that of standard coherent detection. To achieve coherent detection with complex phase information, the local oscillator must mix with the signal with an in-phase and quadrature (90°) component. This can be achieved with the use of a 90° hybrid. Ideally, the component would be a 2x2 90° hybrid, mimicking the system above exactly, but this configuration is theoretically impossible (Hui & O'Sullivan, 2009). Therefore, a real substitute in the 3x3 90° hybrid can be used instead. The transfer function is as follows:

$$\begin{bmatrix}\vec{E}_1\\ \vec{E}_2\\ \vec{E}_3\end{bmatrix}=\begin{bmatrix}\sqrt{0.2} & \sqrt{0.4}\exp(j\frac{3\pi}{4}) & \sqrt{0.4}\exp(j\frac{3\pi}{4})\\ \sqrt{0.4}\exp(j\frac{3\pi}{4}) & \sqrt{0.2} & \sqrt{0.4}\exp(j\frac{3\pi}{4})\\ \sqrt{0.4}\exp(j\frac{3\pi}{4}) & \sqrt{0.4}\exp(j\frac{3\pi}{4}) & \sqrt{0.2}\end{bmatrix}\begin{bmatrix}\vec{E}_s(t)\\ \vec{E}_{LO}\\ 0\end{bmatrix} \quad \text{Equation 8}$$

where E_3 has no power applied and only two outputs are used. Essentially, the 3x3 coupler is being used as a 2x2 coupler.

Working through similar math as before:

$$\vec{E}_1(t)=\sqrt{0.2}\vec{E}_s(t)+\sqrt{0.4}\exp(j\frac{3\pi}{4})\vec{E}_{LO} \quad \text{Equation 9}$$

$$\vec{E}_2(t)=\sqrt{0.4}\exp(j\frac{3\pi}{4})\vec{E}_s(t)+\sqrt{0.2}\vec{E}_{LO} \quad \text{Equation 10}$$

Continuing after photodetection and ignoring direct detection components:

$$i_1(t)\approx 2R\sqrt{0.08}\vec{A}_s(t)\cdot\vec{A}_{LO}\cos(\Delta\emptyset(t)-\frac{3\pi}{4}) \quad \text{Equation 11}$$

$$i_2(t)\approx 2R\sqrt{0.08}\vec{A}_s(t)\cdot\vec{A}_{LO}\sin(\Delta\emptyset(t)-\frac{3\pi}{4}) \quad \text{Equation 12}$$

Since the two current terms apply the cosine and sine of the time dependent phase difference, they extract perpendicular components of the light signal. Thus, complex information can be demodulated.

Digital-Subcarrier Multiplexing (DSCM)

In order to make efficient use of the low-loss bandwidth of the optical fiber, wavelength division multiplexing (WDM) has been used widely. In a WDM system, multiple wavelengths are used to carry independent data channels along the same fiber. A frequency guard band is required such that crosstalk can be avoided when passing through a WDM de-multiplexing filter. Commercial high speed optical systems with 50 GHz channel spacing, 10 Gb/s data rate per channel, and 0.2 bit/Hz efficiency are widely deployed. Increasing data rates to 40 or 100 Gb/s will improve spectral efficiency, however these systems are sensitive to the impact of chromatic dispersion and polarization mode dispersion (PMD).

To continue to increase the spectral efficiency of fiber optic communication links while maintaining high quality of transmission, several techniques are being explored to combine several subchannels of data into a single optical signal, or wavelength. The subchannels can contain independent sets of data in each channel, or they can be modulated such that the entire channel is a single data stream. In the past, sub-carrier multiplexing schemes have been implemented in fiber-optic systems to partition a high data rate wavelength channel into many subchannels to significantly improve the tolerance to fiber chromatic dispersion and PMD. However, these subcarrier channels were created

with analog techniques, and as such, frequency spacing was still needed and spectral efficiency was not increased compared to traditional WDM technology.

Recent advances in high speed digital electronics enables multi-giga-sample operation of analog to digital converters (ADC), digital to analog converters (DAC), and digital signal processing (DSP). This allows the generation and processing of subcarrier channels in the digital domain, utilizing advanced digital modulation formats and DSP algorithms to dramatically increase spectral efficiency. We'll call this technique Digital Subcarrier Multiplexing (DSCM). Two currently popular DSCM techniques being implemented and studied are Nyquist-WDM (N-WDM) and Orthogonal Frequency Division Multiplexing (OFDM).

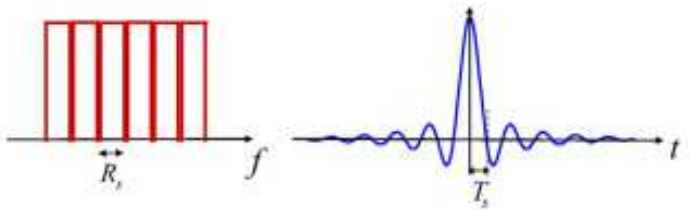


Figure 2 - Spectrum (left) and time pulse (right) for N-WDM (Gabriella Bosco, 2010)

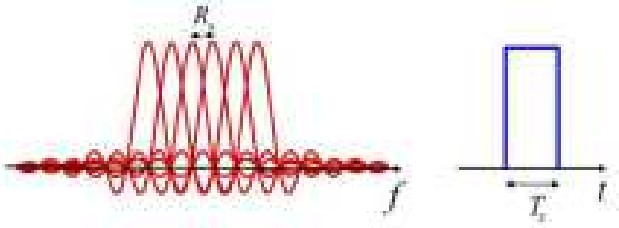


Figure 3 - Spectrum (left) and time pulse (right) for OFDM (Gabriella Bosco, 2010)

N-WDM and OFDM can be thought of as "opposites" in the frequency and time domains. An N-WDM pulse and the spectra of an OFDM subchannel will both, ideally, take the shape of a sinc. The spectra of a single N-WDM channel and an OFDM spectra will both, ideally, take the shape of a perfect square.

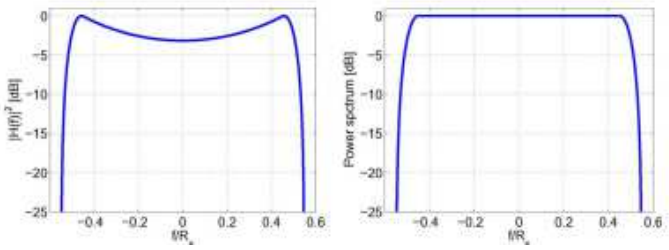


Figure 4 - Nyquist filter transfer function and spectrum at the output (Gabriella Bosco, 2010)

An OFDM signal is generated using the following transmit function (Q. Yang, 2011):

$$s(t) = \sum_{i=-\infty}^{+\infty} \sum_{k=1}^{N_{sc}} c_{ki} s_k (t - iT_s) \qquad \text{Equation 13}$$

$$s_k(t) = \mu(t) e^{j2\pi f_k t} \qquad \text{Equation 14}$$

$$\mu(t) = \begin{cases} 1, & (0 < t \leq T_s), \\ 0, & (t \leq 0, t > T_s) \end{cases} \qquad \text{Equation 15}$$

where c_{ki} is the ith symbol of the kth subcarrier, s_k is the waveform of the kth subcarrier, the number of subcarriers is denoted by N_{sc}, f_k is the subcarrier center frequency, and T_s is the symbol period. The last equation, $\mu(t)$, is the pulse shaping function. Rewriting the expressions, sampling the signal at N/T, and normalizing the equation by 1/N yields:

$$S_n = \frac{1}{N} \sum_{i=0}^{N-1} A_i e^{j2\pi \frac{i}{N} n}, n = 0,1, \dots, N-1 \qquad \text{Equation 16}$$

where S_n is the nth time domain sample. Clearly, this expression is the same as an inverse discrete Fourier transform (IDFT) (Q. Yang, 2011). Similarily, to demodulate, a discrete Fourier transform is used:

$$A_i = \sum_{i=0}^{N-1} R_n e^{-j2\pi \frac{i}{N} n}, n = 0,1, \dots, N-1 \qquad \text{Equation 17}$$

where R_n is the received signal.

An OFDM system requires sampling of the entire spectrum of all the subchannels to be able to process the data of a certain subchannel (Junyi Wang, 2012). This is evident by the OFDM spectrum presented in Figure 3 as well as the IDFT modulation function. The breadth of research and algorithms to implement discrete Fourier functions can optimize the resource usage, or throughput speed, whichever is more crucial to the project.

The formation of a Nyquist-WDM transmit signal need only be filtered by an inverse-sinc filter set to the bandwidth of the subchannel, with a certain roll-off rate (such as 0.1). This allows the subchannels to have a flat spectral shape while minimally overlapping with adjacent subchannels. Ideally, the transmitter utilizes a filter with the number of taps equally that of the length of the signal (if packet based). A time domain or frequency domain filtering method can be chosen by the user based on ease of implementation.

The full N-WDM channel can be created several ways depending on the requirements of the system. If all the signals originate in the same machine which is applying the digital filters, The filters can all be applied, as well as the subchannels shifted into their allotted frequency range, in one operation. If the different subchannels originate from several different sources, each signal can have a baseband filter applied, and the separate signals shifted into their slots, as completely independent operations. See Figure 5 for a diagram of the multiplexing scheme.

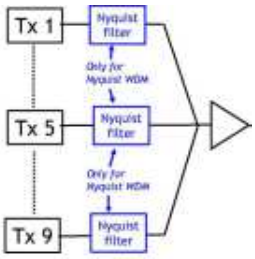


Figure 5 - Multiplexed N-WDM (Gabriella Bosco, 2010)

A clear advantage of Nyquist-WDM is the fact that a single subcarrier can be a separate entity of the signal (Junyi Wang, 2012). A subchannel can be added into an existing signal if that particular frequency slot is currently unoccupied. Also, a single channel of a Nyquist-WDM system can be sampled and demodulated independently: the entire N-WDM channel does not need to be sampled. Therefore, an intermediate router could drop one subchannel and add another to an existing N-WDM spectrum without disturbing adjacent subchannels. When considering a real system, N-WDM can extract a single channel using lower bandwidth ADC, saving money. This project studies the technique of N-DWM, the effects of the parameters in the receiver chain, and the tradeoffs of a realistic, cost effective build.

General Communications and Applicable Digital Signal Processing

Anti-Alias Filter

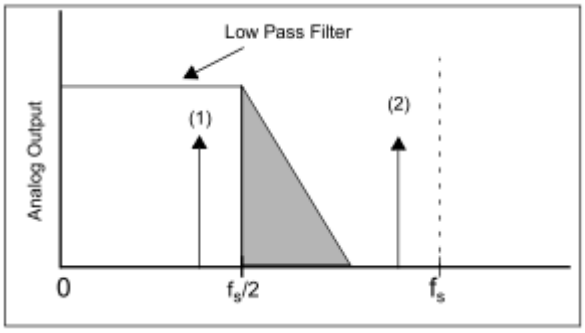


Figure 6 - Analog Anti-Alias Filter Example

If high spectral power exists beyond the Nyquist sampling rate of the ADC, an analog anti-aliasing filter is required (Baker, 1999). The chosen filter needs to sufficiently attenuate the out-of-band signals such that they will not interfere with the desired signal. Since the downshifted signal exists at baseband in this project, a lowpass filter is required for signals beyond the Nyquist frequency. For some systems, it is reasonable to employ a filter which has a passband with the same bandwidth as the Nyquist rate, attenuating undesirable signals beyond the Nyquist rate, as seen in Figure 6. This project, though, must demodulate a single subchannel from a very cluttered spectra. There is no guard-band in a perfect N-WDM system. Therefore, due to the specifications of real analog filters, the passband of the filter should occupy only a fractional portion of the 1st Nyquist region. This allows room for the rolloff of the filter's response from passband to Nyquist rate.

Digital Filtering

Filtering is a fundamental aspect of communications. Filtering can, and must, be done in both the analog domain and in the digital domain. There are many advantages to filtering after the signal has been digitized. Analog filters have many non-ideal qualities: dependent on temperature, being produced with imperfect parts, taking up physical space, having relatively poor frequency rolloff, etc. Digital filters overcome these setbacks and add the advantage of programmability, easy integration in digital systems, ability to produce nearly any response, ability to adapt to incoming signals, and much more. There are inherent disadvantages in digital filering: quantization, complexity, and possible instability. The advantages clearly outweigh the disadvantages in this digital world and digital filtering is implemented everywhere it can be. Of course, it is not always an option, as with anti-aliasing filter above.

Digital systems most commonly implement linear filters. As the name implies, a filter is considered linear if the output signal is a linear function of the observations applied to the filter input (Haykin, 2002). The linear filter family splits into infinite impulse response (IIR) and finite impulse response (FIR). Simply put, the difference between the two is IIR filters apply feedback, whereas FIR filters contain only a feedforward path. Therefore, the response of each impulse subject to the IIR filter infinitely exists, whereas the response of the FIR filter exists only in the set number of taps.

The two filter types have many different features and uses. IIR filters can produce much sharper responses with fewer coefficients than FIR filters. FIR filters, though, have the advantage of always being stable, whereas IIR filters can have instability problems. IIR

filters are based off of analog filters, and thus easier to design; FIR filters are difficult to design without CAD support. A very important feature for communications is that FIR filters can have exactly linear phase.

Two specific forms of FIR filters important to this research is the interpolation match filter and the derivative filter. The interpolation match filter is essentially the combination of two separate filters to achieve an ultimate goal. As a linear system, the filters can be implemented in series in either order or, if possible, as a single combination. An interpolation filter's job is to remove the undesired spectral replications of the signal.

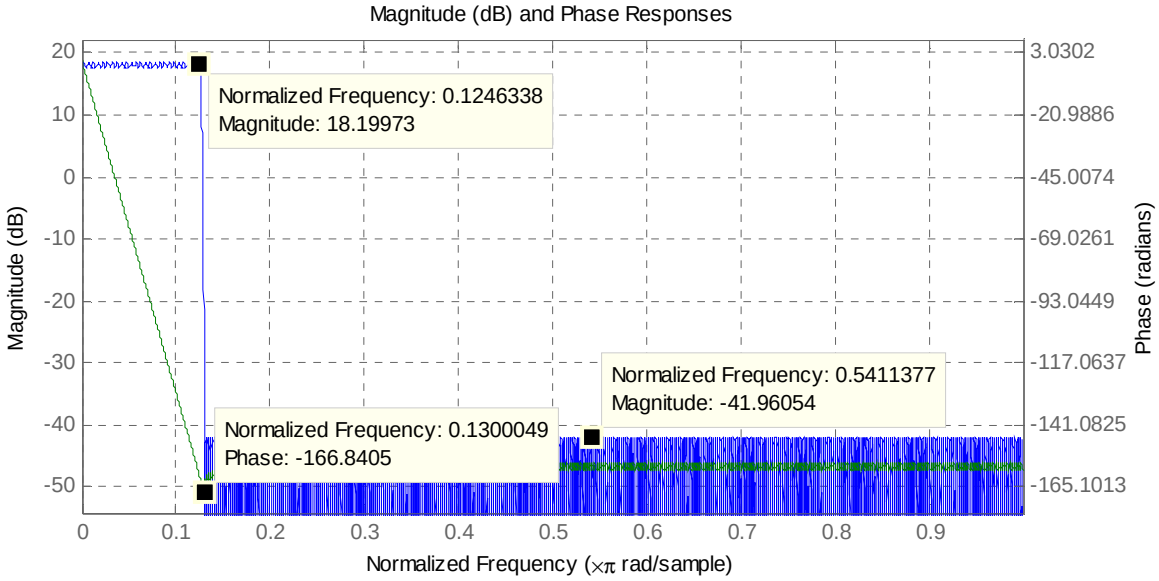


Figure 7 - Interpolation Filter Example (8x) – Blue: Magnitude Response – Green: Phase Response

Figure 7 displays an example of an 8x interpolation filter. There are several key properties to notice. Firstly, the cutoff of the filter lies at 0.125, or 1/8 (Ifeachor & Jervis, 2001): the frequency of the signal decreased by 8 in the digital domain. It is important to add, though, the frequency of the signal has not actually changed; only its digital representation. A digital signal can take many forms based on the designer's requirements.

Secondly, the amplitude of the passband amplifies the signal by ~18.06 dB. This factor compensates for the loss of power during the interpolation process. Assuming the input signal is a voltage signal, the amplification is found by:

$$A = 20\log_{10}(N)\,[dB] \qquad \text{Equation 18}$$

where N is the interpolation factor. Next, the phase response is linear throughout the passband. Since the phase information is equally as important as the magnitude in complex modulation, linear phase is required. The phase in the stop band needs not be linear, because the demodulator doesn't use the higher frequency information.

The Matlab "filter builder" tool created this filter, and all the filters henceforth in this project. This particular filter, in Figure 7, used the "Lowpass Design" with an "Interpolation Factor" of 8. The passband and stopband were specified at .125 and .13, respectively, with a passband ripple of 1 dB and stopband attenuation of 60 dB. The figure shows the stopband attenuated by 42 dB, but relative to the 18 dB amplification of the passband due to the interpolation, this equates to a 60 dB difference.

The above example is a simple interpolation filter with the passband including the entire original signal. A match filter is a single rate filter which selects a certain band, depending on the desired signal. An interpolation match filter combines both these concepts. The desired passband is scaled down and contained within the passband of the interpolation filter. Simply divide the passband and stopband frequencies of the match filter specifications by the interpolation factor to calculate the associated frequencies in the interpolated match filter.

Another important filter in this project is the derivative filter. This filter can also be implemented by an FIR filter. The following provides a quick proof on how the transfer function of the derivative is derived.

$$y(n) = \frac{d}{dn}x(n) \qquad \text{Equation 19}$$

$$F\{y(n)\} = j\Omega \cdot X(\Omega) \qquad \text{Equation 20}$$

$$Y(\Omega) = j\Omega \cdot X(\Omega)$$

$$\therefore H(\Omega) = j\Omega \rightarrow h(n) = \frac{\cos(\pi n)}{n} \qquad \text{Equation 21}$$

where Ω is the digital frequency component and F{∘} is the Fourier transform operator. This function will return the derivative of the entire signal. Although the coefficients of this filter have a relatively simple function in this form, it becomes much more complicated in application. Usually the derivative of only a particular band is desired, as it is in this project. Therefore, Matlab design tools are employed to create the derivative filter.

A final note on Matlab filter builder design tools; the weight vector can save the number of taps and better tailor the filter. The filter design weight vector allows the user to specify which components of the transfer function are most important, and thus, the design tools should devote resources to. This is particularly helpful when there's a predefined number of taps available. For example, consider a lowpass filter. There are two critical components, outside of the frequency choices: passband ripple and stopband attenuation. A user could choose to define a weight vector (in the same order) of [1, .2] if the passband response must be flat, but does not require a significant amount of attenuation in the stopband.

Viterbi-Viterbi Algorithm

A 4-QAM signal is constructed with a certain, predetermined phase. The QAM receiver is programmed to demodulate the signal, expecting a certain phase as well. The phase can differ depending on design requirements or preference. However, during the transmission and downshifting processes the received signal will suffer arbitrary phase shifts. Recall Equation 7, the output current from the photodetector after homodyne coherent detection.

$$i(t) \approx R\vec{A}_s(t) \cdot \vec{A}_{LO}\cos(\Delta\emptyset)$$

The $\Delta\emptyset$ term represents this random phase shift. Figure 8 visualizes this random phase shift. The blue circles represent the transmitted QAM signal, and the green circles have experienced a phase change. The constellation must be realigned to the axes for the demodulator circuit to effectively do its job.

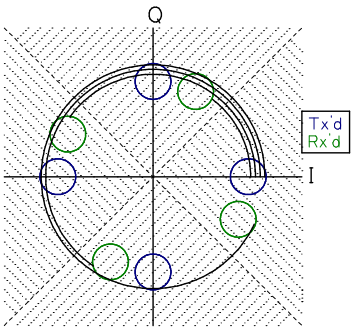


Figure 8 - Random Phase Shift Example – Blue symbol sent – Green symbol received

The Viterbi-Viterbi algorithm solves this problem. The first step is to estimate the phase. Although the equation is applied in one step, it will be explained as a series of operations. First, the phase of the incoming signal must be estimated (Kikuchi K. , 2011). The phase modulation should be removed by applying the M-power to the incoming

complex signal, where M-ary is the PSK format. The phase estimation is then averaged over $2k+1$ samples to improve SNR of the estimated phase. Then apply the "angle" (or arg()) function to the complex average. This function is basically an arctangent function with an interval over [0,2π). Finally, divide the angle by M to remove the power of M previously applied. The phase has now been estimated (Equation 22). Now subtract that phase error from the signal by complex multiplying the signal and the phase error, after negating the imaginary term.

$$\theta_e(n) = \arg\left(\sum_{j=-k}^{k} E(n-j)^M\right)/M \qquad \text{Equation 22}$$

In this paper, the "Viterbi-Viterbi ratio" (V&V ratio) is equal to the number of samples to be averaged divided by the number of samples per symbol. For instance, if the received signal (after interpolation) has 10 samples per symbol and the phase estimation algorithm averages over 51 samples (k=25), the Viterbi-Viterbi ratio is 5.1.

The phase change also has a 2π/M ambiguity (Kikuchi K. , 2011). Proper techniques must be utilized at the transmitter and receiver to combat these effects. A discussion of the method to overcome this barrier takes place in the implementation section of this paper.

Symbol Recovery

Once the phase has been corrected the system must lock onto to the signal clock and perform symbol synchronization. This process' goal is to resample the signal at peak values of the received signal without the benefit of a dedicated clock signal. Some communication systems forfeit some bandwidth to transmit a data clock signal in order to simplify the receivers. In communication systems where bandwidth is at a premium and

cannot afford to sacrifice any for a clock signal, extra circuitry is required to synchronize the receiver with the data clock.

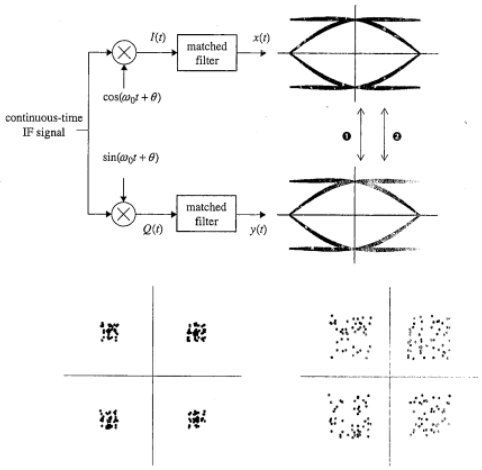


Figure 9 - Symbol Timing Example (Rice, 2009)

The above example shows the importance of proper clock synchronization. A QPSK signal enters the circuit from the left where it is downshifted by an in-phase and quadrature source and match filtered. An ideal eye diagram emerges from the filters. The constellation on the left was sampled at (1) arrow, and the constellation on the right corresponds with the (2) arrow. An approximate 15% symbol length shift dramatically deteriorates the constellation. Even with a clean, filtered signal, a bad symbol recovery circuit can ruin an otherwise good receiver.

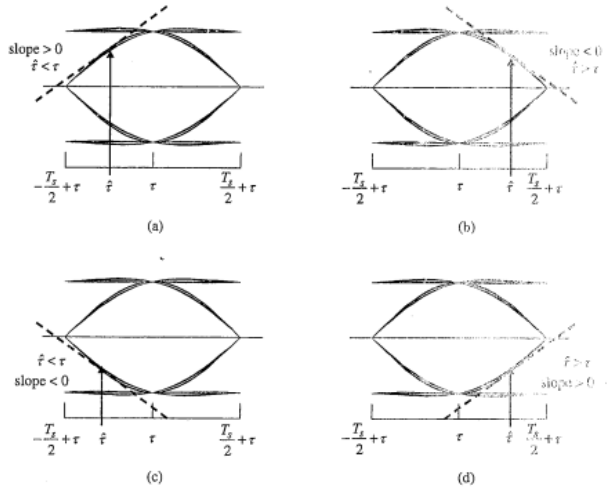


Figure 10 - Eye Diagram Derivatives (Rice, 2009)

If a signal is being reclocked at the wrong time, one of the 4 following situations exists. Eye diagram (a) represents the timing occurring too early on the rising portion of the signal, (b) shows a late timing situation on the down slope, (c) is being clocked early on the downslope, and (d) is late on the upslope. The derivative of the signal clearly indicates the timing error, with larger magnitudes as the point drifts further from the center. This fact can be exploited and used in a feedback system to align the timing circuit. The indicator should have a positive sign if the timing is occurring early to increase the sampling time and negative when late. As it stands, the two early and late situation derivatives have opposite signs. To correct this, the sign of the signal can be multiplied to the derivative. Therefore, situation (a) and (c) produce a positive error and (b) and (d) produce a negative. Now these errors can be utilized to synchronize the system with the signal.

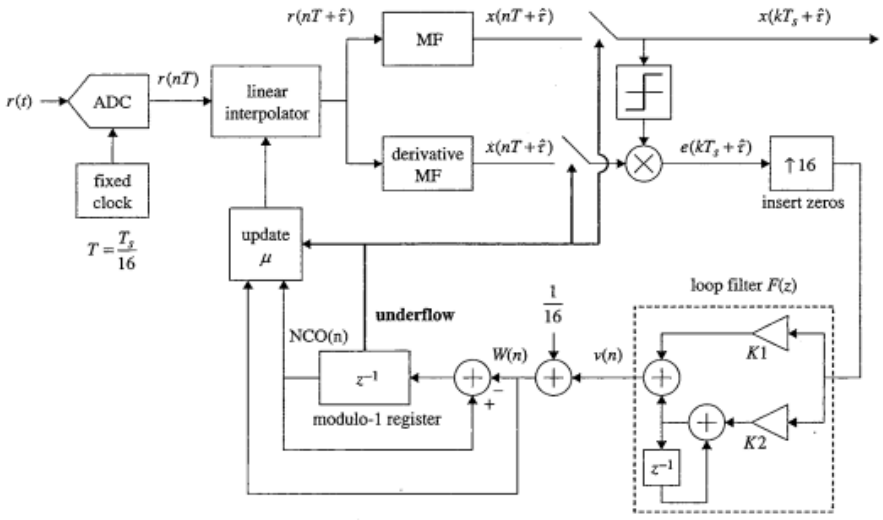


Figure 11 - Symbol Recovery Block Diagram (Rice, 2009)

First, a quick walkthrough of the circuit, followed by a more detailed explanation of the some of the components. The analog signal must be sampled by an ADC with a fixed clock source. A linear interpolator adds additional data points per symbol period. The interpolated signal feeds into a matched filter as well as a derivative filter. The resampling process takes place at this step, as directed by the update block. The derivative signal is multiplied to the sign of the match filter signal to adjust the sign of the derivative, as previously discussed. The error signal is zero padded by 16, the interpolated points per period in this example. This ensures the feedback loop only receives one error pulse per period. The proportional-plus-integrator filter allows the circuit to be tuned to a specific input signal. The processed error, *v(n)*, is added to a constant 1/16 factor. This term, unaltered by *v(n)*, would obviously accumulate to 1 after one symbol period. A modulo-1 register follows another summation, which also provides feedback. The update block processes the data from the modulo-1 register and the fixed summation and triggers the resampling process.

The most crucial part of this, and most, feedback systems is the loop filter. Many configurations exist, but the proportional-plus-integrator filter is attractive due to its balance of simplicity and effectiveness, producing an a steady state error of 0 for both the step and ramp function.

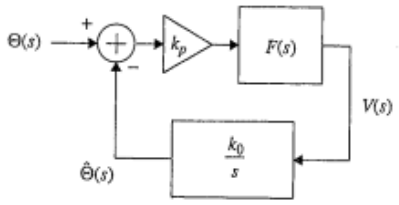


Figure 12 – Linearized Frequency Domain Phase Equivalent PLL (Rice, 2009)

Figure 12 represents the model used to derive the gain coefficients. The phase estimate, $\hat{\theta}(s)$, is the output of the PLL. Let's being at *V(s)* to form the transfer function of the closed loop:

$$V(s) = k_p F(s)\left(\theta(s) - \hat{\theta}(s)\right) = \frac{s}{k_0}\hat{\theta}(s) \qquad \text{Equation 23}$$

$$\hat{\theta}(s) * \left(s + k_0 k_p F(s)\right) = k_0 k_p F(s)\theta(s) \qquad \text{Equation 24}$$

$$H_a(s) = \frac{\hat{\theta}(s)}{\theta(s)} = \frac{k_0 k_p F(s)}{s \mp k_p F(s)} \qquad \text{Equation 25}$$

The continuous time function of a proportional-plus-integrator filter leads to the final closed loop transfer function:

$$F(s) = k_1 + \frac{k_2}{s} \qquad \text{Equation 26}$$

$$\therefore H_a(s) = \frac{k_0 k_p k_1 s + k_0 k_p k_2}{s^2 + k_0 k_p k_1 s + k_0 k_p k_2} \qquad \text{Equation 27}$$

$$H_a(s) = \frac{2\zeta\omega_n s + {\omega_n}^2}{s^2 + 2\zeta\omega_n s + {\omega_n}^2}$$

Equation 28

$$\zeta = \frac{k_1}{2}\sqrt{\frac{k_0 k_p}{k_2}}$$

where $$w_n = \sqrt{k_0 k_p k_2}$$

The discrete time PLL is essentially is same as the continuous time model (Rice, 2009). An analog phase detector and loop filter are replaced by their discrete counterparts, and a direct digital synthesizer (DDS) replaces the VCO. The design of a digital PLL typically begins by transforming a continuous time version PLL to produce a discrete-time version (Rice, 2009). The process again starts by forming the transfer function from the linearized closed loop system. The two systems can be seen below in Figure 13.

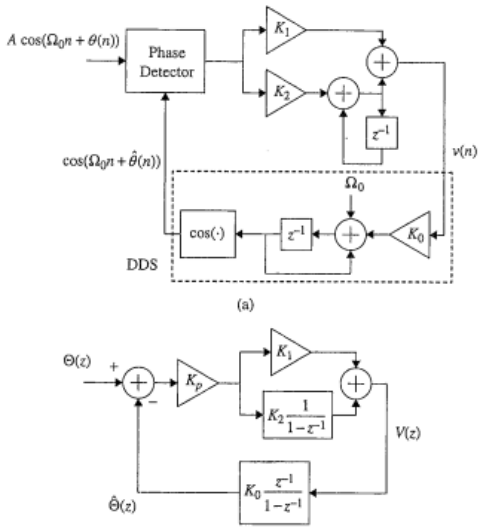


Figure 13 - (above) discrete time PLL and DDS; (below) linearized phase equivalent

Similarly as before, a transfer function can be derived from the system. A more complete derivation can be found in Appendix A.

$$V(z) = K_p F(z)\left(\theta(z) - \hat{\theta}(z)\right) = \frac{1}{K_0}\frac{1 - z^{-1}}{z^{-1}}\hat{\theta}(z) \quad \text{Equation 29}$$

$$\hat{\theta}(z) * \left(\frac{1 - z^{-1}}{z^{-1}} + K_0 F(z)\right) = K_0 F(s)\theta(z) \quad \text{Equation 30}$$

$$H_d(z) = \frac{\hat{\theta}(z)}{\theta(z)} = \frac{K_0 F(s)\theta(z)}{\frac{1 - z^{-1}}{z^{-1}} + K_0 F(z)} \quad \text{Equation 31}$$

$$H_d(z) = \frac{K_p K_0 (K_1 + K_2) z^{-1} - K_p K_0 K_1 z^{-2}}{1 - 2(1 - \frac{1}{2} K_p K_0 (K_1 + K_2)) z^{-1} + (1 - K_p K_0 K_1 z^{-2})} \quad \text{Equation 32}$$

The digital gain factors, K_0, K_1, K_2, and K_p, can be found by incorporating the continuous time loop filter equation (Equation 26). Applying the bilinear transform to the continuous time transfer function produces a relation in the denominators that can be exploited. The full transformation can be found in Appendix B, resulting in the following digital representation of the continuous system:

Equation 33

$$H_a\left(\frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}\right) = \frac{\frac{2\zeta\theta_n + {\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2} + 2\frac{2{\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2} z^{-1} + \frac{-2\zeta\theta_n + {\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2} z^{-2}}{1 - 2\frac{1 - {\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2} z^{-1} + \frac{1 - 2\zeta\theta_n + {\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2} z^{-2}}$$

where $$\theta_n = \frac{w_n T}{2}$$

Setting the coefficients of the $H_a(s)$ (Equation 32) and $H_d(z)$ (Equation 33) polynomials equal yields:

$$1 - \frac{1}{2} K_p K_0 (K_1 + K_2) = \frac{1 - {\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2} \quad \text{Equation 34}$$

$$1 - K_p K_0 K_1 = \frac{1 - 2\zeta\theta_n + {\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2}$$ Equation 35

Some simple algebra solves for the loop constants:

$$K_p K_0 K_1 = \frac{4\zeta\theta_n}{1 + 2\zeta\theta_n + {\theta_n}^2}$$ Equation 36

$$K_p K_0 K_2 = \frac{4{\theta_n}^2}{1 + 2\zeta\theta_n + {\theta_n}^2}$$ Equation 37

The previous and following equations contain these variables: dampening ratio ζ, natural frequency ω_n, sampling period T, and equivalent noise bandwidth B_n. Substituting the θ_n term for a proportional plus integrator filter produces the final result for the gain factors (Rice, 2009):

$$\theta_n = \frac{B_n T}{\zeta + \frac{1}{4\zeta}}$$

$$K_p K_0 K_1 = \frac{4\zeta\left(\frac{B_n T}{\zeta + \frac{1}{4\zeta}}\right)}{1 + 2\zeta\left(\frac{B_n T}{\zeta + \frac{1}{4\zeta}}\right) + \left(\frac{B_n T}{\zeta + \frac{1}{4\zeta}}\right)^2}$$ Equation 38

$$K_p K_0 K_2 = \frac{4\left(\frac{B_n T}{\zeta + \frac{1}{4\zeta}}\right)^2}{1 + 2\zeta\left(\frac{B_n T}{\zeta + \frac{1}{4\zeta}}\right) + \left(\frac{B_n T}{\zeta + \frac{1}{4\zeta}}\right)^2}$$ Equation 39

The K_1 and K_2 gain factors can now be calculated for the symbol recovery system. The dampening ratio is set to critically dampening at $1/\sqrt{2}$. For the modulo-1 system, K_0 is set to -1. The noise bandwidth, B_nT and phase detector gain, K_p, depend on the specific application. The noise bandwidth factor must be tuned to the specific signal and its expected noise power. The value K_p can be extracted from the following chart:

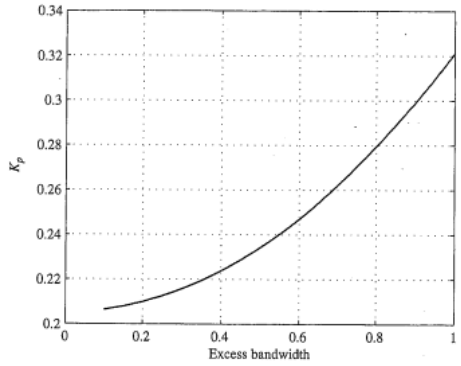


Figure 14- Phase detector gain for raised cosine pulse

Data Start Sequence

A crucial function of a communications system is determining the start of a data block. When processing the actual 1's and 0's being received after demodulation, the system must be able to identify the beginning of a string of data. One such tactic involves placing a certain, known data pattern at the start of the block and constantly scanning for the pattern. When the receiver detects the pattern it sets off a trigger and other digital components down the signal path can begin their necessary processes.

Barker codes are commonly used as a message preamble to mark the start. Barker codes are unique for their high peak to side lobe ratio. Barker codes produce a large spike (equal to the length of the Barker code) surrounded by small ripples when autocorrelated. This presents the system with a predictable and unique response to search for. To further the uniqueness and decrease the probability of a data anomaly triggering the system, a Barker sequence followed by a negated version can provide an even more uncommon response. Barker codes of length 1, 2, 3, 4, 5, 7, 11, and 13 have been discovered. The functional peak of the autocorrelated signal is equal to the length of the Barker code

Figure 15 displays the response of a correlation function of a random sequence with a regular and negated length 13 Barker codes inserted in the middle. Note the spikes of 13 at index 57 and 70 with values of 1 of -1 in between. The autocorrelation function replicates an FIR filter with the sequence values flipped from left to right. Therefore, any system with the access to FIR filters can implement the autocorrelation function, and therefore the Barker code start of sequence process.

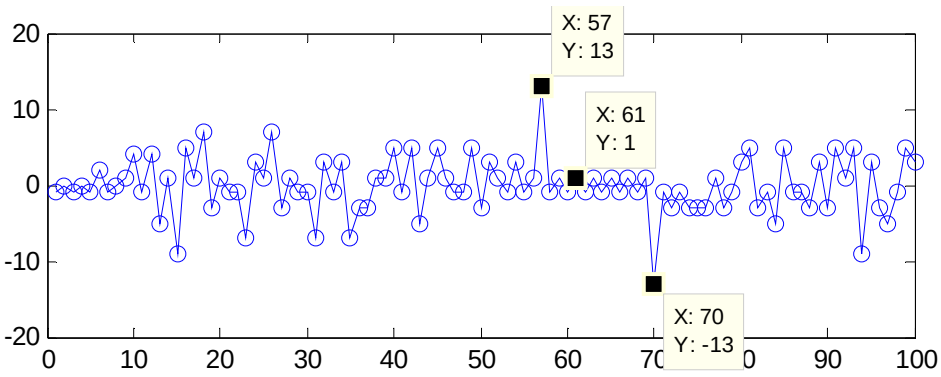


Figure 15 – Random signal + length 13 Barker code correlated with length 13 Barker code

Preliminary Simulation, Simulink, and Post-Processing

Matlab Simulink Components

The following models were designed, created, and simulated in Matlab Simulink. System Generator (in Simulink) then generates the models into VHDL files. These files were then connected and implemented in VHDL. The data chain follows the order of this section. The figure below shows a high level overview of the receiver system, sectioned off by Simulink model.

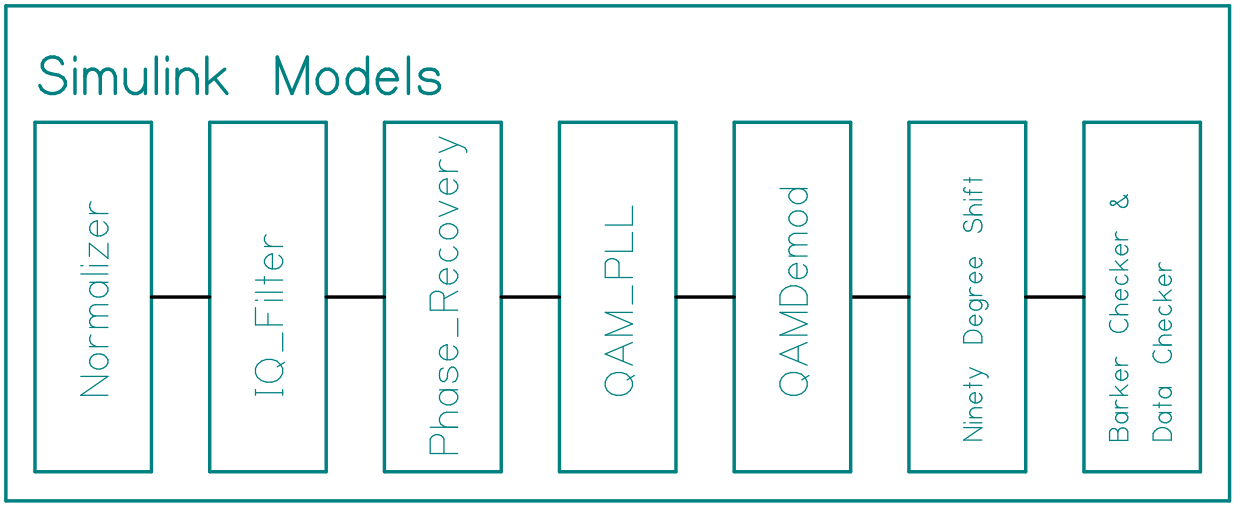


Figure 16 - Simulink Receiver System Configuration

The input data stream consists of 11 subchannels. The baseband channel is a pseudo-random binary data block and is consistent from trial to trial. The baseband channel is surrounded by 5 channels on each side which are randomly generated preceding each trial. Each data stream is individually modulated into 4QAM signals. From there, the subchannel signals are up or down-shifted to their respective center frequency, filtered by the Nyquist filter, and finally summed together to create the full channel. Subsequent to the Simulink model descriptions is further detail of the binary data generation and full channel transmit signal generation.

Normalizer

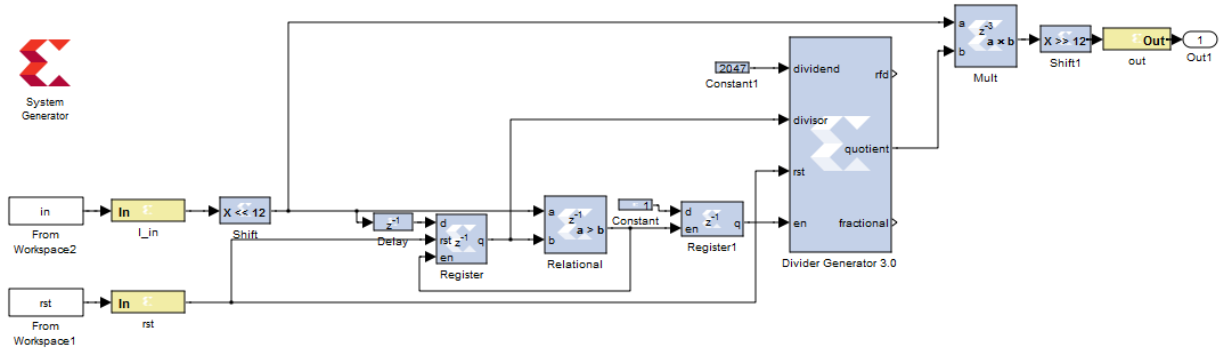


Figure 17 - Normalizer Simulink Model

The data stream must be normalized first before any digital signal processing. The normalization process ensures that all data blocks will be treated equally regardless of input magnitude. A signal well below the threshold voltages of the analog to digital converter will certainly be subject to a higher level of quantization error, and the normalization process cannot fix that problem. It is still necessary for the filtering process to always have relatively similar data magnitudes to ensure regularity. If the amplitude is too small, lesser fluctuations in the waveform can be lost in when rounding on the output.

This normalizer system first shifts the input signal leftward (higher) 12 bits. This is only necessary for the Simulink model to correctly place the decimal point in the divider block and does not actually take any FPGA resources. The delay, register, and relational blocks work to hold the maximum data amplitude in the register. Once the first maximum is found, Register1 enables the divider block. Being a signed 12 bit system, the maximum amplitude possible is 2047. The divider block creates a ratio of 2047 and the maximum detected point. This ratio is then multiplied to the data stream, such that the absolute value

product with the highest amplitude will be 2047. The binary number is then shifted back to a range of -.5:.5 and output.

IQ_filter

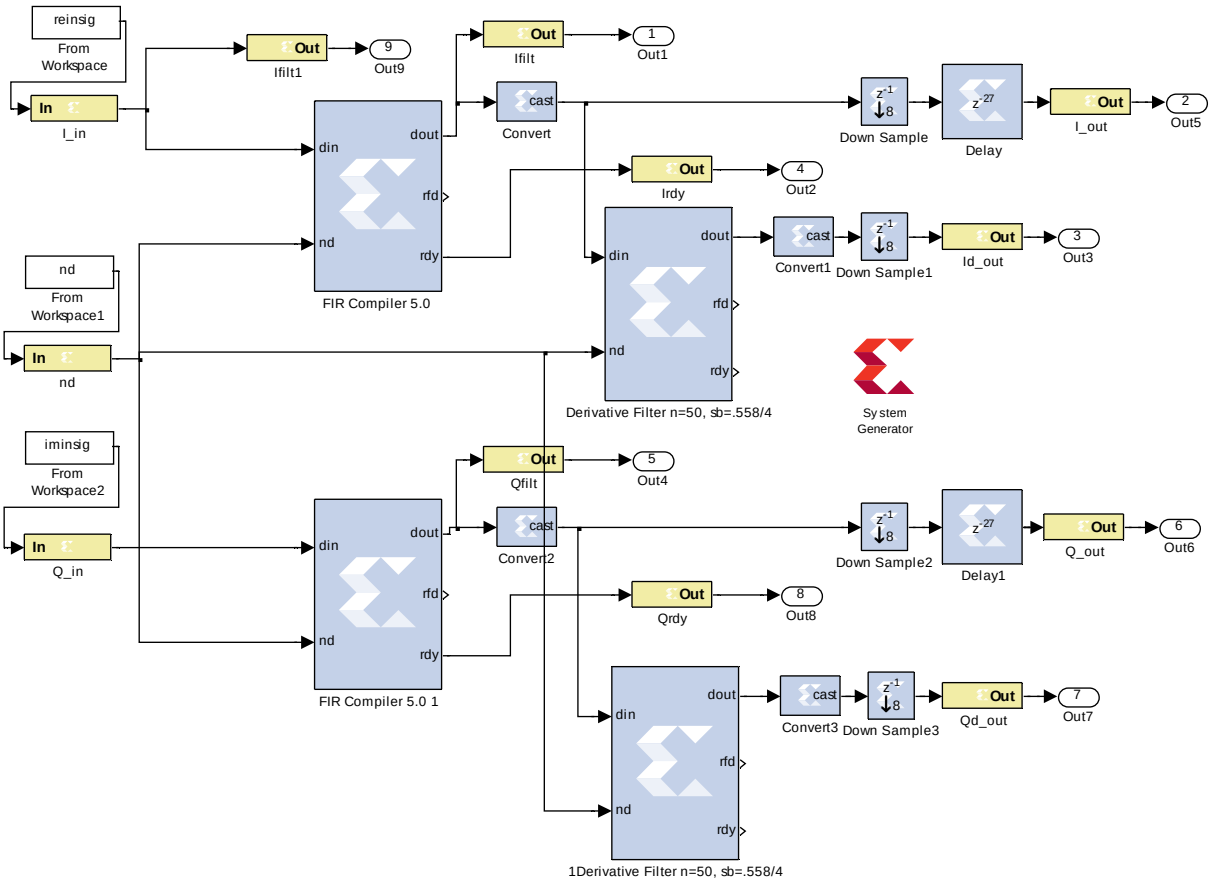


Figure 18 - IQ_Filter Simulink Model

The two channels of data require filtering to implement the Nyquist WDM demodulation scheme. The inputs to this model are reinsig (the I channel data stream), iminsig (the Q channel data stream), and nd (set always to '1' to enable the filters). A 100 order inverse sync, match filter shapes the response. The normalized passband and cutoff frequencies are defined by:

$$F_{ch} = \frac{F_{s,Tx} * ChanSp}{Pt/Sym} \qquad \text{Equation 40}$$

$$f_{p,mf} = \frac{F_{ch}}{F_{s,ADC} * N * ChanSp}$$ Equation 41

$$f_{c,mf} = \frac{F_{ch} * (2 - (1 - \beta))}{F_{s,ADC} * N}$$ Equation 42

where F_{ch} is the subchannel symbol bandwidth (~702 MHz for 32 point/symbol) at 100% channel spacing, *ChanSp* is the channel spacing ratio, $F_{s,Tx}$ is the Tx sampling frequency (~21.4 GHz), $F_{s,ADC}$ ADC sampling frequency (1.6 GHz), N is the interpolation factor, and β is the N-WDM roll-off factor. This filter block also performs a 4x interpolation to increase the samples per symbol from 2.39 to 9.5621 in the 32 point per symbol case. The filter amplifies the passband region by ~12 dB to offset the 4x interpolation (20*log10(4) ~= 12 dB). The cutoff frequency is placed at the peak Nyquist filter point of the closest adjacent sub-channel. The passband frequency is independent to the channel spacing, whereas the cutoff frequency is. This allows the filter more "room" to produce a flat passband response and higher level of rejection of undesired signals as the channel spacing is increased.

The following Table 1 defines some key attributes of different match filters with different channel spacing ratios. All the following assume a 32 points per symbol transmission rate, as above, which yields a single-sideband bandwidth of 334.7 MHz. The Fpass*4 and Fstop*4 columns are included to compare the true passband and stopband frequencies prior to interpolation. The Fpass and Fstop column reveal the normalized bandwidth, assuming a sampling frequency of 2 and the interpolation has been applied. Passband ripple is difficult to define because inverse sinc transfer function inherently introduces a difference in passband response, so a difference of adjacent ripples is used. The following estimate is the difference from DC to Fpass. The difference between the

Fpass and stopband peaks is the stopband attenuation. The selected filter design method, equiripple, produces a stopband with peaks of equal height; therefore selecting any peak for measure is adequate. As this is an interpolation filter, this stopband will include copies of the desired baseband signal as well as unwanted channels, thus a high level of relative rejection is required. The desirable linear phase response is realized in all the filters.

channel space	Fpass*4 (freq | normalized)	Fstop*4 (freq | normalized)	Fpass | Fstop (normalized)	Passband ripple (dB)	Stopband attenuation (dB)
1	334.7 MHz | .418	352 MHz | .440	.1046 | .1150	2.56	14.64
1.1	334.7 MHz | .418	387 MHz | .484	.1046 | .1265	0.85	25.36
1.2	334.7 MHz | .418	423 MHz | .528	.1046 | .1380	0.31	34.48
1.3	334.7 MHz | .418	458 MHz | .572	.1046 | .1495	0.10	43.88

Table 1 - Interpolated Match Filter Specifications vs Channel Spacing

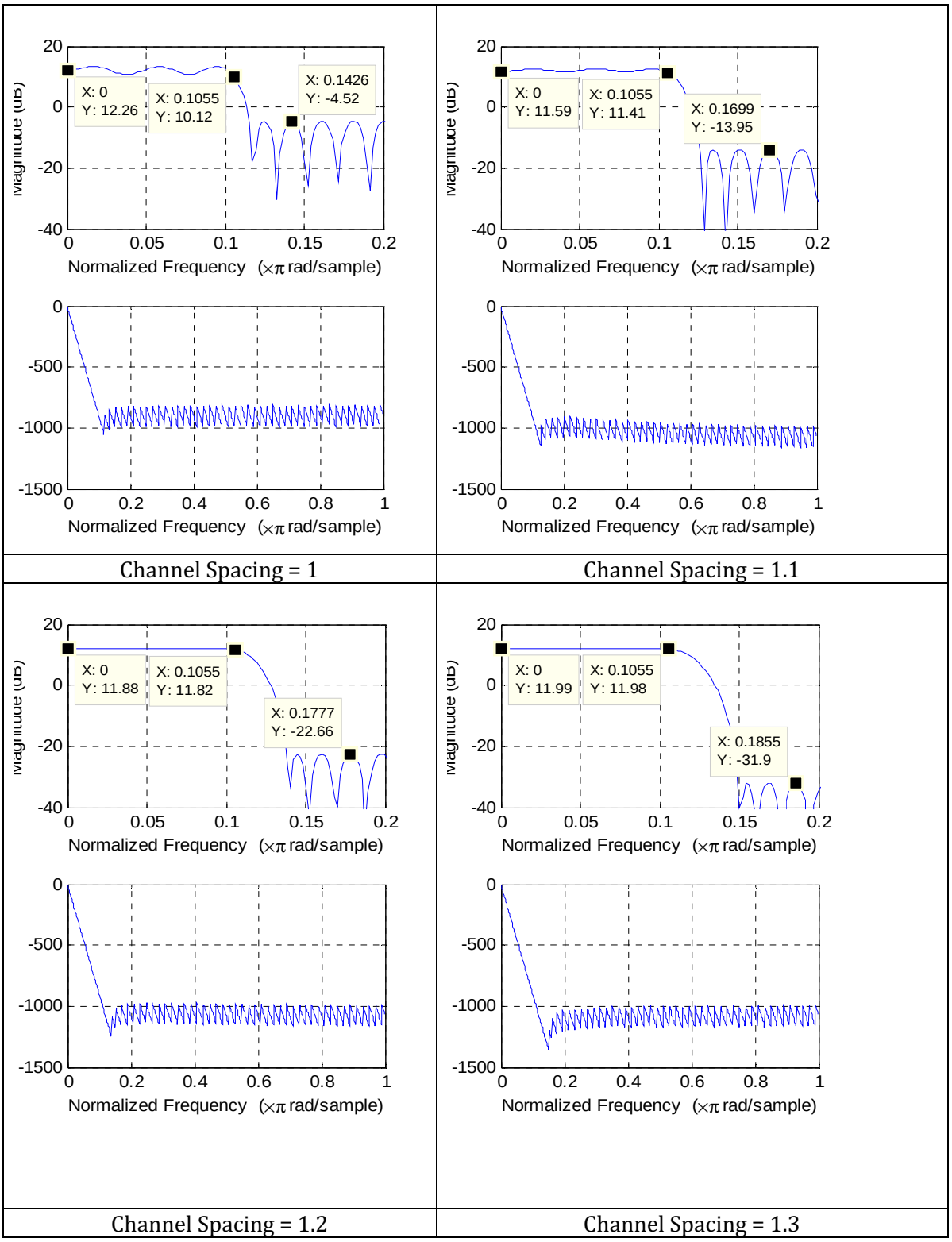


Figure 19 - Interpolated Match Filter Responses vs Channel Spacing

The method of clock recovery requires the derivative of the input signal to run a digital PLL. Therefore, after match filter processes the signal an FIR derivative filter calculates the change. A conversion block is placed between the two filters for ease of use and to reduce the resources necessary in the derivative filter by truncating the output. Since the derivative filter is in series with the matched filter, high attenuation in the signal stopband has already been achieved and mustn't be duplicated. Thus, the weight vector of the filter weighed heavily on the passband shape of the derivative and put less emphasis on the stopband.

Both the passband and cutoff frequencies of the derivative filter differ from the match filter. The passband does not divide out the channel spacing factor, allowing all the transmitted phase information to be encapsulated in output of the filter. The cutoff frequency is similar to the match filter's, only the $2/1.9$ factor on the beta term opens the main lobe further to, again, transfer the maximum phase information of the baseband signal.

$$f_{p,df} = \frac{F_{ch}}{F_{s,ADC} * N} \qquad \text{Equation 43}$$

$$f_{c,df} = \frac{F_{ch} * (2 - \frac{2}{1.9} * (1 - \beta))}{F_{s,ADC} * N} \qquad \text{Equation 44}$$

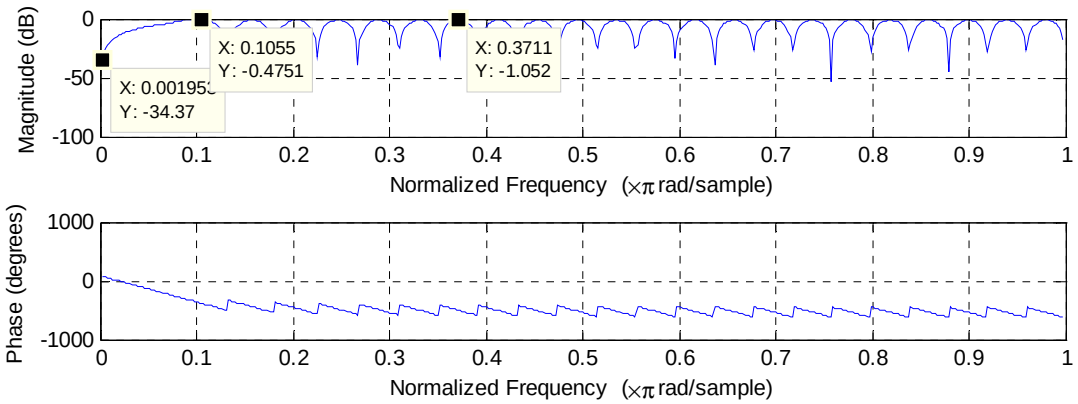


Figure 20 – Derivative Filter Frequency Response

Finally, all the signal paths are downsampled by 8 to conserve resources. The delay blocks hold the match filter output value for 27 clocks to synchronize the signal with the correct phase of the derivative. Without the downsampling blocks the delay would be 8 times longer, unnecessarily wasting flip-flops. The block parameters of the filters and delay must be adjusted depending on input signal bandwidth and filter order. This is done automatically by inserting the appropriate variable names from the script into the prompts.

Due to the limited resources necessary to perform the FIR filtering in the FPGA (namely the 48 available XtremeDSP blocks), the matched and derivative filters are overclocked by 32 times the FIFO read data rate. Since the data from the matched filter is 4x interpolated, though, the derivative filter is only overclocked 8 times its input data rate. Further discussion on data clock rates located below in the simulation details.

Phase_Recovery

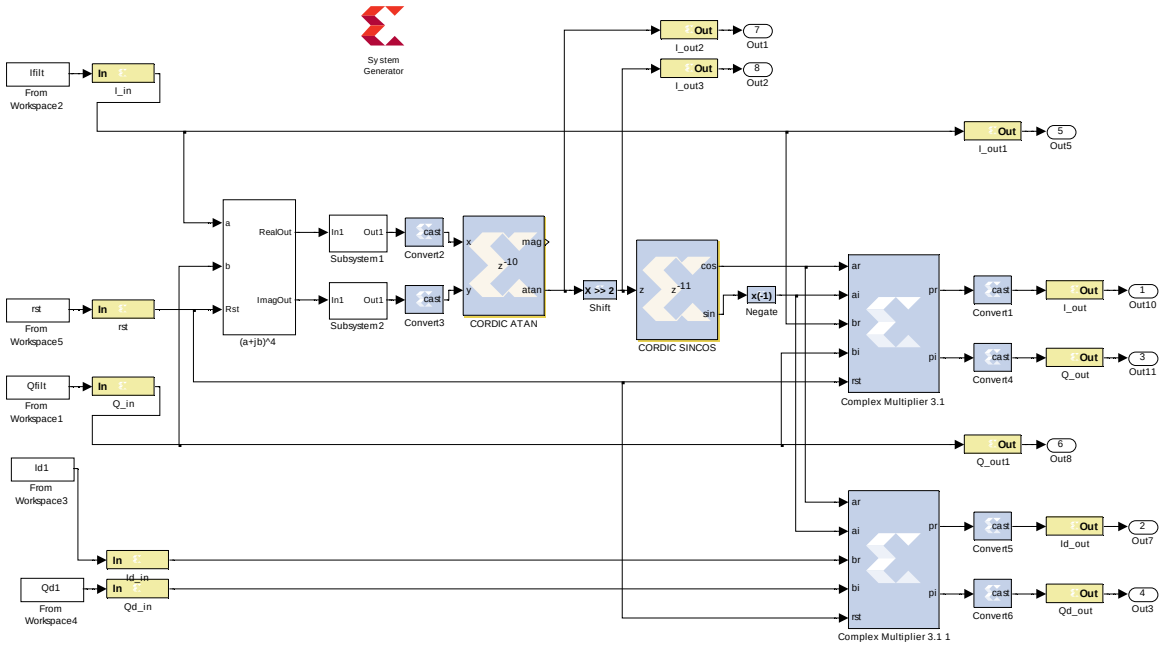


Figure 21 - Phase Recovery Simulink Model

To correct the phase and align the constellation to the transmitted signal, the Viterbi-Viterbi algorithm was implemented. The Simulink system is straight forward, with respect to the algorithm. First the complex signal is squared twice due to the 4 points of the constellation. The subsystem shown below completes this process, named "(a + jb)^4" in Figure 21 above. The multiplication is a complex process as needed.

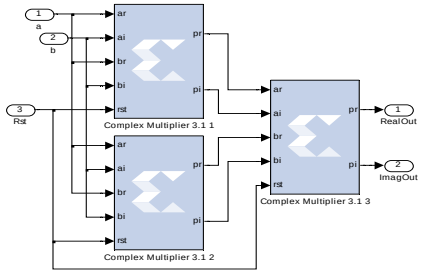


Figure 22 - "(a + jb)^4" Simulink Subsystem

The product is independently summed over 64 points by channel. Ideally, the current point being aligned is centered in the summation by using delays on the pass-through data lines to assure proper alignment. For instance, the bottom two derivative lines would be delayed in order to be temporally aligned with the phase product calculated by the central Viterbi-Viterbi system. The thin linewidth of the local oscillator and requirement to save FPGA resources allows this process to be ignored. Figure 23 shows and implements the column of z^{-16} delays, followed by a chain of adders to complete the 64 point summation. Each proceeding 16 point delay is latched to the previous delay, thus the delay accumulates down the chain. Each delay feeds into a block which sums the last 16 data points. The "SumDelay16" block internals are shown to the right. The 4 16 point sums are then summed in "Sum4."

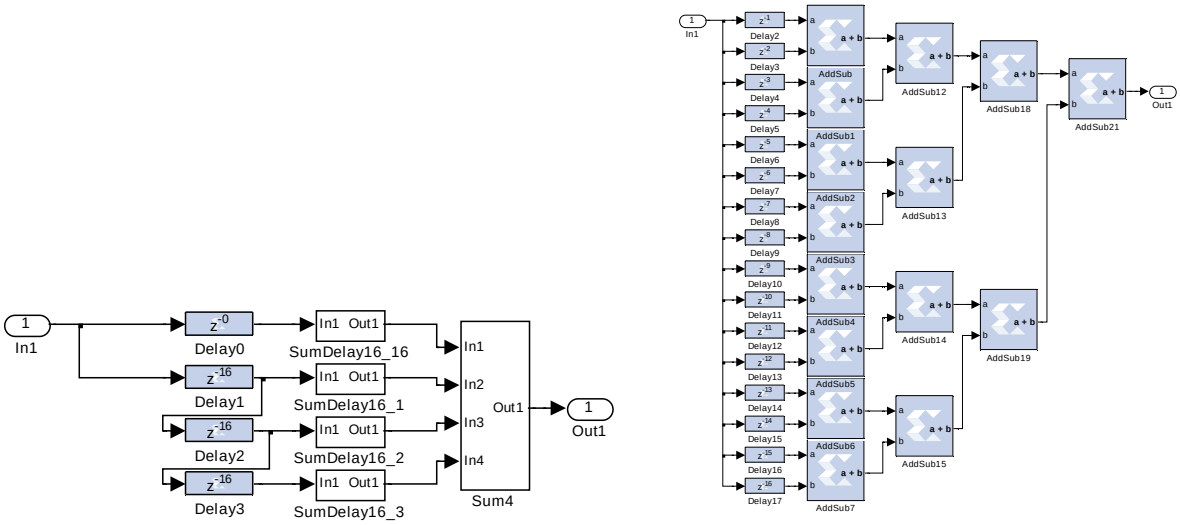


Figure 23 – Summing Simulink Subsystems

A "CORDIC ATAN" block calculates the magnitude and angle of the 64 points of data. The magnitude of the point is unneeded and left unconnected. The "shift" block performs a

divide by 4 function by shifting the binary data right two places. This is to compensate for the fourth power operation. The "CORDIC SINCOS" outputs the cosine and sine of the current phase detected by the previous process. The final step involves keeping the sign of the cosine term, negating the sine term, and complex multiplying this to the current data point. The output constellation should then be corrected and rotated to the input phase, with the desired point means at 0, 90, 180, and 270 degrees.

QAM_PLL

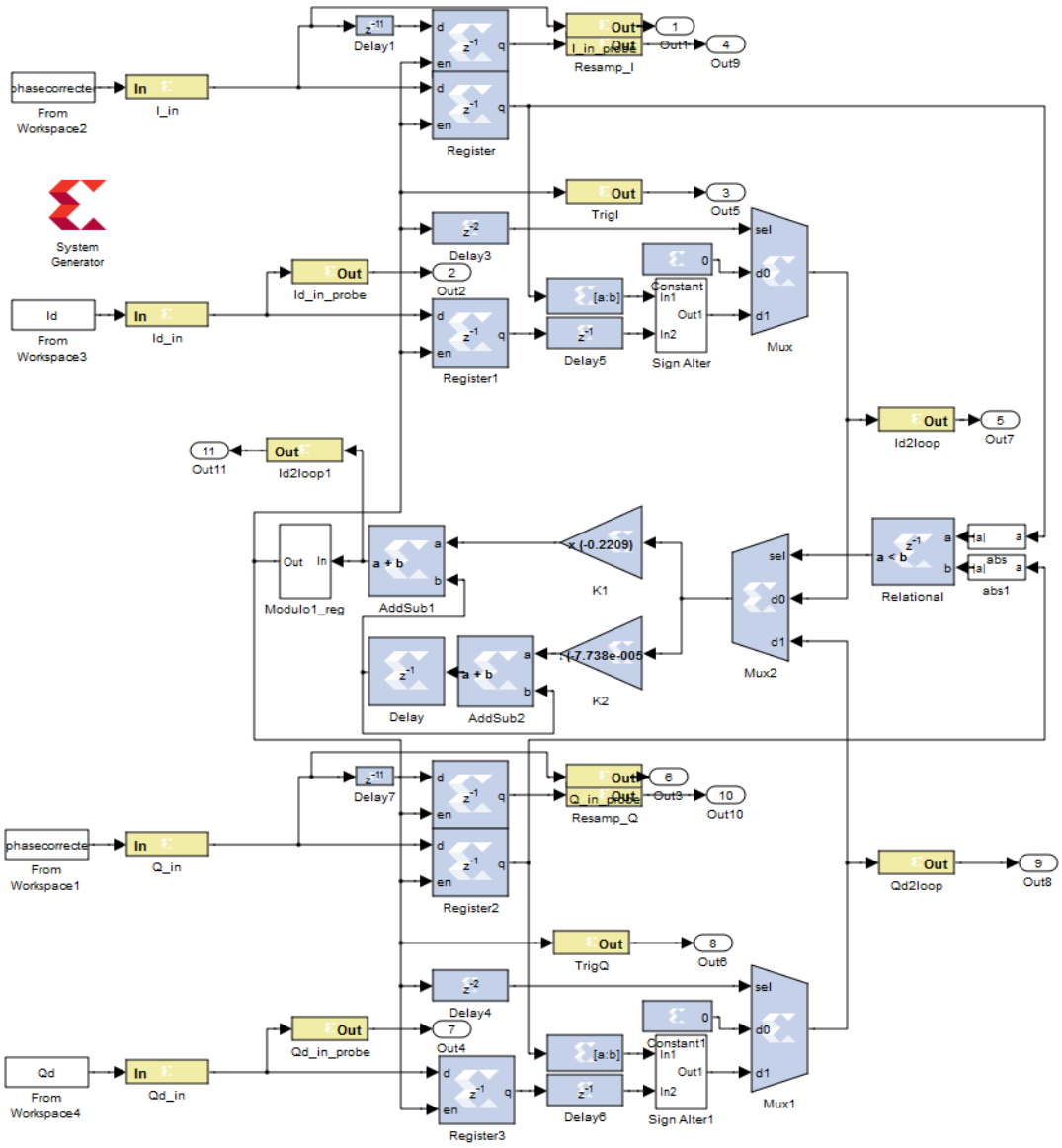


Figure 24 - QAM_PLL Simulink Model

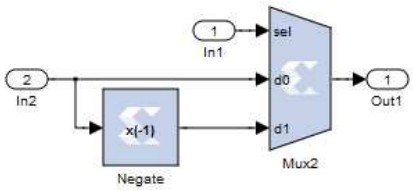


Figure 25 - Sign Alter Simulink Subsystem

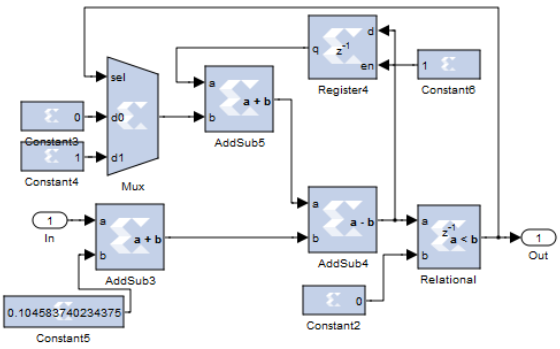


Figure 26 - Modulo -1 Simulink Subsystem

The most complex system is the digital QAM PLL. The system consists of 2 equivalent components (the upper and lower portions of Figure 24), which come together into the center, feedback portion. The top (or I) equivalent component will be referred to in this discussion. The process of this system was discussed in detail in the previous section. The primary role of this system is to reclock the waveform to switch from the DAC clock domain to the QAM signal's clock domain. The data is reclocked by the column of registers. The 'enabled' source of the registers is the output of the modulo -1 subsystem, displayed in Figure 26.

Beginning at the inputs, the phase-corrected waveform and phase-corrected derivatives of both the I and Q channels are stored in the registers. The registers themselves begin with a 1 on their enabled ports, so after a single clock, the data continues to the output q of the registers. The very top register is the reclocked output of the system. The second register feeds into the relational input of the feedback loop and will be discussed shortly. The third subsection registers the derivative. As discussed previously, the sign of the derivative must be altered before being fed into the feedback system depending on whether the current clock approximation is in front of or behind the QAM peak. The signal's first bit, which represents the sign, and the derivative feed into the 'Sign Alter' subsystem to complete this process. The sign altered derivative and a constant '0' are ten muxed together. The 'sel' bit is the reclocking signal, such that the approximated peak's derivative passes through the mux for one clock, followed by zeroes until the next reclock

The center feedback section beings with a mux. This mux selects between the I and Q sign-altered derivative signals. The 'sel' bit chooses the derivative whose anti-derivative has a larger magnitude. For instance, consider a moment in time with I = 0.8, Id = -.1, Q = 0.1, and Qd = -0.3. The derivative signal selected to drive the feedback loop in this instance would be from the I side. The reason this extra layer of complexity was added is due to the band-limited nature of this system. When a QAM signal is very band-limited, as in a

Nyquist WDM signal, the peaks tend to be flatter than the zeros of the I and Q waveforms. Therefore, the derivative of the peaks more accurately represent whether the reclocking estimate is truly ahead or behind.

The feedback loop begins with the two scaling multipliers for the 1st and 2nd order signals. The two are summed and the 'Modulo1_reg' preforms the reclocking process. First the scaled signal is added to 0.1046… which is the inverse of the ratio of clocks per symbol period of the transmitter clock and DAC clock. The 0.1046… constant creates a correctly spaced reclocking signal if the input stimulus consisted of only zeros. The summed signal subtracts subtracts from the feedback loop of the Modulo-1 counter. This creates the downward slope of the feedback loop. A relational block checks to see if the feedback difference is less than zero. If not, nothing happens. Once the difference is less than zero, a '1' is added into the loop to begin counting towards the next period, and a '1' is output to the outer system. This output signal is the reclocked signal which is the estimation of the input QAM clock.

QAMDemod

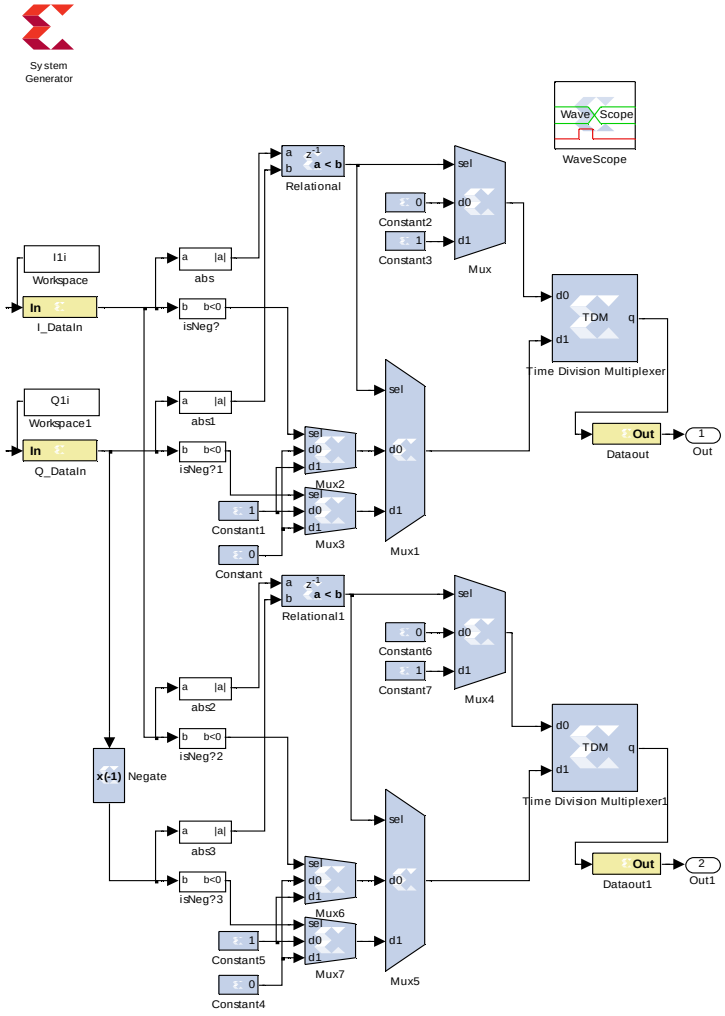


Figure 27 - QAM Demod Simulink Model

The QAM demodulation performed by this Simulink model requires no normalization to the input symbol. Figure 28 displays the demodulation decision areas; the solid lines define the I and Q data axes, the dashed line divides the QAM decision areas, and the dotted lines fill in the decision areas. The numbers at the end of each division line represent the angular location of the line. First, the absolute value of the I and Q points is taken and compared. Whichever has a larger magnitude chooses which axis on the

constellation the symbol exists. As seen in Figure 28, this decides the first binary digit of the symbol: I>Q = 0, Q>I = 1. The demodulator decides on the second digit based on the sign of the dominant point: I>Q, sgn(I) chooses second digit. Thus, since the decision engine relies only on the magnitude comparison of the two points and the sign of the larger, no the data requires no normalization for demodulation.

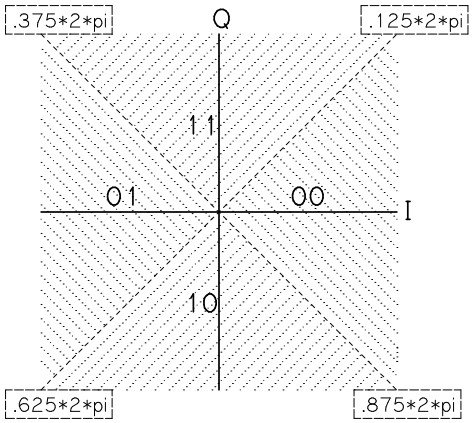


Figure 28 - Transmit QAM Constellation

The QAM Demod block simultaneously converts each QAM symbol into 2 different 2-bit words. One word is translated regularly from the I and Q (I/Q) symbol (the upper system in Figure 27). The second negates the Q value and translates the symbol from the I and –Q (I/-Q) symbol (the lower system in Figure 27). The two demodulators are otherwise equivalent. These two bit streams are needed for the following blocks to perform their phase correction duties.

Ninety Degree Shift

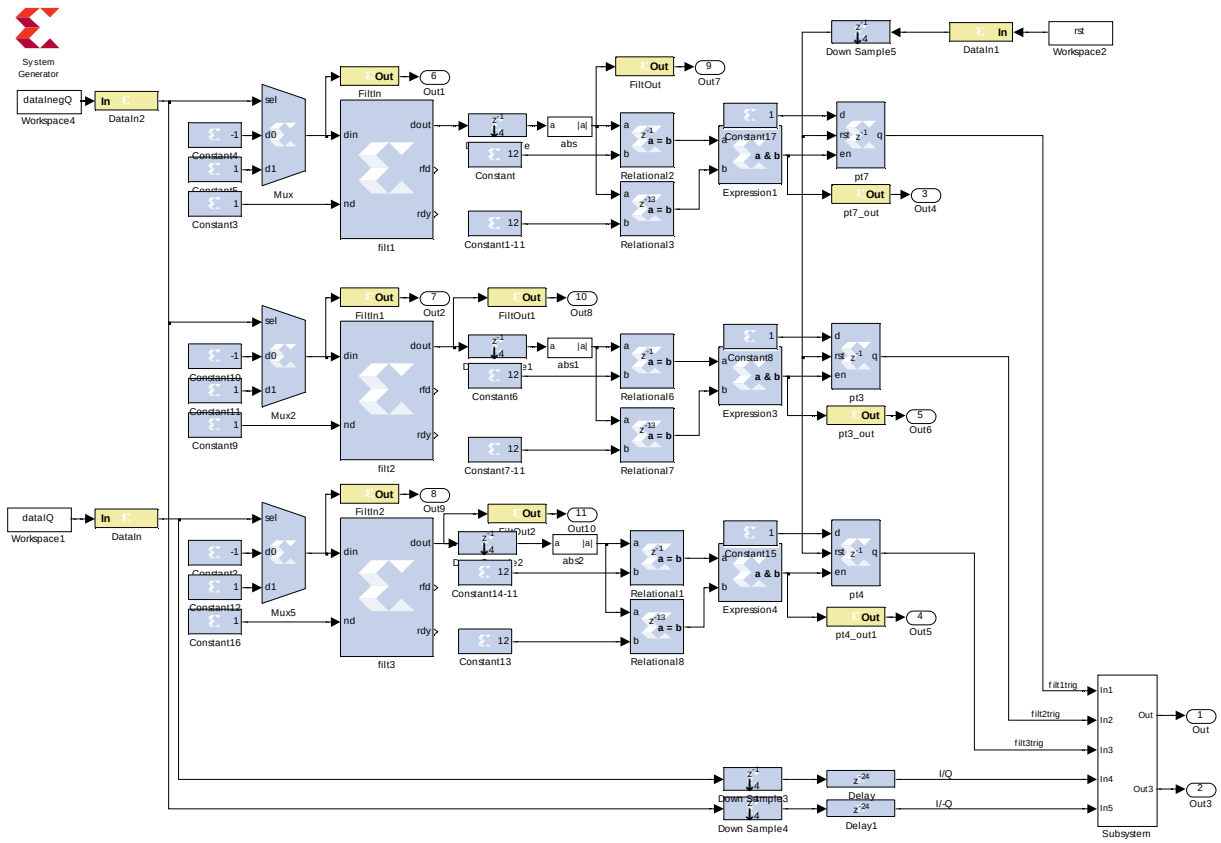


Figure 29 - Ninety Degree Shift Simulink Model

The 90 degree shift model exploits the 11 length Barker codes located at the beginning of the data sequence. Referring to Figure 28, the data phase can flip over the I axis, over the Q axis, over either of the decision axes, or any combination of the 4. Therefore, sometimes the phase of the I axis is correct with respect to the expected constellation, but incorrect on the Q axis. For this reason, the QAM Demod box outputs the I & Q data stream and the I & -Q data stream.

The length 11 Barker code has the following sequence:

-1	**1**	**-1**	**-1**	**1**	**-1**	**-1**	**-1**	**1**	**1**	**1**

Table 2 - Length 11 Barker Code

Note, to test the correlation the zeros are represented as negative ones to create a larger, more accurate spike in the correlation process. To check for 180 degree phase shifts on the Cartesian and constellation axes, 3 variations of the above code are checked with 1 more as default. The first, the default, does not flip any bits and feeds through the I/Q bitstream by leaving all the select bits of the MUXs on the bottom of the model set to 0. When this mode is enabled, the 00 transmitted phase is aligned with the 00 and so on.

The other 3 variations of the length 11 Barker code overhead will trigger one of the three similar paths. First, each path translates the zeros into negative ones with the input mux. The sequence of ones and negative ones are correlated with variations of the input Barker code using the FIR filter builder. By entering the following filter coefficients flipped left-to-right, the filter operation becomes a correlation operation.

Symbol#	***1***		***2***		***3***		***4***		***5***		
Input	***-1***	***1***	***-1***	***-1***	***1***	***-1***	***-1***	***-1***	***1***	***1***	***1***
Filt1	***-1***	***1***	***-1***	***-1***	***1***	***-1***	***-1***	***-1***	***1***	***1***	***1***
Filt2	***-1***	***-1***	***-1***	***1***	***1***	***1***	***-1***	***1***	***1***	***-1***	***1***
Filt3	***-1***	***-1***	***-1***	***1***	***1***	***1***	***-1***	***1***	***1***	***-1***	***1***

Table 3 - Barker code and Filter Coefficient Comparison

Note the partitioning by every 2 bits, because each QAM symbol represents 2 bits.

The absolute value of the output of each filter (correlator) is then taken. To cause a trigger to occur, the system must detect a magnitude 11 spike twice, exactly 12 samples apart. The separation between the spikes represents the length 11 Barker code, plus the extra bit added after each code in the data formation. The relational blocks check if the

data equals 11 at 1 delay (minimum) and 13 delay, for a separation of 12. The two outputs drive an AND logical block, whose output is a trigger lasting one clock length. To extend the trigger longer, the trigger sets the following D flip-flop. A reset signal is necessary to return the D flip-flop to 0. A universal reset follows the end of every data set.

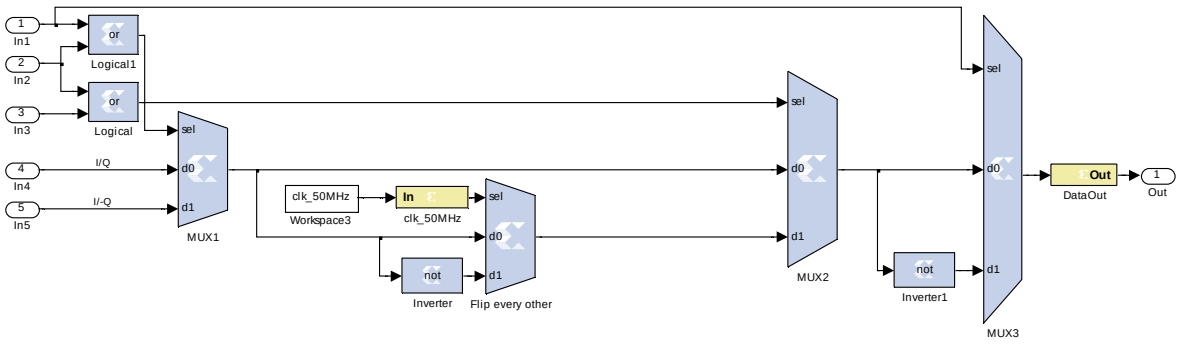


Figure 30- Constellation Correction Subsystem

The triggers control a system of MUXs choosing between different 90 and 180 degree rotated data lines, as seen in Figure 30. There are 4 MUXs in the subsystem. The first (left to right), named MUX1, chooses between it's the I/Q and I/-Q demodulated bitstreams. This flips the constellation over the I axis. The second MUX runs continuously on a 50 MHz clock, flipping every bit, regardless of the triggers. A 50 MHz clock is used as that is the data rate. Flipping every other bit flips the constellation over the I and Q axes at the same time: 11->10, 10->11, 00->01, and 01->00. The next MUX (MUX2) chooses between the unaltered data line or the one flipped by the 50 MHz clock. The final MUX (MUX3)chooses between another unaltered data line or one flipping every bit. This can be seen as flipping the constellation over Axis 3 in Figure 31. The following table breaks down the choices and how the data is altered.

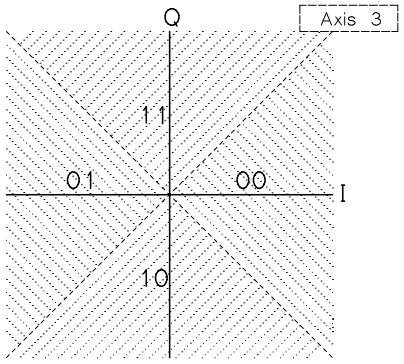


Figure 31 - QAM Constellation Correction

MUX	SEL 0	SEL 1	Alteration
MUX1	I/Q	I/-Q	Flip over I axis
MUX2	MUX1 Out	Not every other	Flip over I&Q axes
MUX3	MUX2 Out	Not all bits	Flip over Axis 3

Table 4 - Phase Shift MUX's Alterations

Event	MUX1 Select	MUX2 Select	MUX3 Select	Rotation
No Trigger	0	0	0	0°
Filt1 Trigger	1	0	1	90°
Filt2 Trigger	1	1	1	-90°
Filt3 Trigger	0	1	0	180°

Table 5 - Triggers vs MUX Selections

This whole system is 4 times overclocked with respect to the data rate to reduce the resources needed for the FIR components. After the data has been processed by the filters, the data lines are downsampled by 4 to their native clock speed. The blocks after the filters contain registers which would actually increase the resources used if the data was overclocked. The I/Q and I/-Q bitstreams are also downsampled and delayed before

entering the phase correction subsystem. The delays are necessary to correct the all the data processed by the system and not merely the data after a trigger event occurs.

Barker Checker & Data Checker

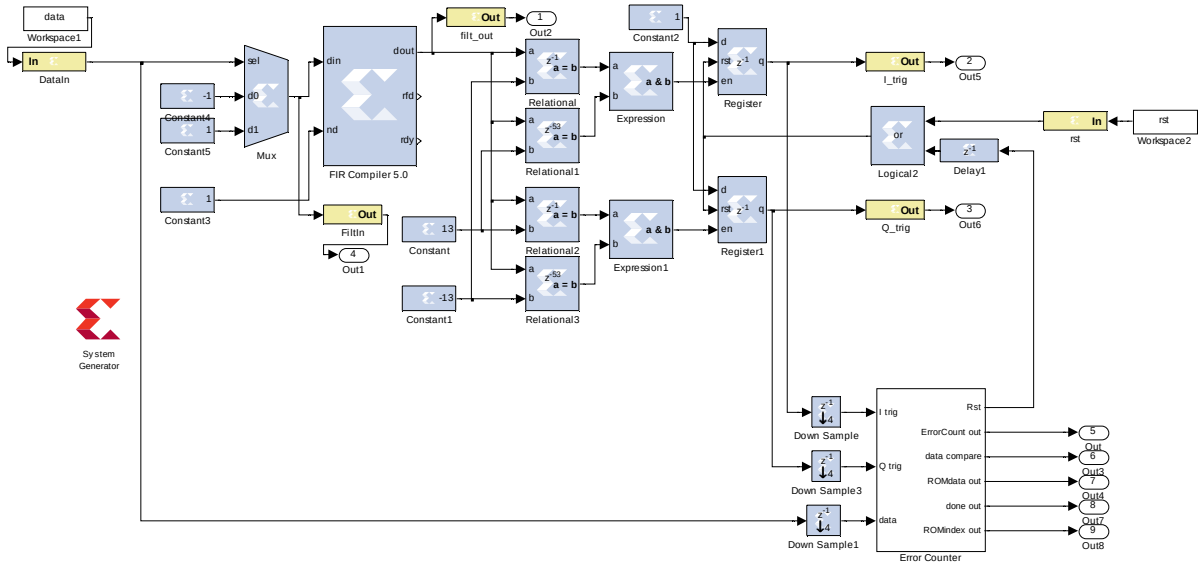


Figure 32 - Barker Checker and Data Checker Simulink Model

The bit error counter system completes the Simulink receiver. To begin counting and comparing the stored binary data to the received data it must be triggered. The system towards the top will trigger the BER comparator circuit much like the previous component, except the sequence the datastream is correlated with is the length 13 Barker code:

1	**-1**	**1**	**-1**	**1**	**1**	**-1**	**-1**	**1**	**1**	**1**	**1**	**1**

Table 6 - Length 13 Barker Code

The system has two triggers, one for a positive then immediate negative spike, one for the opposite. The data can still be 90° out of phase depending on the phase relation of the 50 MHz clock and the bitstream. A negative spike first will flip every 0 to 1 and every 1 to 0. The overclocked triggers and received binary data are downsampled and fed into the Error Counter Subsystem.

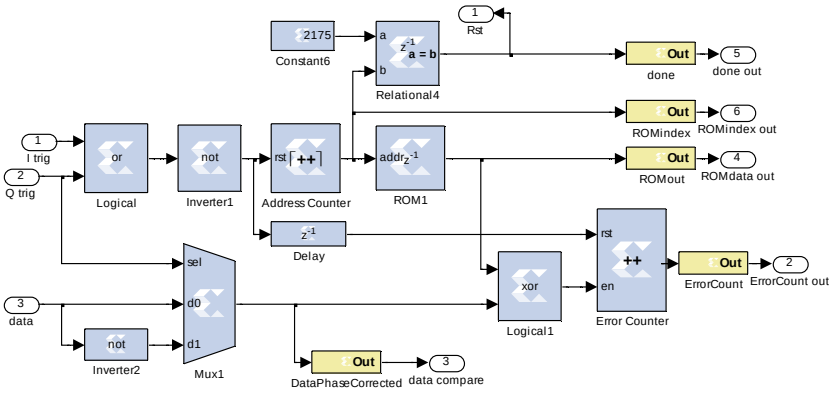


Figure 33 - Error Counter Subsystem

The error counter inputs are the I trigger, Q trigger, and the binary data. Once the system receives either trigger the reset input of the address counter flips to 0, and the address counter begins to count. The ROM has stored the transmitted binary waveform and as the address counter increments, the bits are output. One clock cycle later, the reset of the error counter flips to 0 as well. The ROM binary waveform and the again phase-corrected received binary data drive an XOR bit by bit. If a Rom bit and received bit differ, the enable of the error counter receives a 1 for one clock cycle, incrementing the counter. Once the address counter reaches the total number of transmitted bits per data block (2176), a done signal is activated. An external VHDL component then reads the total errors at this point. A clock cycle later, an internal reset signal disables the triggers, which in turn enables the resets on the counters.

Simulation Execution

The above Simulink components are so useful because they can be used in simulation and compiled for hardware implementation. The issues of sampling, synchronization, and data rates can be resolved in Matlab instead of on the FPGA in VHDL. The filters are verified in Matlab with full access to the data streams, as well as the phase and clock recovery processes, QAM demodulation, and error checking. The following section explains the simulation process in Matlab. The script *ReceiverSimulink.m,* located in Appendix C, is the top level program of the Simulink simulation system. The responsibilities of the script include, but are not limited to: creating input data, combining task-specific functions, executing the Simulink modules, completing retiming steps, analyzing the module outputs, and displaying the results in plots. The line numbers referenced are labeled in the Appendix. Any referenced custom functions are also explained in detail below.

ReceiverSimulink.m

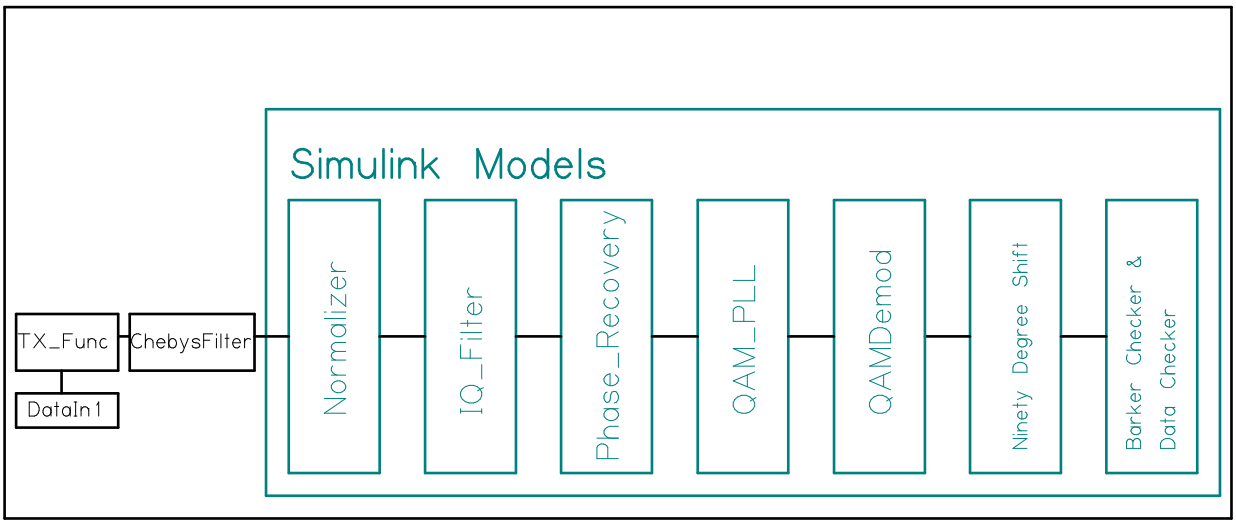


Figure 34 - Integration of Simulink Models into a Matlab Simulation

To begin, the script clears all variables from the current workspace. Then, *filenames.m* loads data and module filenames into the workspace to make calling these files easier and less cumbersome. The function establishes the Tx sampling frequency "Fs" and bits per symbol constants, 21.4136 GHz and 32, respectively. The scale factor on the transmitter frequency was added by sampling a sinusoid being sent and adjusting the predicted transmit speed until it exactly matched. The function *TX_func.m* (details given below) creates the Nyquist-WDM data stream to be loaded into the physical and simulated transmitter.

To resample the simulation data from the transmitted sampling rate to the received, two constants "*upsamprate*" and "*downsamprate*" (U & D in the following equation):

$$\frac{F_{s,Rx}}{F_{s,Tx}} = \frac{U}{D} \qquad \text{Equation 45}$$

With the given 1.6 GHz received and 21.4186 GHz transmitted sampling rates, the upsampling and downsampling rates are 344 and 4605, a ratio of 74.7014e-003. Due to memory constraints brought on by upsampling the signal 4605 times, U and D were reduced to 7 and 94, resulting in a ratio of 74.4681e-003, deemed sufficient in this simulation. The upsampling occurs before the anti-alias filter to replicate an analog system and then the downsampling mimics the sampling process by the ADC.

The file "filter_550" loads the response of the anti-aliasing filter (BLP-550+), as measured by a spectrum analyzer. The 800 MHz indicator in Figure 34 marks the ADC Nyquist frequency. The filter attenuates the input signal by 34 dB or higher in the rejection band. The desired signal bandwidth (500 MHz) fits well into the 2 dB passband (550 MHz).

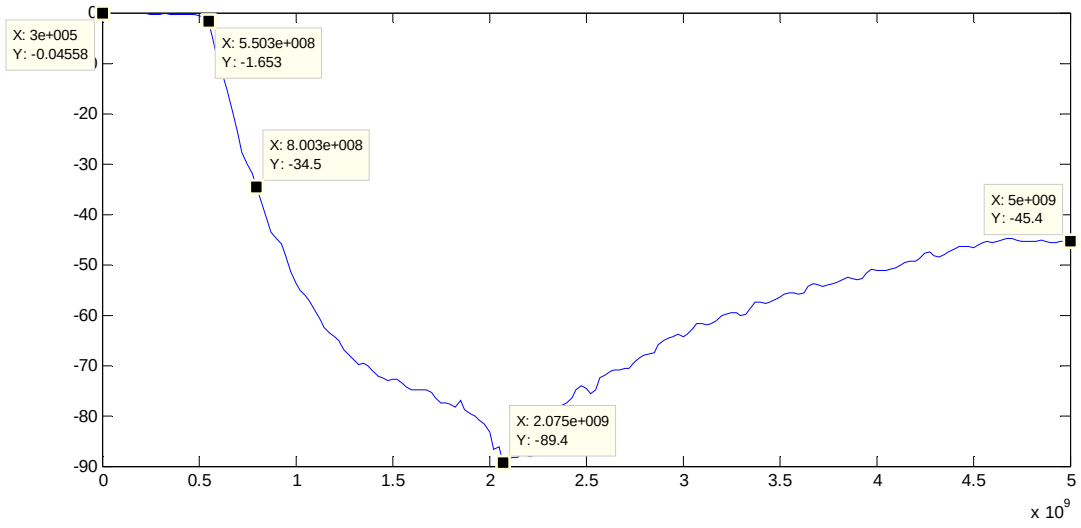


Figure 35 – Measured Analog Anti-Aliasing Filter Response

A custom function *chebysfilter.m* (explained below) filters the analog signal with the anti-aliasing filter.

The analog portion of the script is complete after the output of the anti-aliasing filter is copied to replicate the transmitter repeating the signal. The *circshift* function adds an integer phase shift to the input signal to mimic the random time-shift the sampling would begin. The signals, now "digitized", must be demodulated with the Simulink models. A final scaling factor of `(20*max(abs(insig)))` divides the input signal down to replicate the experimental average signal peak-to-peak voltage to ADC input peak-to-peak voltage ratio, 20%-35%.

The inputs of the Simulink models require a time column and a data column. The IQ Filter model clock is 32 times faster than the data clock to reduce FPGA resources. Therefore, in the simulation the time array counts in intervals of. The "tt" variable in the script shows the ratio of the input data rate and the clock rate of the Simulink next model to run. For example:

```
tt=1:4:4*length(yout)-1;
```

Equation 46

demonstrates a block whose clock rate is 4 times higher than the input data rate. This process will inherently function this way in the FPGA with different blocks running on different clocks.

After the match filter model executes, the outputs are downsampled by a factor of 8 to replicate the difference in clock frequencies of the match filter block and the following phase recovery block: the match filter clock is 8 times faster than the phase recovery block. The output data of the filter block is 8 times oversampled in Matlab, as well, so the downsampling make the simulation easier.

The phase recovery box implements the Viterbi-Viterbi algorithm to shift the input constellation from an arbitrary phase to 0, 90, 180, or 270 degrees from transmitted phase. The output of the phase correction block needs no timing change to be processed by the QAM PLL block. The QAM PLL block, when locked on correctly, outputs the peak (desired) QAM points, throwing out the extra interpolated points between the QAM constellation values.

The following section, labeled "calculate PLL clocking shift vs time" in the Matlab script, measures and plots out the successive sampling shifts by the PLL. If the PLL locks onto the input data correctly, then the plot should have an overall linear trace per data block. The input data for the entire system, as previously mentioned, is the repetition of a single data block. Although measures were taken to reduce the phase shift from block to block, a gap realistically still exists. Therefore, within each data block the trace stays flat if implemented correctly or has a non-zero slope if not locked on correctly.

After plotting the input and output constellations, the binary demodulation process begins. The time vector for the input I and Q vectors of the QAM Demod block increment by 2 to account for each input QAM symbol representing 2 binary bits. The QAM PLL block resampled the I and Q signals to only their peak values. The phase of the constellation now matches an input QAM constellation, but could still suffer from 90, 180, or 270 degree phase shifts. The Viterbi-Viterbi algorithm cannot account for 90 degree shifts. These 90 degree rotations are corrected with the "90 degree shift" phase correction block at the beginning of data stream block. The QAM Demod component creates two output binary waveforms from the complex QAM signals for the phase shifting block to exploit. One

completes a straight forward demodulation. The second reverses the sign of the quadrature signal. This provides the following block with enough information to adjust the constellation accordingly.

The two outputs of the QAM Demod module are 4 times oversampled by adjusting their time vectors and then loaded into the 90 degree phase shift detector. This block correlates the input signal with the expected 11 bit Barker codes had they gone through any permutation of 90 degree shifts.

Lastly, the script displays if and where the data triggered a phase shift event in the 90 degree shift model, displays if and where the data triggered the data checking model, and how many errors occurred in each instance.

Data and Clock Timing

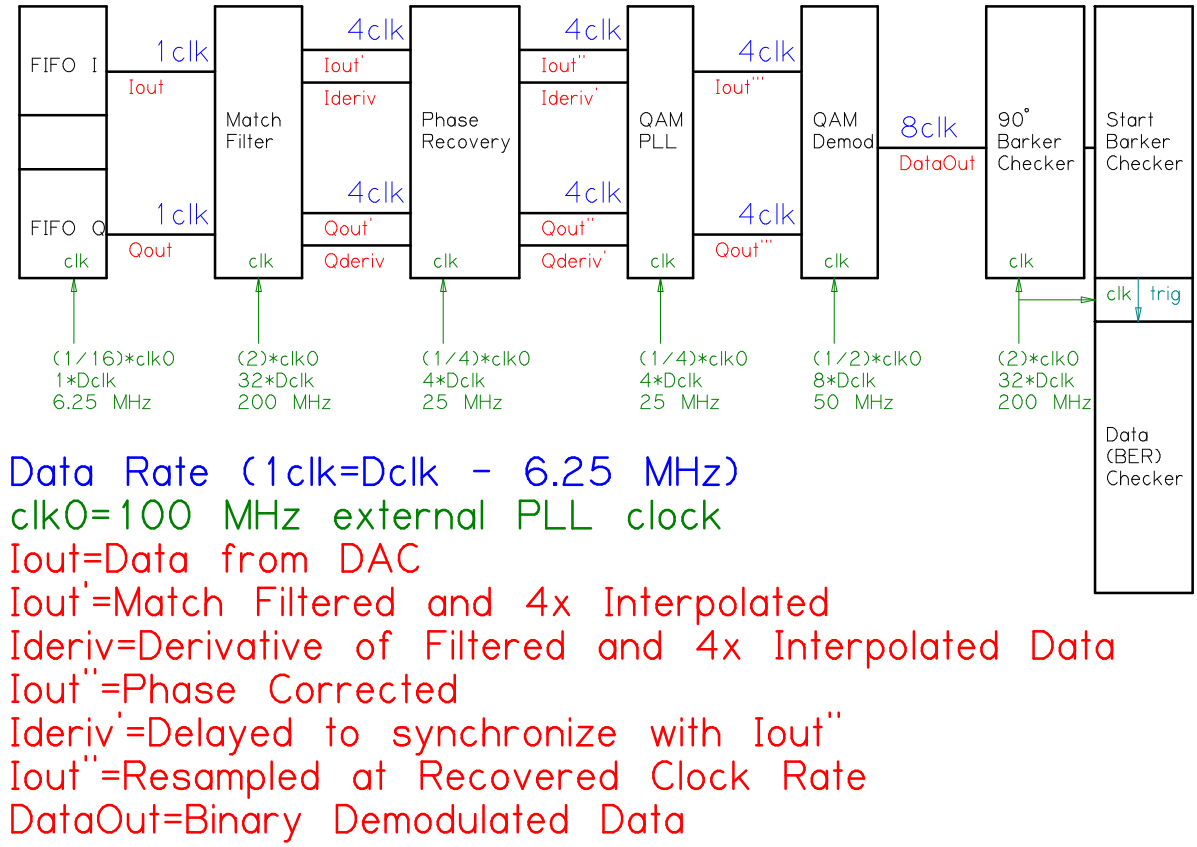


Figure 36 - Data and Clock Timing

The complexity of the data and clock timing of both the simulation and real systems requires a focused look at the data chain. Figure 36 maps out the data rate after each step in the receiver as well as the clock input required for each component in the FPGA. The data rates affect the programming of the simulation more directly, whereas the clock rates affect the FPGA programming more directly. Of course, both are virtually occurring in the simulation and physically occurring in the FPGA, but the simulation only depends on data rates of the input and the components in the FPGA only react to the input clock speed.

Therefore, the above figure includes all the timing parameters together for organization's sake.

As noted in the figure, the blue text denotes the data rate between each component with respect to the FIFO read clock (6.25MHz). Obviously, the first data line (Iout) is 1 times the data clock. The match filter interpolates the data by 4 times, therefore the data rate increases to 4 times the data clock (25 MHz). The phase recovery and QAM PLL do not alter the data rate. The QAM Demod bock, by the nature of 4 QAM demodulation, doubles the input data rate by two (50 MHz). The input of the demod block is the I and Q channels representing a 2 bit symbol, thus the output is twice as fast as the input. The 180 degree phase shifter, begin-sequence checker, and bit error rate detector do not alter the data rate.

The green text lists the clock speed for each block in three different ways; the three numbers for each are equivalent. The first is a ratio of the 100 MHz reference clock of the FPGA, the second is a ratio of the 6.25 MHz FIFO read clock, and the final is the physical clock speed. The match filter utilizes a 200 MHz clock, created from the clock doubler of the "ADV_DCM." As previously mentioned, the 100 order match filter and 50 order derivative filter would require more DSP48 blocks in the FPGA than are available. The Simulink block for FIR filters cannot adjust to any other method. For that reason, the filter block functions at a rate of 32 times the input data rate. The phase recovery and QAM PLL blocks do not use the critically limited DSP48 blocks, thus overclocking is unnecessary. These blocks would actually use more resources due to the delays in the data path: the blocks would require 4 times as many delays if overclocked by 4 times, for instance. The QAM Demod component time division multiplexes the 4QAM symbol data into a single

binary data stream, thus the clock rate doubles with respect to the previous block. The two following blocks use the same FIR filter compiler as the match filter block to check the Barker codes, so the maximum overclocking of 200 MHz or 4 times the input data rate is used.

The clock rates of each Simulink model are set by opening the Xilinx "System Generator" prompt and selecting the "Clocking" tab, as seen below. In this example, the *PhaseRecovery* model is set to run on a 40 ns (or 25 MHz) clock. Depending on implementation, this can be seen as bookkeeping more than anything. With "Expose Clock Ports" selected, the correct clock signal must be assigned to the input clock pin of the VHDL component. In models with multiple clock domains, for example those with downsampling blocks, there will appear several clock inputs in the VHDL component which must be tied to the correct clock. The system generator would compile a DCM into the component when "Expose Clock Ports" is not selected and multiple clock domains exist. This DCM would be used to divide or double the frequency of the input clock to match the requirements. Since FPGAs have a limited amount of DCMs, the separate clock domains will be created by an upper level VHDL component and input to the generated Simulink model.

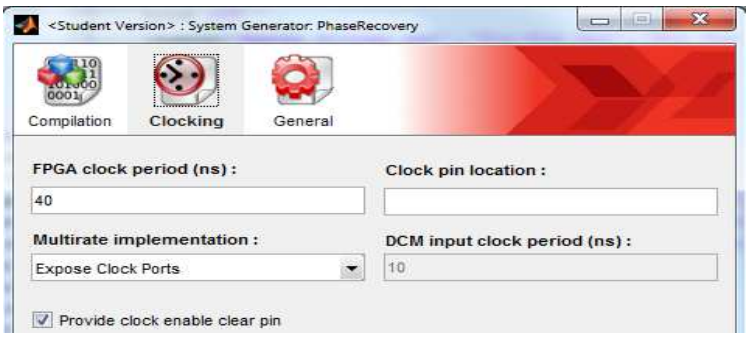


Figure 37 - Simulink Clock Configuration

TX_Func.m

`[tol,spe1,t,h,DataIn1]=TX_func(chans,chanspace,bitpersymbol,Fs)`

- Outputs
    - tol –complex output data stream, containing "chans" channels – (time domain)
    - spe1 –complex baseband signal, channel 1 of tol – (time domain)
    - t –time index for tol and spe1
    - h –Nyquist filter – (frequency domain)
    - DataIn1 - known pseudo-random bit stream modulating channel 1 (digital domain)
- Inputs
    - chans – array to choose which channels to include in tol (eg. [1 2 6 7])
    - chanspace – channel spacing variable (1 is pure Nyquist modulation)
    - bitpersymbol – oversampling rate of each symbol in the transmitter
    - Fs – sampling rate of the transmitter

The binary data of channel 1 (baseband) of the transmitted signal is pseudo-random, but known, and the same every time the simulation is run. Since the system checks against this channel to calculate the bit error rate, the original bitstream must be known. Thus, the function passes the data to the script it's called from. The pattern and required Barker codes are formed by *DataIn1_maker.m*, explained below.

All the other channels are pseudo-random and regenerated every simulation iteration. The first 2 and final 2 bits of every data channel (including channel 1) are duplicates to provide a reduced voltage jump when the cycle repeats. Two bits, not 1, are required since 4-QAM splits the 2 bits, effectively, into 1 bit in the in-phase channel and 1 bit in the quadrature channel. Therefore, prior to the Nyquist filtering, the I and Q level of the first symbol and final symbol equal one another. Following the Nyquist filtering, a difference in the exact level is inevitable, but the difference reduces with this preemptive measure. The duplication process is executed on line 31 for all channels, except channel 1 which is done during pre-processing.

The loop from line 27 to 43 first creates the data bit stream for each channel (except 1), then modulates an I and Q signal to -1, 0 or 1. Summing two consecutive bits, then applying a cosine or negative sine function calculates the current I and Q values. The function repeats each I and Q value by a factor of "bitpersymbol" using the *repmat* and *reshape* functions.

Next, the function calculates the Nyquist filter based on the sampling frequency, the channel spacing, and beta values. The Nyquist filter emerges from a boxcar filter multiplied by an inverse sinc function. Each channel's frequency response is multiplied to the filter, then to a complex sinusoid to shift each channel to its respective frequency slot. The *sum* function produces the "tol" variable, as the separate channels are organized into rows of the same matrix.

The following plot displays the tol signal in blue. The Nyquist filter of channel 1 laid over the unfiltered signal shows the repetitions of the digital spectrum reduced to only the

baseband signal. The channel 2 filter overlaps the channel 1 for high spectral efficiency. The 3 dB amplification of the edge of the subchannels and steep rolloff are apparent in this plot.

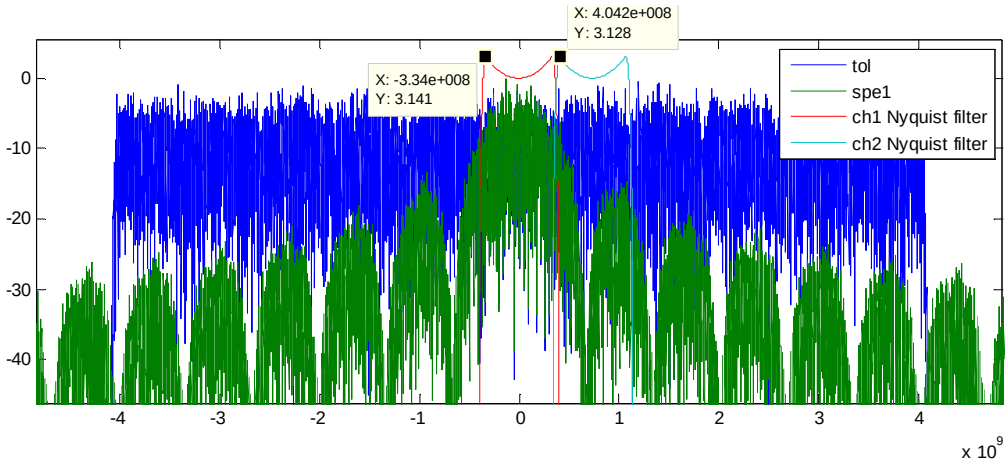


Figure 38 – Baseband Subchannel Input, Nyquist filter, and Transmitter Output Spectra

DataIn1_maker.m

In lieu of inputs and outputs, this Matlab script creates the desired pseudo-random bit pattern as specified by the variables located at the top o be saved and reused. The *bitpersymbol* variable is used to calculate the total number of binary bits that will be contained by the data stream. The transmitter memory limits the length of the data block to 2^15 points. The transmitter also requires the data block be an integer factor of 64, which the calculation of *TotalBit* takes into account.

The first 20 bits of the data block contain two crucial sections of data. The first 3 symbols match up with the final 4 symbols of the data block to reduce a phase jump when the memory repeats itself at the end of every cycle. The 8 proceeding symbols alternate the I channel from 1 to -1 to help the PLL lock onto the clock. Essentially, a low frequency

component reduces a frequency spike, then a high frequency component aids the clock recovery process.

Starting at bit 21 and extending to bit 44 is the 11-bit Barker code, followed by its negative, used for 180 degree phase shift detection. Immediately thereafter at bit 45 and extending to bit 70 is the 13-bit Barker code, followed by its negative. This block is used for beginning of data stream detection.

chebysfilter.m

```
[out,OUT]=chebysfilter(IN,fin,data)
```

- Outputs
  - out – filtered input signal – (time domain)
  - OUT – filtered input signal – (frequency domain)
- Inputs
  - IN – input signal – (frequency domain)
  - fin – frequency index of IN
  - filter – two column matrix
    - column 1 – frequency index of filter
    - column 2 – response of filter (dB) – (frequency domain)

This process needed its own function due to the different frequency indices of the input signal and the filter. The function interpolates both the input data and filter accurately match the responses, multiplies the two in the frequency domain, and outputs the result. As seen in Figure 35, the highest frequency component measured of the filter was 5 GHz. To extend the filter to higher frequencies, the final value of the filter was simply extended out. The 800 MHz point on the signal marks the edge of the ADC passband. The plot below demonstrates the analog filter (response in red) acting upon the total Nyquist-WDM spectrum in blue to produce the output in green. All three components, being in the "analog" domain, regularly extend much higher in frequency. This graph has been zoomed in for sake of legibility.

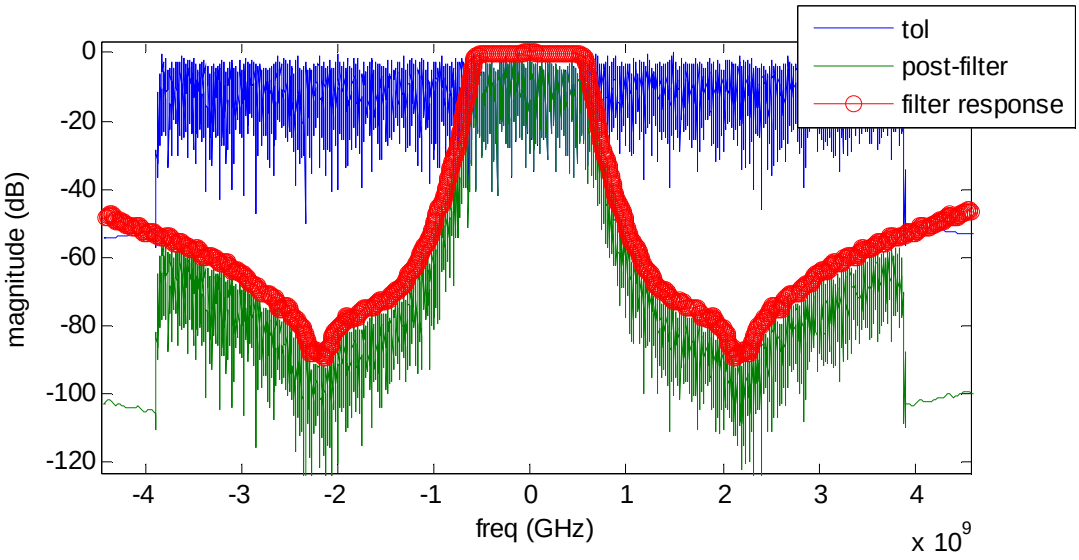


Figure 39 - Anti-Aliasing Filter Demonstration

Simulink Simulation Results

Once the transmit waveform is created, it is oversampled, analog filtered, downsampled, digitally filtered and interpolated. The following figures show a comparison of the spectrums through these processes. These spectra were modulated using 110% subchannel spacing.

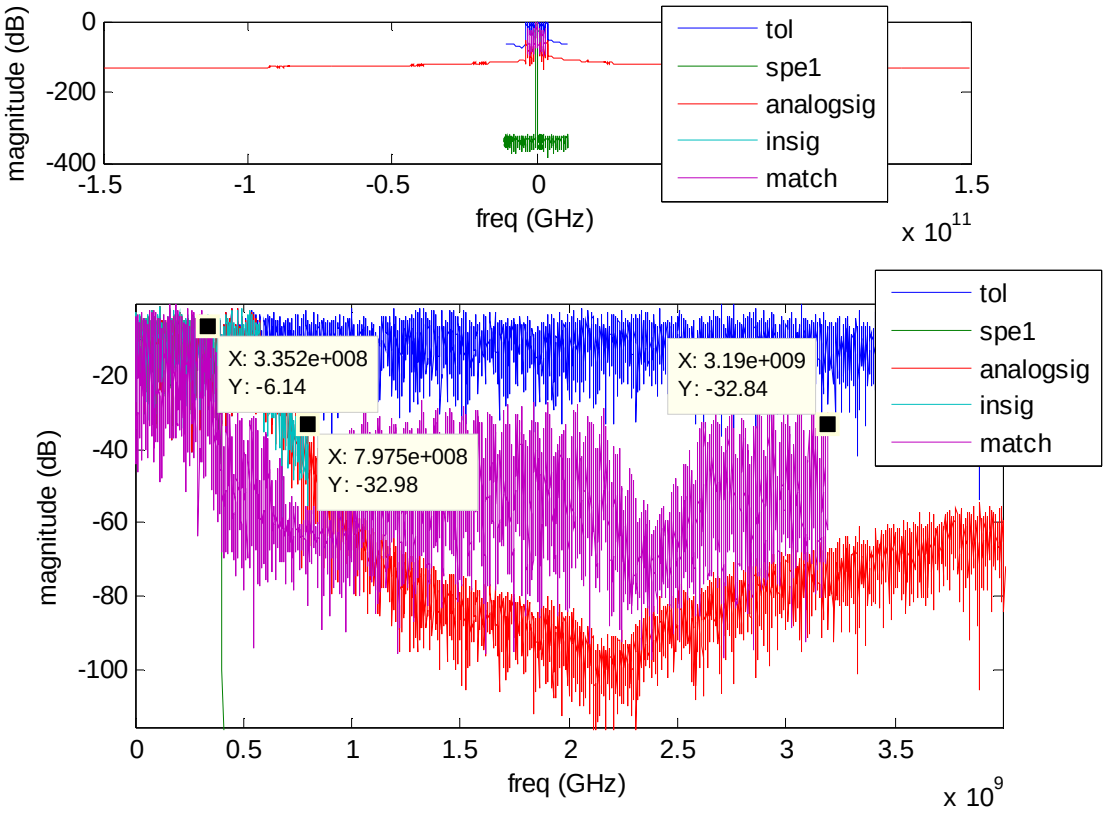


Figure 40 - Simulink Simulation Spectra Comparison (above: full; below: zoomed in)

The green spectrum represents the baseband signal. The system attempts to demodulate this sub-channel. The 11 channel Nyquist-WDM waveform (tol) is the blue waveform. The red spectrum displays the "analog" signal after oversampling and analog anti-alias filtering. The signal is considered analog due to the highly oversampled nature and the lack of

quantization. The signal is downsampled to recreate the 1.6 GHz ADC sampled spectrum and presented in teal in the above figure; note the marker at 800 MHz, the edge of the ADC sampling spectrum. The magenta spectrum displays the interpolated and match filtered signal. The magenta spectrum filters out the baseband signal with 32 dB of attenuation of the interpolate sidebands to 3.2 GHz, or 4 times 800 MHz.

An interesting plot to study is the QAM PLL shift over time. This plot, featured below, visualizes the PLL tracking the phase of the input signal and attempting to stay locked on. Like its analog counterpart, the digital PLL's first order and second order gain must be correctly tuned in to function correctly. Notice the flat response, followed by sudden jumps. The jumps represent the data stream resetting from the beginning. The PLL skips advances forward searching for the nulls in the derivative signals, overshoots slightly, then settles quickly. The two markers show the maximum (6.872) and minimum (4.596) peaks of one data block. Divide that difference by the interpolated number of samples per symbol (9.562), and the range over one period is 24.1%. On the far right edge of the figure, the plot spikes upward when the simulation input signal "runs out" and the PLL attempts to find the peaks of a DC signal.

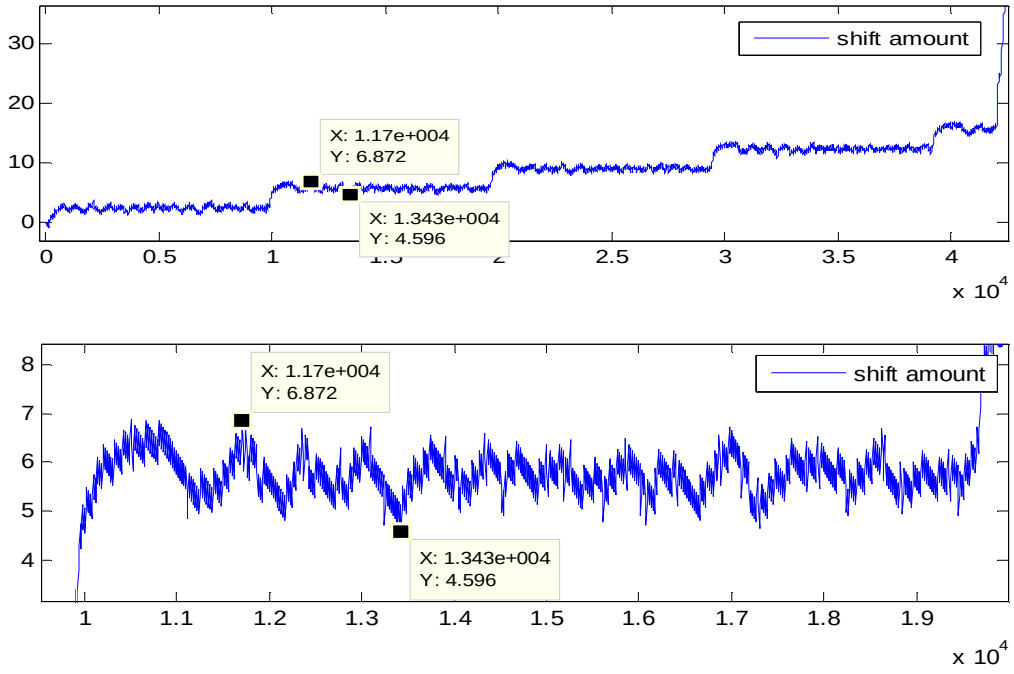


Figure 41 - QAM PLL Shift

The following scatter plots should be used to see the Viterbi-Viterbi algorithm and QAM PLL at work. The blue dots represent the complex match filter output, the green the output of the Viterbi-Viterbi phase correction block, and the red the output of the PLL reclocking function. The signal had a 36 degree rotation applied to it, as seen by the corner of the blue constellation in the 1st quadrant. Originally the corners of the transmitted constellation line up with the horizontal and vertical axes. The Viterbi-Viterbi algorithm performed its duties well, realigning the constellation on the I-Q axis.

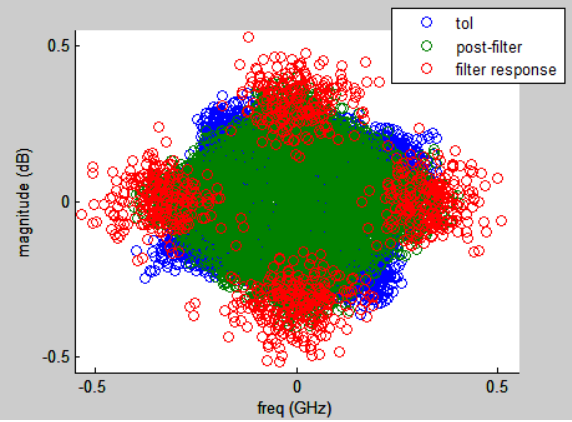


Figure 42 - Scatter Plot Comparison

The following is an example of a typical output from the Matlab command window:

```
shift 180 305
shift 180 8497
shift 180 16689
shift 180 24881
shift 180 33073
trigI 505
trigI 8697
trigI 16889
trigI 25081
trigI 33273
error count 14
error count 13
error count 15
error count 14
```

The 90 degree shift model executed a 180 degree shift, starting at counts 305, 8495, 16689, 24881, and 33073. In the following block, the data checker detected the beginning of the data block at counts 505, 8697, 16889, 25081, and 33273. Clearly, the two blocks detect the appropriate sequence 200 ticks apart. The difference arises from the delay block in the 90 degree shift model and the fact that the two models trigger on different strings; 11 bit Barker code and 13 bit Barker code. The associated error counts fill in at the end. Notice

the data checker triggered 5 times, yet only 4 error counts are listed. The final trigger, located at 33273, never reached the end of its data block, as the model is only set to run for 40,000 clock cycles. Therefore, the "end of string" signal never enabled, and consequently the overarching system would not accept the trial as complete, as it is on a per-trial basis.

Variables of every link of the digital signal processing system were altered and tested to analyze their effect on the bit error rate. The first studies compare differences with the match filtering stage. A few variables of note for the proceeding section are V&V ratio, in-phase, and N. The V&V ratio is the ratio of the Viterbi-Viterbi averaging length and the symbol period of the signal. For instance, if the Viterbi-Viterbi stage averages over 50 points and the symbol period is 10, the V&V ratio is 5. The in-phase is the input phase of the data with respect to the main axes and is given by radians over 2 pi. The term N refers to the match filter length. The first graph below displays the BER while varying N. Increasing the match filter order beyond 200 taps significantly reduces the gains.

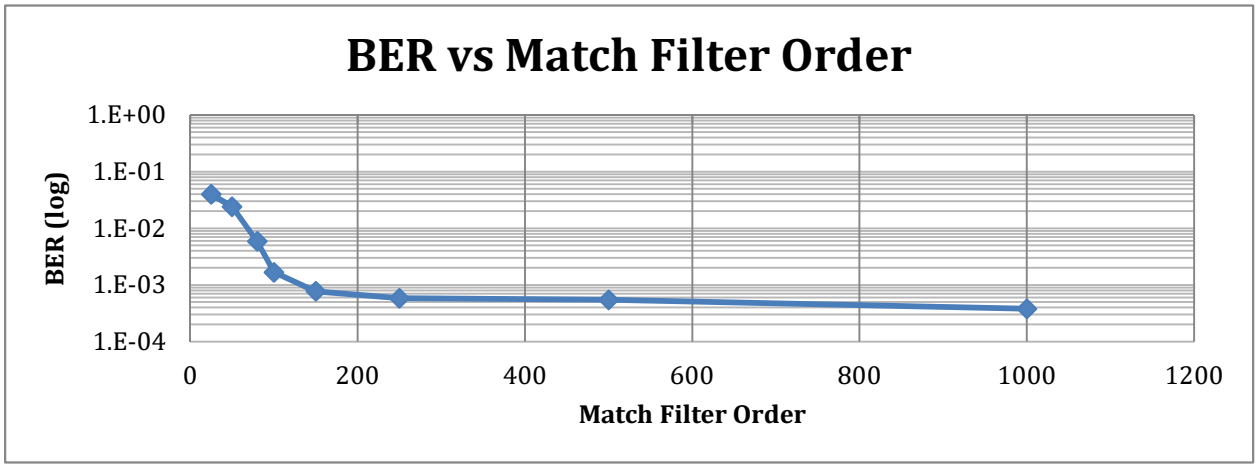


Figure 43 - BER vs Match Filter Order (N)

In a similar comparison, the match filter order was altered while differing the channel spacing. Changing the channel spacing maintains the bandwidth of each individual subcarrier while shifting the center frequency. A channel spacing of 100% will produce a signal where subcarriers have no extra bandwidth, whereas 130% would have a 15% spacing on each side of each subcarrier. The reduction of gains after 200 filter taps remains. Interestingly, once past 120% channel spacing is surpassed, filter order does not follow the same trend. This can be attributed to the fact that the minimal bandwidth from the adjacent subcarriers bleed into the desired channel. Increasing the channel spacing significantly defeats the purpose of Nyquist WDM, though, so it is preferable to utilize the signal processing to increase the bandwidth efficiency.

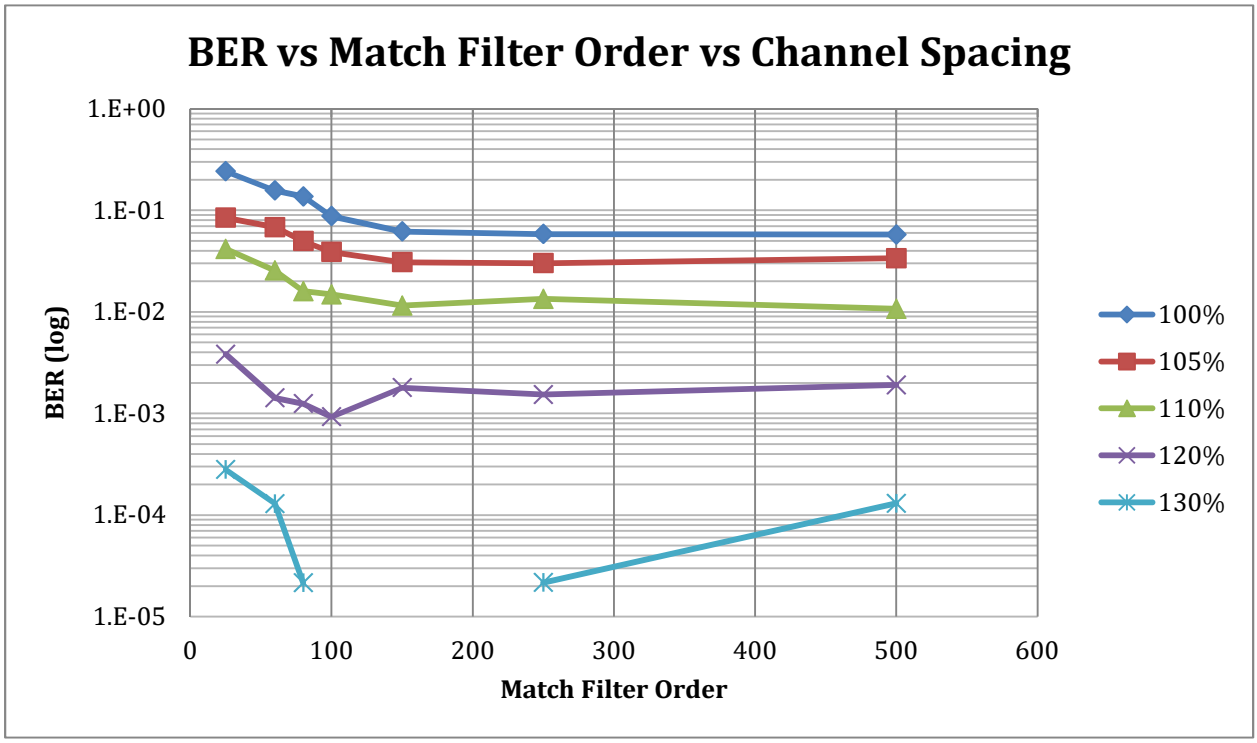


Figure 44 - BER vs N vs Channel Spacing

The next comparison takes on two different variables. The first is the "Percent Analog Filter Bandwidth". This parameter denotes the relative bandwidth of the baseband signal and the analog filter in place before the ADC. During this test, the bit per symbol was altered to expand and contract the bandwidth of the signal. Therefore, in the test, the higher signal bandwidth also represents a faster bit rate.

The results, shown in the graph below, represent the expected outcome. Tracking along the x-axis, the signal first utilizes 32% of the analog filter bandwidth, continuing onto 100% and beyond. The results reveal a decline in BER from approximately the 32% point to the 60% point. The BER remains relatively flat from 60% to 100% of the filter bandwidth, then dramatically increases as the baseband signal expands into the stopband of the analog filter.

The nature of Nyquist-DWM relies on a dramatic and effective filtering scheme to remove power from the adjacent channels from the desired. When the baseband signal takes up a mere 35% of the analog filter, that filter allows a large amount of power from the adjacent subchannels to be digitizes. This creates a situation where the DSP is completely relied upon in the filtering process. From 60%-100%, the complete analog and digital system works in harmony to eradicate the undesired signals from the desired. In other words, the analog filter is employed to both filter out higher band signals that would create aliasing problems as well as helping to filter out the adjacent subchannels. The increase in BER as the signal bandwidth increases beyond the analog filter bandwidth clearly stems from the fact that the analog filter is attenuating the desired signal.

The second variable, represented by the different lines, changes the Viterbi-Viterbi ratio. As previously discussed, the V&V Ratio is the length of the summation function to the digitized points per symbol; eg, a received digital signal of 10 points per symbol per channel and Viterbi-Viterbi function which sums over 20 points would result in a V&V Ratio of 2. This graph shows a mostly predictable outcome with a strange aspect. As the test increases from a V&V Ratio of 1 to 16, the BER predictably decreases monotonically. The irregularity in the trend arises at the V&V Ratio of 0; ie, no averaging. Denoted in green triangles, this line, looking left to right, begins at a predictable BER, but proceeds to follow a path between V&V Ratios 4 and 8 as the bandwidth of the signal increases. To further study this concept, another test was conducted.

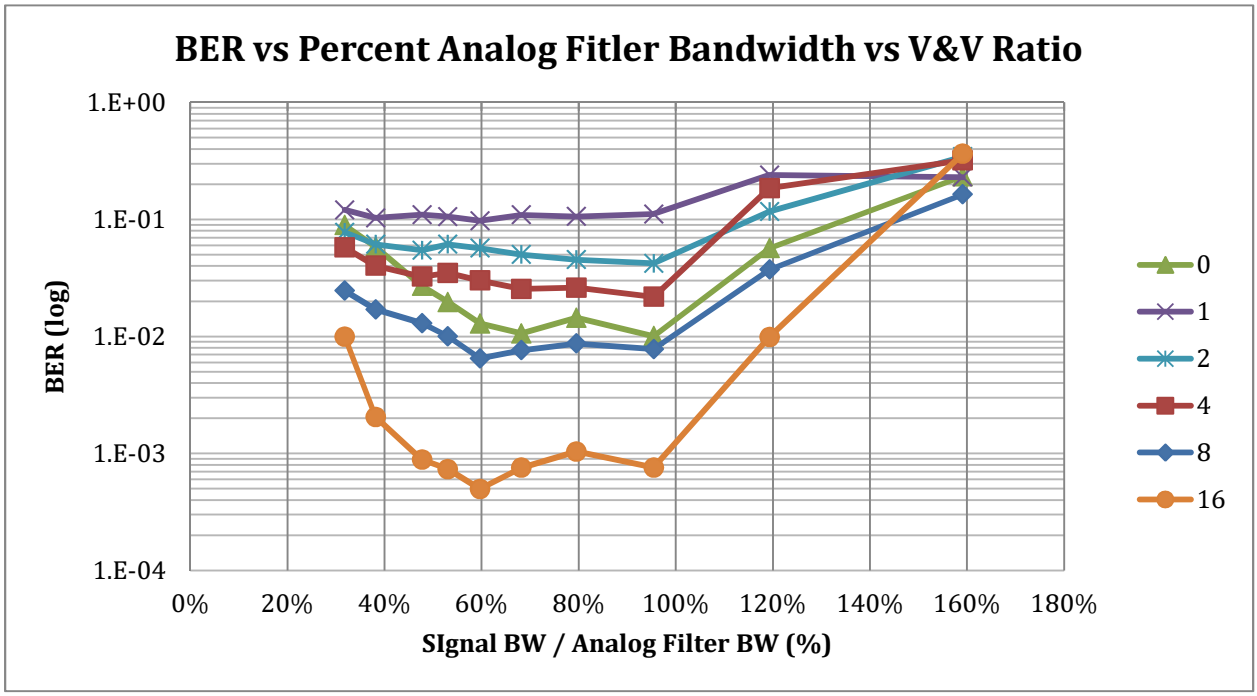


Figure 45 - BER vs % Analog BW vs V&V Ratio

A key aspect to any one trial of the system is the input phase angle. The previous tests all processed a received signal with a fixed input angle of 0.083*2*pi, or 1/6*pi. Locking the phase angle as a constant removed that extra variable to be able to study one factor at a time. The following chart displays the affect the input phase angle has on the BER, as well as comparing different V&V Ratios.

This chart displays several interesting features of the input phase angle correction algorithm of the receiver. Firstly, all the lines exhibit an almost perfectly logarithmic (viewed linearly on this logarithmic chart) trend. Reading the chart from right to left, they all stem from a BER of 0.5 at .125*2*pi input angle and branch out as expected with the higher V&V Ratios having a steeper decline. Once again the 0 V&V Ratio line, represented with blue diamonds, crosses paths and does not follow the same trend. Furthermore, the BERs reported towards the higher input phase angles are completely unacceptable for a production system.

The 0 V&V Ratio trendline imparts that the system should employ a V&V Ratio of at least 16 if one will be used at all. As replicated above, the 0 V&V Ratio line is nestled between the 4 and 8 at 0.83*2*pi. Beyond that point, the line appears to follow a parallel path of the 16 line. The .125*2*pi BER arises from the fact that that angle lies right on the division of two constellation zones. The V&V algorithm must make a decision at any point in time to shift the current point 45 degrees clockwise or counterclockwise. Being on the division line, it will statistically be incorrect 50% of the time, hence the BER of 0.5. A realtime, continuous receiver system would utilize a phase tracking system such that the phase from the previous packet would be carried over and adjusted accordingly. This

system breaks the data chain between stored arrays, due to hardware limitations this simulation is meant to study, and therefore no tracking can be implemented.

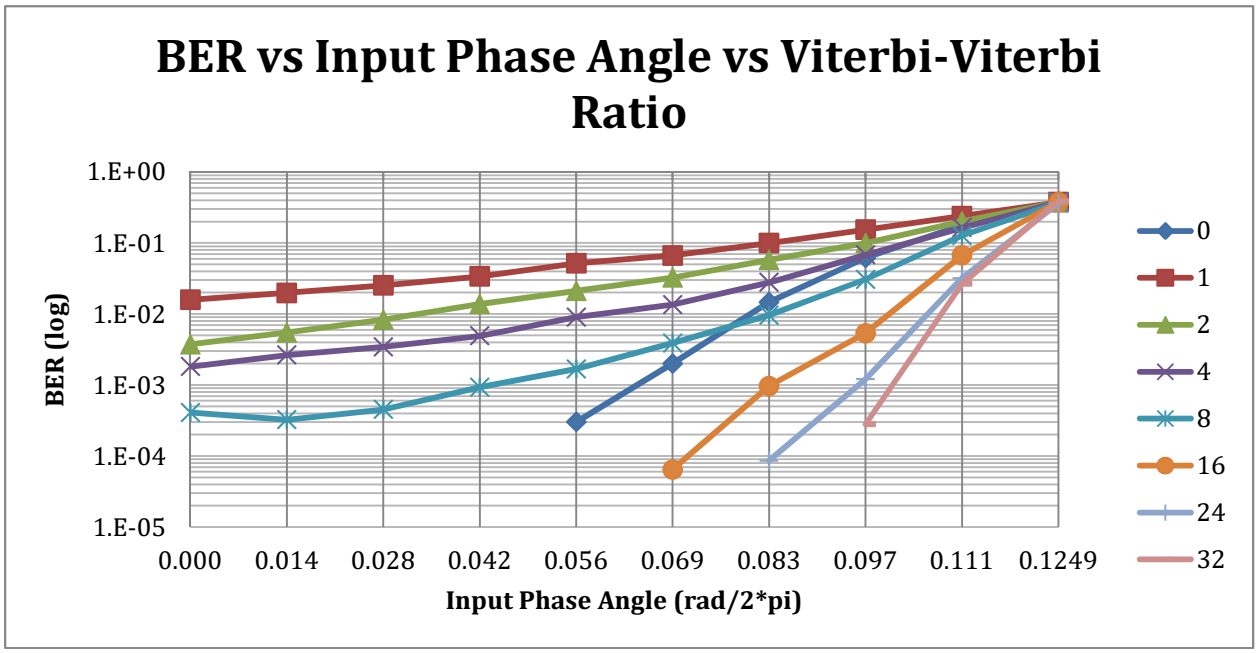


Figure 46 - BER vs Input Phase Angle vs V&V Ratio

All associated data for the graphs above is included in Appendix E.

Post-Processing of Data from a Real Fiber-Optic System

In the previous section, the data remained in Matlab for the entirety of the simulation. It is important to confirm the functionality of the developed real-time DSCM receiver algorithms are effective when processing waveforms undergoing the distortions and irregularities brought on by practical components such as an optical transmitter, optical fiber, and ADC. This is an important test to further validate the design of the Matlab Simulink models prior to implementing them onto the real FPGA.

The same functions above generated the baseband (DataIn1_maker.m) and full channel data waveform (Tx_func.m) to use in the real system. To comply with the transmitter hardware requirements, the waveform was quantized to integers from 0-31, as well as having a data length of 32,768 per I and Q channel. The imag() and real() Matlab functions split the waveform into the I and Q waveforms for storage in the transmitter. The transmit function modulated the binary stream at a rate of 32 points per symbol, or 16 points per bit, to mimic the optimized "percent analog filter bandwidth" during the simulation trials.

The hardware transmitter modulates the complex signal onto a laser light source at a rate of approximately 21.4 GHz, producing a waveform with 11 1.3 Gb/s subchannels. The signal travels through SM optical fiber. At the receiver, coherent homodyne detection downshifts the desired signal directly to baseband by mixing the optical signal with the local oscillator. Due to the lack of independent laser sources, a portion of the optical power from the transmit laser was sent to the receiver as the local oscillator. This setup assures

correct frequency matching, and the coherent detection is truly homodyne. After the 90 degree optical hybrid, two photodiodes covert the light signal to electric domain, then two amplifiers boost the in-phase and quadrature signals. The electrical signals are filtered by the analog anti-aliasing filters and an oscilloscope saves the output waveforms at 25 Gs/s sampling rate. The script *ReceiverPP.m* loads (after transfer with a USB thumbstick) and processes the data.

The post-processing and simulation scripts only have 2 differences. The input data rate of the simulation was the transmitter sampling rate (~21 GHz) and the input data rate of the post-processing script is 25 GHz. Secondly, the analog filter function no longer needs to be simulated since it was physically implemented. Therefore, *chebysfilter.m* is neglected.

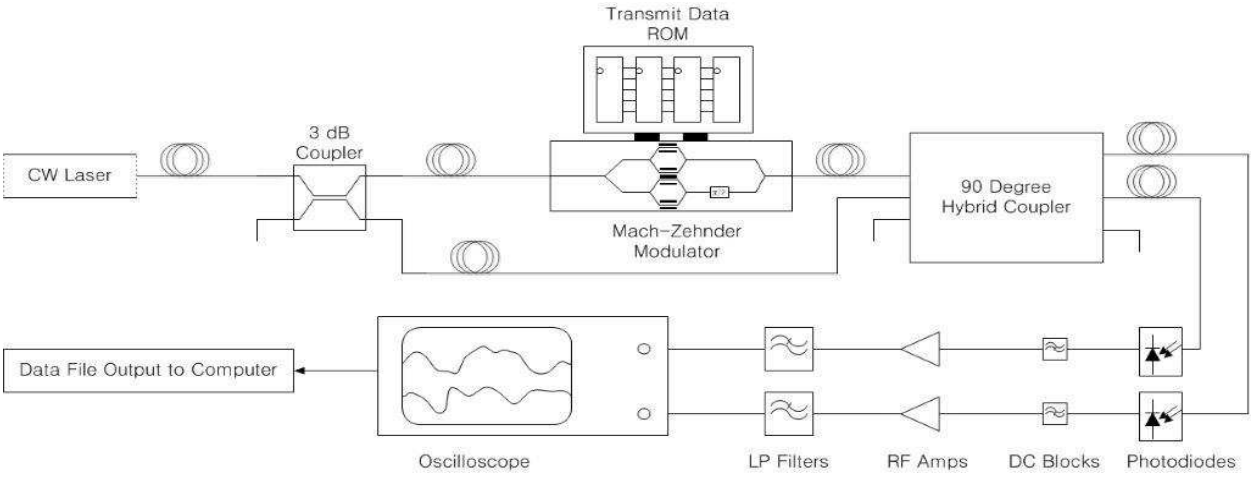


Figure 47 - Test Setup

Due to resource scarcity on the FPGA, the filters and DSP blocks had to be properly sized to find the optimal blend of analog bandwidth usage, filter order, and VV ratio. The filters can be put in an "overclocked" state, which trades resources for input clock rate. In other words, the filters would run at a much higher clock rate than the data rate. The most

scarce and most needed slice of the FPGA is the "DSP slice". On the particular FPGA chosen for this project, only 48 DSP slices are available. Without overclocking the FPGA would only allow for one 48 tap filter. Another factor to consider is the match filter also operates as an interpolator. That means that the filter must always run at a rate 4 times higher than the input data.

The DSP slice resource must be shared by the match filter, derivative filter, and Barker sequence detectors. The simulations done previously expressed that at least a 100 tap filter should be used for the match filter. An examination of the core generator software provided by Xilinx revealed that the resource savings were almost proportional to the OC ratio. For instance, an overclock ratio of 16 yielded 13 times less DSP slice usage. After careful consideration of this and the previous simulations, the match filter was chosen to have 100 taps at 32 times overclock of the input data rate. The derivation filter contains 50 taps at the same speed. Since the derivative process occurs after interpolation, the derivative filter is only 8 times faster than its input data rate.

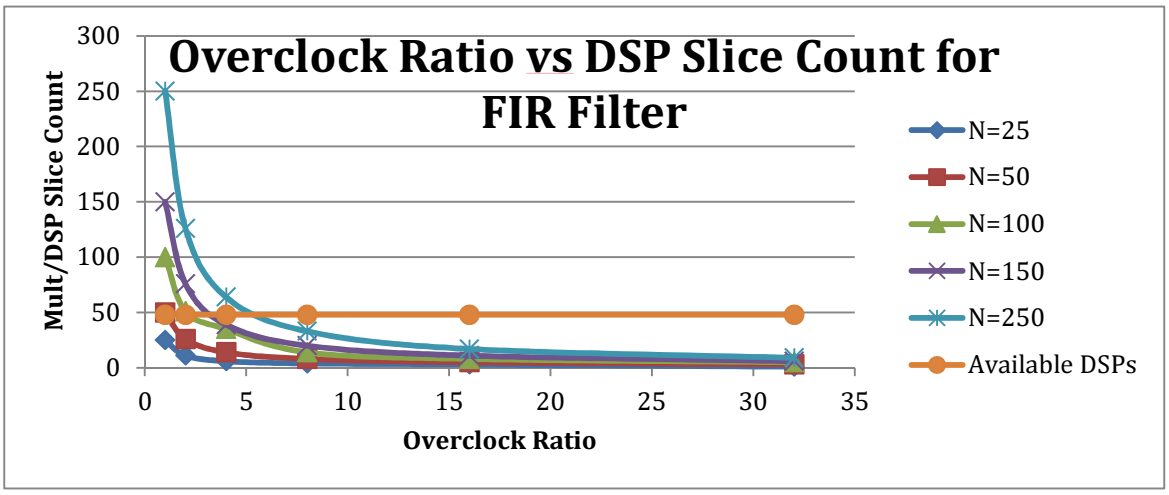


Figure 48 - Resource Savings with Higher Overclocking Rates

Post-Processing Results

The first operation of the receiver after digitizing the signal is filtering. As previously discussed, the N-WDM match filter also interpolates the by 4 to aid in the resampling process. The figure below displays a waveform of real data recorded by the oscilloscope after being transmitted, modulated, downshifted, and sampled. The effect of the filtering is obvious: the waveform becomes much smoother and representative of a QAM signal. The third graph is provided to easily visualize the interpolation.

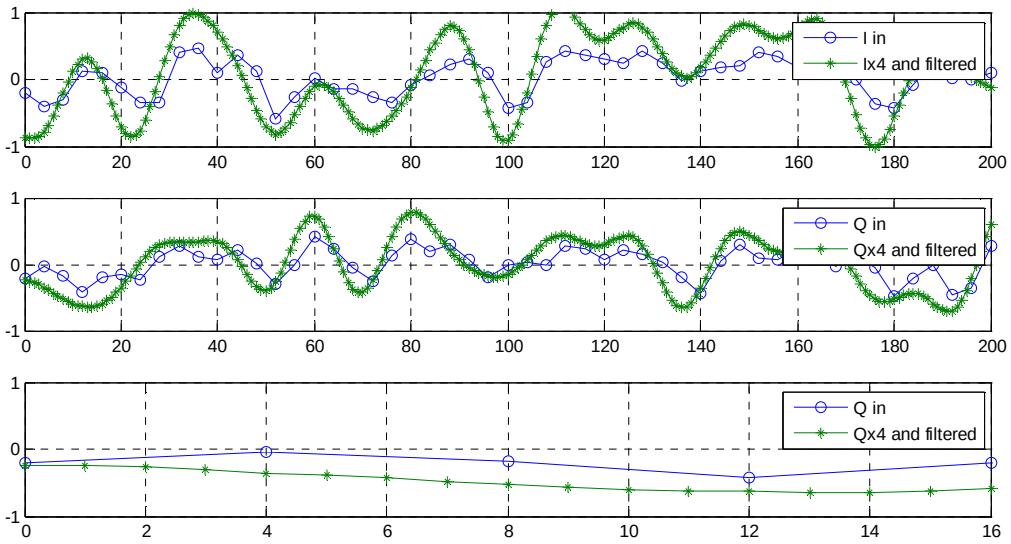


Figure 49 - Input & Filtered Data vs Time

The match filtered data then passes through a derivative filter to drive the PLL. The plot below shows how well the derivative filtering process works. Two key points are marked. The point at 1388 represents a local maximum for the filtered signal, and the

derivative signal passes through the x-axis as expected. At 1405, an inflection point can be seen in the main signal and by the derivative's local max.

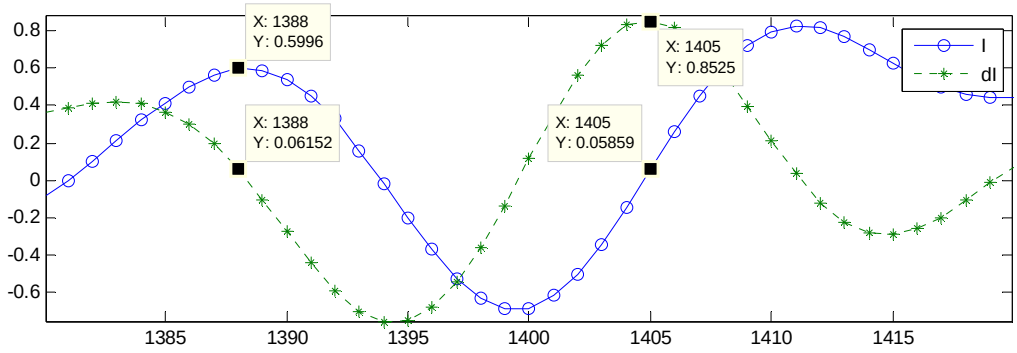


Figure 50 - Filtered and Differentiated Signals vs Time

The input spectrum from the real data looks similar to that of the simulation data, but noisier. This is expected of a real system, having noise sources like laser phase noise, modulator inaccuracies, quantization errors, and so on. The figure below displays the input spectrum (blue) and the output data from the filter stage (dashed-green). The green spectrum has undergone the Nyquist-WDM matched filtering as well as 4x oversampling.

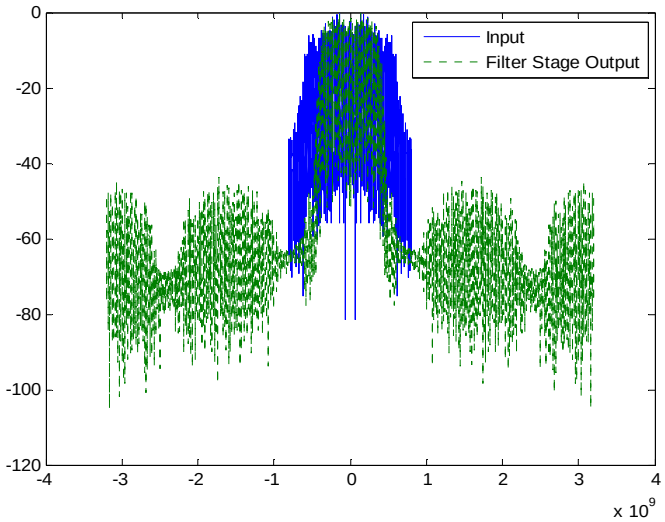


Figure 51 – Input & Filtered Data (dB) vs Frequency (GHz)

The output of the QAM-PLL module provided an interesting result. As with the simulated data, the best way to visualize the output is by viewing the PLL shift quantity. The pure simulation provided an output which produced an easily spotted jump at the transition between every data block. The same variable of the QAM PLL plotted when processing real data produces no such jump. This could be attributed to the smoothing effect of a modulator, photodetector, and ADC. As seen on the graph, the QAM PLL is able to track the peak of the input waveform by approximately +-1 point. With a digitized bit per symbol of approximately 9.8, this represents a 10% tracking accuracy.

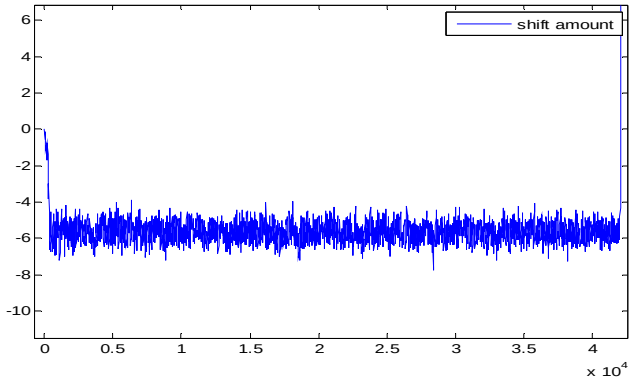


Figure 52 - QAM PLL Shift Amount

The test was performed 4 times each for 2 different channel spacings. Twice the system produced a null result, meaning it could not identify the beginning of the sequence by sensing the Barker code. The figure below on the left displays the data set would could not be correctly decoded. The blue constellation shows the input data almost perfectly at 45 degrees, the worst angle for this system to decode. The V&V operation produces a constellation, in green, almost perfectly on axis, but unfortunately the system most

definitely shifted points which should've have been adjacent to separate quadrants. The constellations on the right display a more ideal input data set.

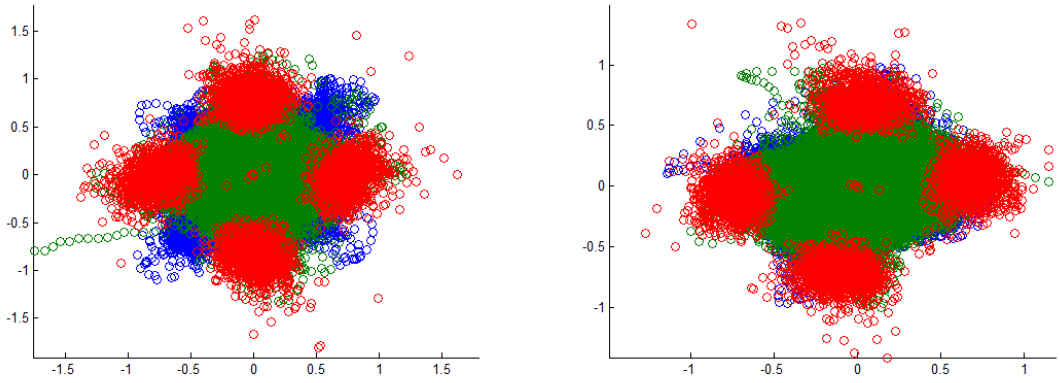


Figure 53 - V&V and QAM PLL Constellations vs Input Data (125% Channel Spacing)

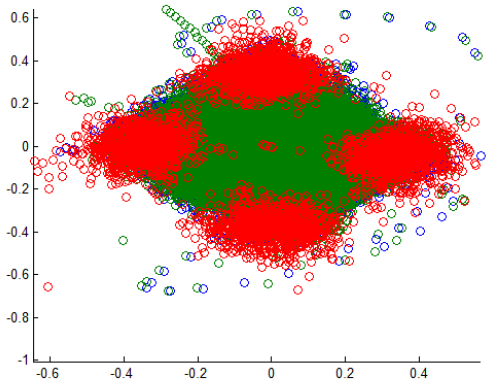


Figure 54 - V&V and QAM PLL Constellations vs Input Data (115% Channel Spacing)

The QAM PLL outputs, which feel the QAM decoder, from the two channel spacings above show how helpful 10% more bandwidth is to the Nyquist-WDM system. This bandwidth is crucial for a system which relies on an under resourced match filter and phase tracking system.

The test system used a limit resources, being just large enough to fit on the physical FPGA provided for this project. The combination match and interpolation filter is order

100 and the derivative filter is order 50. The system utilizes a 4x interpolation function as in all the previous tests. The V&V summation averages over 64 points. With a interpolated points per symbol of 9.8, this gives a V&V Ratio of ~6.5. The normalized bandwidth of the PLL loop, BnT, is 2.5119e-3 and the loop gain factor, Kp, is 1.5849e-3.

For a more accurate comparison of real data and simulated data, the simulation system was run 20 times at 115% and 130% channel spacing with all of the same parameters at the real data system. Furthermore, the input phase angle was completely randomized to simulate the input angle of the real data system. The trials were simulated 20 times for each channel spacing and suffered 2 trials with null results.

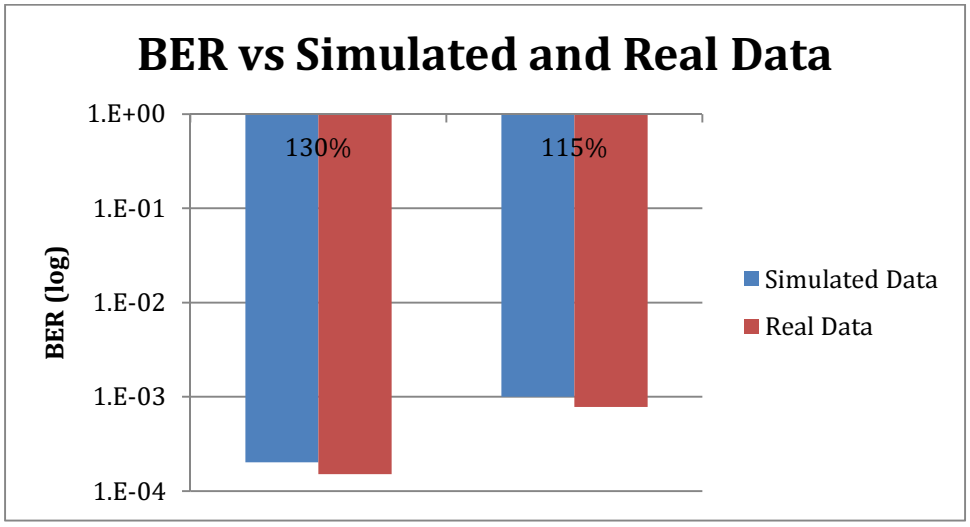


Figure 55 - Comparison of Real and Simulated Data

The system performed almost just as well with the simulated data as with the actual transmitted and digitized data. The BER suffered only a 0.12 and 0.11 (log10) penalty for the 130% and 115% tests. The BER for the 130% and 115% channel spacing trials was 1e-3.70 and 1e-3.00, respectively.

Hardware Implementation Perspective

ADC12D1600RB

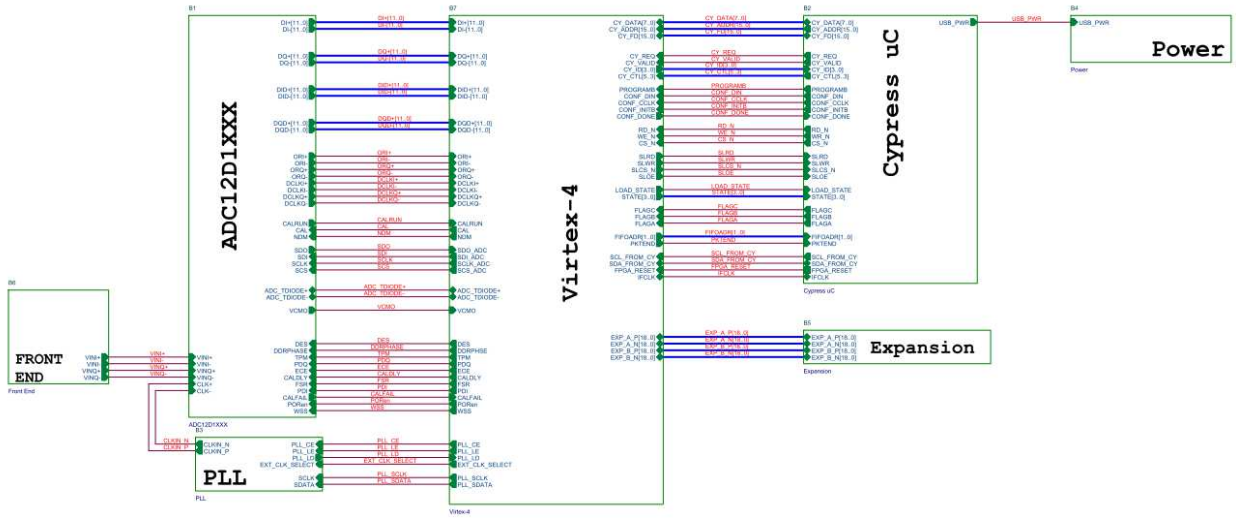


Figure 56 - ADC12D1600RB Top Level Design

The main electronics board used in this project is the ADC12D1600RB. It is a 12-bit, dual 1.6 GHz analog-to-digital converter reference board. The board includes a Xilinx Virtex-4 FPGA and an on-board USB controller (Cypress uC in Figure 56). An on-board PLL helps to lock onto a crystal and increase clock rate to be used at the ADC sampling clock. Affixed with the necessary power regulation system, LEDs, jumpers, and SMA terminals, this board comes ready to use.

Since this is a reference board setup, the FPGA comes preloaded with a demo program to evaluate the ADC. The FPGA was used only to sort the data from the ADC and transmit the data to a PC through the USB controller in the prepackaged system. The included PC software, WaveVision5, calculates the FFT and allows the user to view the

incoming analog waveform. In this project, however, the FPGA will not only sort the data, but process it as well. The USB controller will go un-used.

As discussed earlier, there was a need to overclock the filter stage to free up enough resources to adequately process the data. An insurmountable problem arises from the overclocking process in the real system. The FPGA can only function in the 400 MHz range. The input data stream from the ADC is 1.6 GHz. Therefore, the FPGA obviously cannot support clock speed 32 times the input data rate. As a result, the input data will be cached and blocks of data will be processed at a time. This was not the intent of the system, which planned to be real-time, but the resources available on the FPGA held the project back.

ADC12D1600

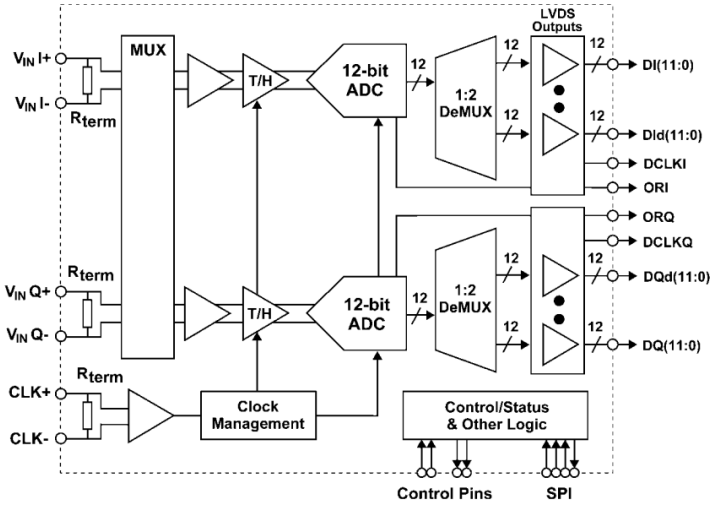


Figure 57 - ADC12D1600 Functional Block Diagram

The ADC12D1600 is a dual channel analog to digital converter. The chip is highly configurable, with the ability to operate in a single channel mode sampling at up to 3.2 GHz or in a dual channel mode sampling at up to 1.6 GHz per channel. The dual channel setup shares a single clock, but can be configured to sample on the same or separate edges of the clock. The ADC has differential analog inputs, as well as a differential clock input. The outputs are also all in a differential setup. The ADC's main settings can be configured with simple jumpers, but advanced configurations (extended control mode) require the use of a serial interface to write to registers.

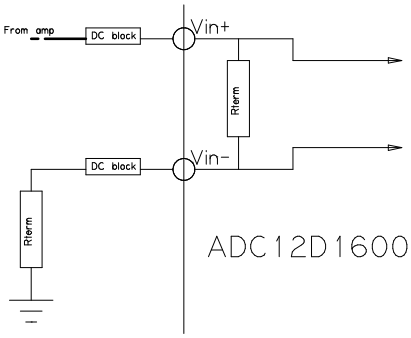


Figure 58 – Single Channel Analog Input Configuration

For this project, the ADC12D1600 operates in the dual channel, same-edge sampling mode. The analog inputs are setup in single-ended configuration shown above in Figure 58. Ideally, the analog input signal would be balanced, but due to the lack of a balanced photodetector this is impossible. The full scale range (FSR) pin is set low, fixing the analog input range to 600 mV_{p-p} (National Semiconductor Corporation, 2011). The outputs of the ADC are demuxed, then processed by a DDR converter, to produce 2 12-bit DDR signals per channel. The data must then be organized by whichever chip receives the signal, and the process to do so is discussed in the Implementation section, under ***demux_fifo_sort.vhd***.

Dual 1.0/1.6 GSPS ADC, Fin = 125 MHz	
■ ENOB	9.6/9.4 Bits (typ)
■ SNR	60.2/58.5 dB (typ)
■ SFDR	71/70.3 dBc (typ)
■ Power	3.38/3.88W (typ)
■ Full Power Bandwidth	2.8/2.8GHz (typ)

Table 7 - Key Specifications of the ADC12D1600

The ADC12D1600 datasheet reports having 9.4 effective number of bits (National Semiconductor Corporation, 2011). The SNR and spurious-free dynamic ranges are 58.5 dB and 70.3 dBc, respectively, when sampling at 1/6 GHz.

Xilinx Virtex-4 (XC4VLX25)

Device	Configurable Logic Blocks (CLBs)[1]				XtremeDSP Slices[2]	Block RAM		DCMs	PMCDs	PowerPC Processor Blocks	Ethernet MACs	RocketIO Transceiver Blocks	Total I/O Banks	Max User I/O
	Array[3] Row x Col	Logic Cells	Slices	Max Distributed RAM (Kb)		18 Kb Blocks	Max Block RAM (Kb)							
XC4VLX15	64 x 24	13,824	6,144	96	32	48	864	4	0	N/A	N/A	N/A	9	320
XC4VLX25	96 x 28	24,192	10,752	168	48	72	1,296	8	4	N/A	N/A	N/A	11	448
XC4VLX40	128 x 36	41,472	18,432	288	64	96	1,728	8	4	N/A	N/A	N/A	13	640
XC4VLX60	128 x 52	59,904	26,624	416	64	160	2,880	8	4	N/A	N/A	N/A	13	640
XC4VLX80	160 x 56	80,640	35,840	560	80	200	3,600	12	8	N/A	N/A	N/A	15	768
XC4VLX100	192 x 64	110,592	49,152	768	96	240	4,320	12	8	N/A	N/A	N/A	17	960
XC4VLX160	192 x 88	152,064	67,584	1056	96	288	5,184	12	8	N/A	N/A	N/A	17	960
XC4VLX200	192 x 116	200,448	89,088	1392	96	336	6,048	12	8	N/A	N/A	N/A	17	960

Table 8 - Virtex 4 LX Model Comparison

The ADC reference board includes the Virtex-4 FPGA. Although this FPGA is 3 generations old, it is still a capable chip. The chip installed on this reference board, as noted in the above table, contains a relatively small amount of logic cells, block and distributed RAM, XtremeDSP slices, and other resources compared to the rest of its family, as well as the following generations.

PLL (LMX2531 LQ1570E)

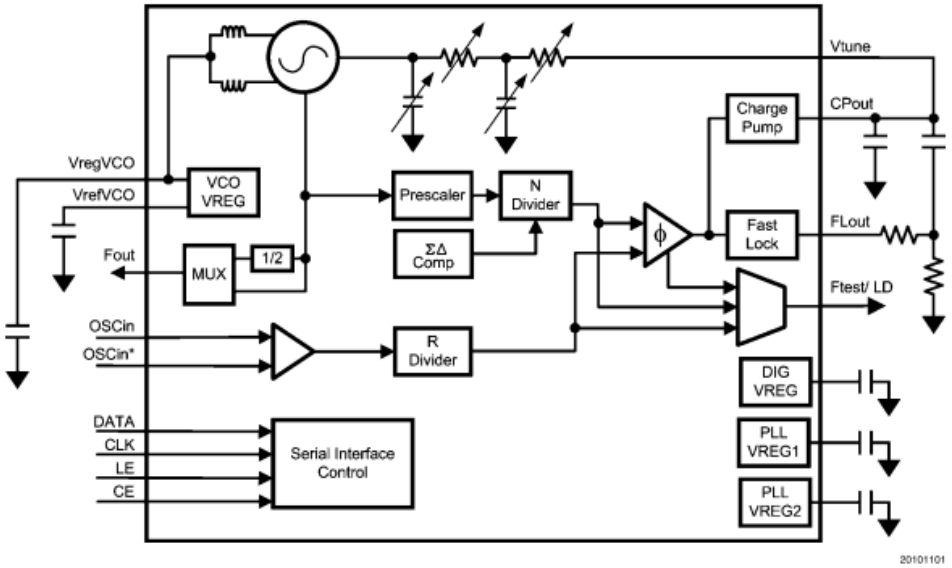


Figure 59 - LMX2351LQ1570E Functional Block Diagram

The LMX2531 is a high performance frequency synthesizer with an integrated VCO. This chip takes in a reference signal and produces an output of virtually any frequency between 765-818 MHz or 1530-1636 MHz. The chip is programmed via serial interface by the user to configure the frequency, filter, selectable outputs, and more.

The functionality of this chip begins by producing a phase detector frequency f_{PD} (Texas Instruments, 2013). The phase detector frequency is given by

$$f_{PD} = f_{OSCin}/R \qquad \text{Equation 47}$$

where R is the "divider value." Tradeoffs of maximum output frequency and phase noise exist for the R values, which are restricted to 1, 2, 4, 8, 16, and 32. The other variable the user has to control the output frequency consist of an integer and fractional values, given by

$$N = N_{integer} + N_{fractional} \qquad \text{Equation 48}$$

to produce a multiplier value precise to the $1/4194303^{rd}$. The integer value is an 11 bit number, and the fractional part consists of a 22 bit numerator and 22 bit denominator. The resulting output frequency is

$$f_{F_{out}} = \frac{f_{VCO}}{D} = f_{PD} * N = f_{OSCin} * N/R \quad \text{Equation 49}$$

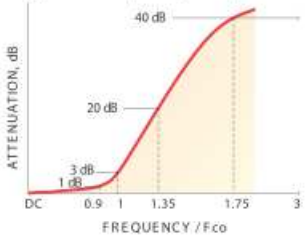
where D is 1 or 2, providing low band frequency options.

BLP-550+

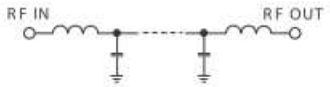
Low Pass Filter Electrical Specifications

PASSBAND (MHz)	fco (MHz) Nom.	STOPBAND (MHz)		VSWR (:1)	
				Passband Typ.	Stopband Typ.
(loss < 1 dB)	(loss 3 dB)	(loss > 20 dB)	(loss > 40 dB)		
DC-520	570	750-920	920-2000	1.7	18

typical frequency response



electrical schematic



Typical Performance Data

Frequency (MHz)	Insertion Loss (dB)		Return Loss (dB)	Frequency (MHz)	Group Delay (nsec)
	$\bar{x}$	σ			
10.00	0.02	0.1	44.6	10.00	1.55
152.50	0.10	0.1	28.0	80.00	1.61
222.50	0.15	0.1	28.8	152.50	1.61
295.00	0.27	0.1	18.0	222.50	1.61
365.00	0.38	0.1	15.3	295.00	1.75
437.50	0.36	0.1	19.6	365.00	1.93
520.00	0.57	0.1	20.4	437.50	2.32
550.00	0.75	0.1	20.8	520.00	2.99
570.00	1.17	0.3	12.7	540.00	3.34
580.00	1.68	0.4	9.2	550.00	3.66
620.00	7.07	1.1	2.3	570.00	4.37
720.00	26.51	1.6	0.2	580.00	4.59
730.00	28.52	1.6	0.3	600.00	4.68
740.00	29.95	1.6	0.3	620.00	3.72
750.00	31.57	1.6	0.4	650.00	2.39
760.00	33.24	1.7	0.4	720.00	2.08
800.00	39.85	1.9	0.3	730.00	0.04
870.00	50.78	2.4	0.2	740.00	0.99
900.00	55.60	2.9	0.2	750.00	0.81
910.00	58.10	4.7	0.2	760.00	0.89
920.00	59.19	4.2	0.2	780.00	1.49
1147.50	68.70	3.0	0.2	800.00	1.71
1290.00	68.19	3.5	0.1	850.00	2.48
1432.50	69.02	4.4	0.1	870.00	2.81
1572.50	75.22	5.7	0.1	900.00	1.61
1715.00	73.18	5.2	0.1	910.00	0.91
1787.50	71.84	3.4	0.1	920.00	0.83
1857.50	71.44	3.7	0.1	1147.50	0.80
1930.00	71.78	6.9	0.1	1217.50	0.73
2000.00	68.82	2.1	0.1	1290.00	0.65

Table 9 - Key Specifications of the BLP-550+

The ADC, as discussed previously, samples at a rate of 1.6 GHz. Therefore, the 1st Nyquist region exists from DC-800 MHz. The BLP-550+ analog filter was chosen for its wide, flat passband and steep rolloff. As noted in the above table, typical specifications state at 800 MHz the attenuation is ~40 dB. The desired signal bandwidth will be contained within DC-550 MHz to avoid a greater than 1 dB penalty. The analog filter not

only acts as an anti-aliasing filter, but also attenuates the adjacent channels within the sampling Nyquist rate.

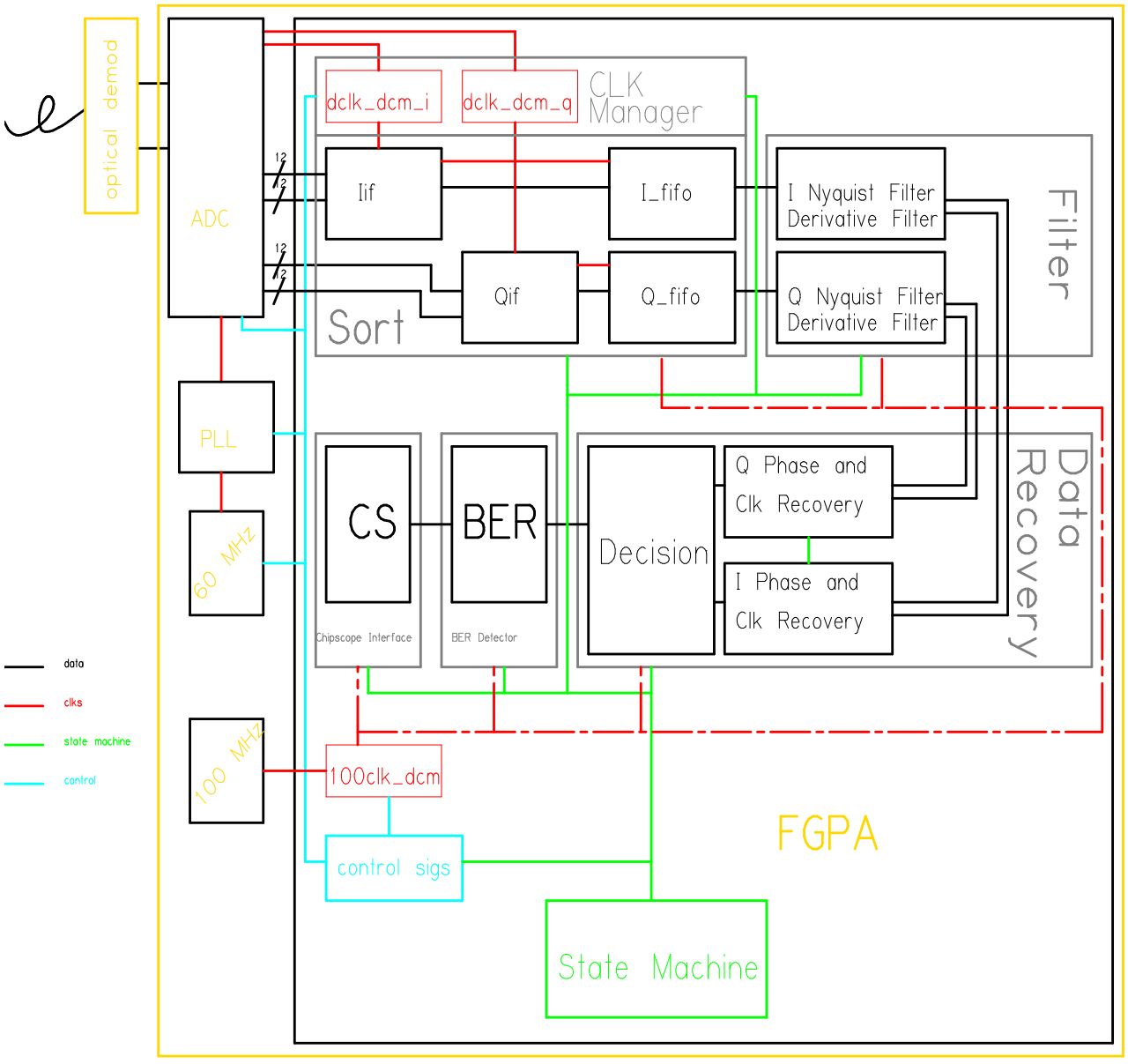


Figure 60 - Top Level Block Diagram

VHDL-only Components

The following files were designed, created, and implemented fully in VHDL. Any labeled signals in the vhdl block diagrams have the same name as in the program itself for debugging and alteration purposes.

dclk_dcm.vhd

This component implements the primitive "DCM_ADV," or Advanced Digital Clock Manager Circuit. It is a configurable DLL. In this instance, the component locks onto the ADC's in-phase and quadrature data clocks. These clock traces, as most the signals from the ADC, are differential signals. Therefore, a differential clock buffer is instantiated prior to the DCM. The clock period is set to 2.5 (ns), and the DCM is set to "MAX_SPEED" mode, as this is towards the top frequency for this component. This component was generated by the Xilinx Architecture Wizard.

clk100_dcm.vhd

This component is similar to the above, instead dedicated to the single ended 100 MHz signal. The input signal originates from a USB controller (unused) and, being single-ended, only needs a single-ended buffer. Due to the different data rates in the data acquisition and analyzing stages, a 6.25 MHz (100 MHz/16) signal is also created by the DCM utilizing the "CLKDV_DIVIDE" parameter.

genericcounter.vhd

This is simple modifiable counter. The inputs are a clock source, a clear signal, and constant integer (N) for how many bits wide the output should be. The output is an N bit vector continuously counting at a rate of half the input clock.

PLL_init.vhd

The ADC's clock is driven by an external frequency synthesizer which uses a 60 MHz crystal as a clock source. The desired clock speed of the ADC is 1.6 GHz, so the PLL must multiply the clock speed by a ratio of 1.6e9/60e6=26.6667. The phase detector of the PLL (as per the datasheet) performs optimally with a reference frequency of about 2.5 MHz. The phase detector frequency is given by

$$f_{PD} = f_{OSC_{in}}/R \qquad \text{Equation 50}$$

where OSC_{in} is 60 MHz and R is the phase detector divider rate. R is set to 16, resulting in a f_{PD}=3.75 MHz. The VCO output is given by

$$f_{VCO} = \frac{f_{OSC_{in}} * N}{R} = f_{PD} * N \qquad \text{Equation 51}$$

where N is the counter value to increase the frequency. N is set to 426.6667, creating the desired output frequency of 1.6 GHz.

The R, N, as well as other values, are set via a serial data interface from the FPGA to the PLL. There are 13 registers to be programmed, each with 20 bits of data. The interface is a standard 3 pin system with a data line, clock, and load enable signal. As seen in Figure 61 below, the data for the register in clocked in first, followed by the register address. The load enable signal must be logic low to start and continue the input process, then logic high to enter the stored value into the specified register. The 6.25 MHz clock runs this process. The R and N values are programmed as stated above and all other registers are programmed with the default settings as according to the datasheet.

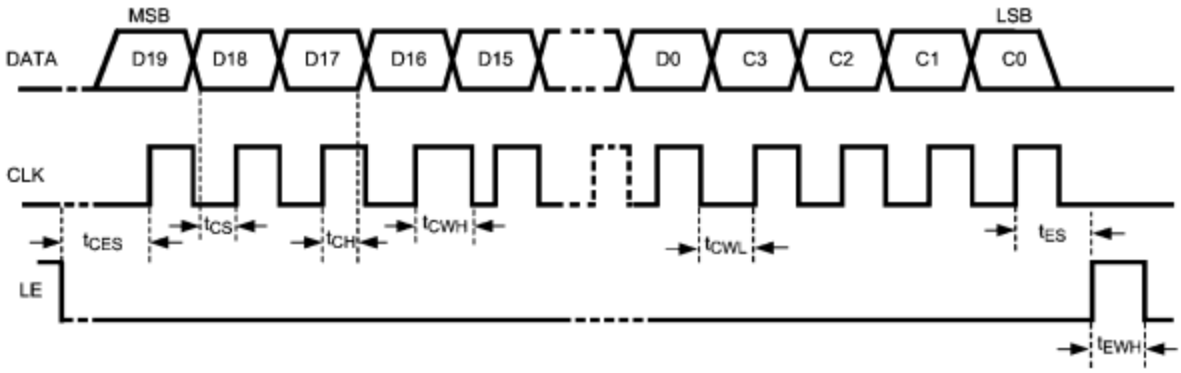


Figure 61 - Serial Interface Diagram

adcDATAinterface.vhd

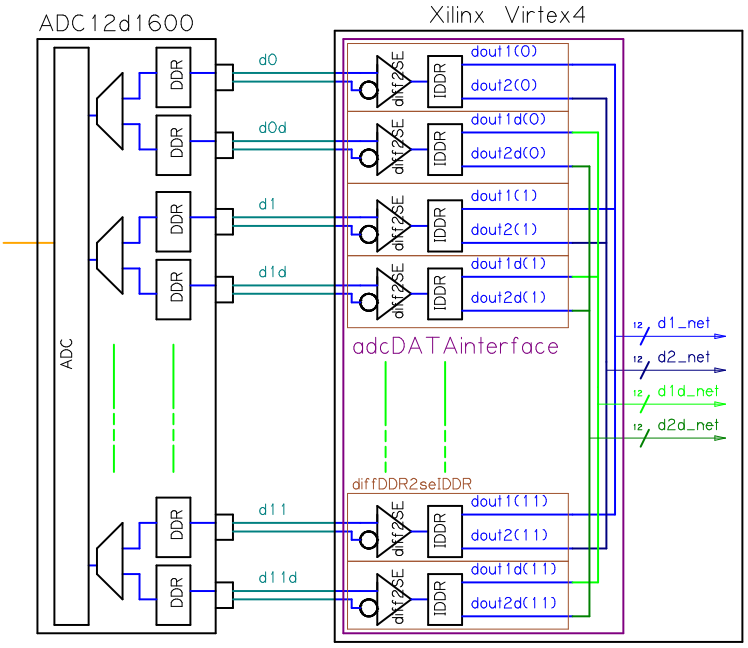


Figure 62 - Single Channel adcDATAinterface.vhd Block Diagram

The data lines from the analog to digital converter to the FPGA are differential signals. The data is also sent as dual data rate (DDR) and demuxed into 2 signals. This component converts the differential signals to single-ended and implements inverse DDR

(IDDR), outputting 4 singled ended signals. For organization's sake, *adcDATAinterface* instantiates 12 components (12 bit ADC) called "*diffDDR2seIDDR.vhd*", doing the differential to single ended and inverse DDR process bit by bit. As seen in the figure above, *adcDATAinterface* is the larger purple box and *diffDDR2seIDDR's* are the smaller brown boxes. The output is 4 12-bit signals. Although not pictured, the full speed data clocks from each channel's *dclk_dcm* are used to clock the IDDRs. Also not pictured are reset signals which are triggered at the beginning of each "**begin write**" command.

demux_fifo_sort.vhd

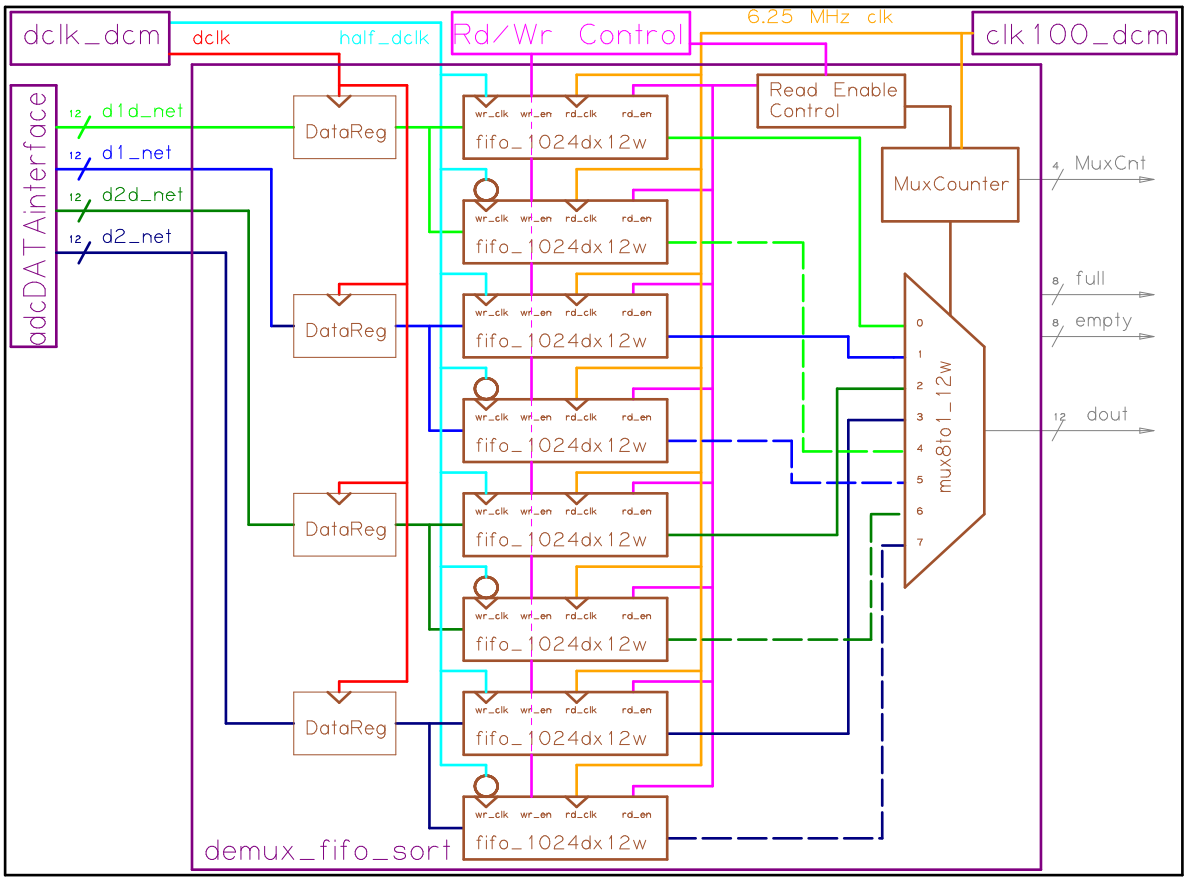


Figure 63 - Single Channel demux_fifo_sort.vhd Block Diagram

From the *adcDATAinterface* component, there are 4 data lines, 12-bits wide each, from each channel. This data must be saved to memory and sorted. As with the *adcDATAinterface,* an instantiation of *demux_fifo_sort* for each channel is required. First, the data is registered using *DataReg.vhd*, which merely holds onto the data for one clock cycle with 12 D flip-flops in parallel. The registers are always reading on every positive-edge of the write clock, despite the current read or write state.

When in the write state, the data is stored in FIFOs. Each of the 4 data lines split into 2 FIFOs, reducing the data clock speed by 2 ensuring accurate data storage. The FIFOs' write clock is the divide by 2 clock from the *dclk_dcm*. One FIFO writes on the rising edge while the other writes on the falling edge.

Once the FIFOs fill up, the read/write state machine is triggered, switches clock domains to the 100 MHz clock, and begins to read from the FIFOs. Note the order of output listed on the output mux. Therefore, whereas the write function is performed as a parallel process, the read function acts as a single serial output as needed by the demodulator. As per the ADC's data sheet, the order of data from the parallel input data lines is:

Order	Data Bus Name	Data Bus Description
1	d1d_net	Rising edge DDR; DEMUX value 1
2	d1_net	Rising edge DDR; DEMUX value 0
3	d2d_net	Falling edge DDR; DEMUX value 1

4	d2_net	Falling edge DDR; DEMUX value 0

Table 10 - Data Sort Order

The data out stream from this component feeds directly to the receiver/demodulator module. The read clock from the FIFOs is 6.25 MHz (100MHz/16). The purpose of the slow output speed will be discussed later on. Once the read state begins and the data is sent to the demodulator, it doesn't cease until the FIFOs are empty, at which point the read/write state machine will wait on the demodulation process to complete.

Each of the 8 FIFOs has a depth of 1024 and width of 12 bits. The 12 bit width is due to the 12 bit ADC. The 1024 depth, totaling 8192, guarantees at least one full transmitted data block is stored. The transmitter repeats a block of data 2^15 bits long per channel. Each symbol has 32 data points per channel for the main tests and simulation. Therefore, using a rounded off 21 GHz transmitter sampling rate and 1.6 GHz receiver sampling rate, the received points per symbol (N) is:

$$N = 32 * \frac{1.6e9}{21e9} \approx 2.44 \qquad \text{Equation 52}$$

At 2.44 bits per symbol, each data block will take the following amount of memory:

$$\frac{Mem}{Block} = 2^{15} * \frac{2.44}{32} = 2497 \qquad \text{Equation 53}$$

$$\frac{Total\ Mem}{Mem/Block} = \frac{8192}{2497} = 3.28 \qquad \text{Equation 54}$$

Therefore, each cycle stores 3.28 data blocks.

DataSort.vhd

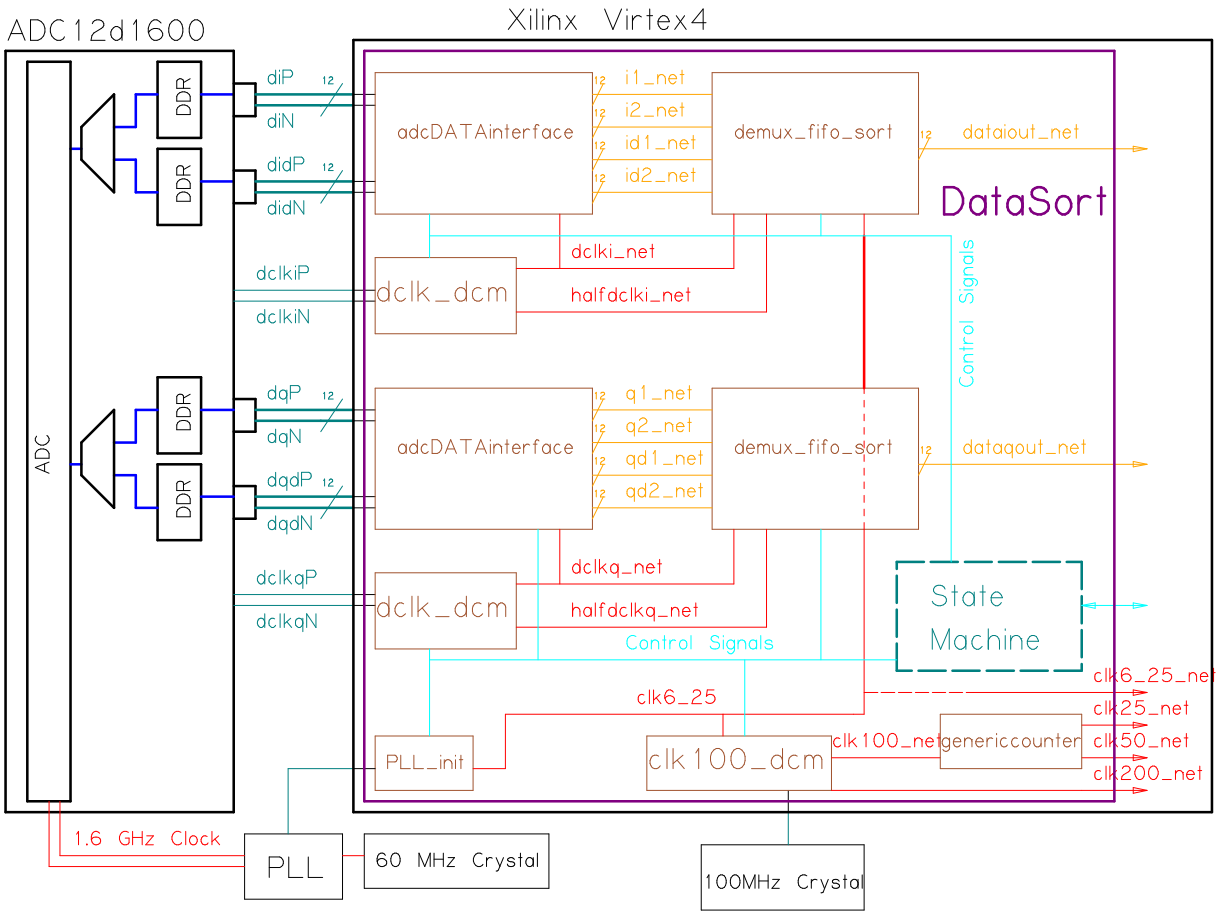


Figure 64 - Dual Channel DataSort.vhd Block Diagram

The DataSort.vhd file instantiates, connects, and runs the interface components. The two channels from the ADC are independent, but equivalent. Therefore, the same data chain exists for each channel, but with their respective inputs. As seen in Figure 64, the 12 bit, differential signals are connected from the ADC straight to the adcDATAinterface component, which performs differential to single ended conversion, followed by the inverse DDR process. From the interface component, the 4 12-bit signals continue to the

demux_fifo_sort component. The signals are registered, stored in FIFOs, read from the FIFOs, then sorted on the way out, ready to be processed by the DSP section of the top level design.

The ADC outputs a separate clock for each channel, thus a DCM component is required for each to run the IDDRs, clock the input registers, and write to the FIFOs. The read clock for the FIFOs, though, is the same 6.25 MHz clock from the 100 MHz DCM. A state machine contained in the DataSort component assists in this clock domain transfer, as well as controlling read, write, DSP, and reset operations.

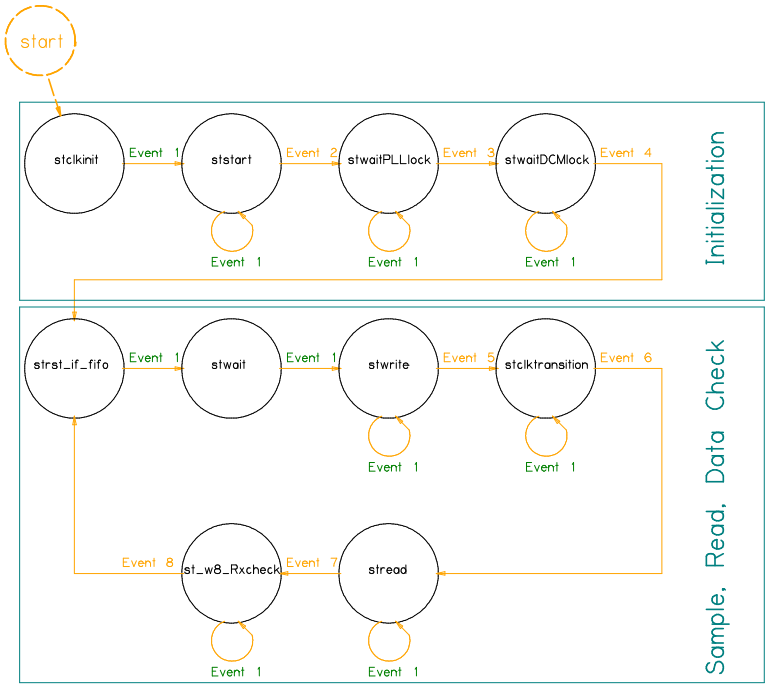


Figure 65 - DataSort.vhd State Machine

Event	Trigger
1	Rising edge of 6.25 MHz clock & 100 MHz DCM locked
2	Event 1 & PLL Initialization Complete
3	Event 1 & PLL Locked Signal Received (external)
4	Event 1 & ADC I and Q Clock DCMs Locked
5	Event 1 & All Data Sort FIFOs Full
6	Event 1 & All Data Sort FIFOs Not Empty
7	Event 1 & All Data Sort FIFOs Empty and Invalid
8	Event 1 & Receiver Check Done

Table 11 - DataSort.vhd State Machine Events

Figure 65 and Table 11 describe the state machine in DataSort.vhd. The initial state, *stclkinit*, checks for the 100 MHz DCM to lock onto the input 100 MHz crystal. A control signal *pllstart* also is triggered during the 1st state to begin the PLL initialization process. The state machine continues onto the 2nd state, *ststart,* after receiving a rising ede from the 6.25 MHz clock. *Ststart* waits for the PLL initialization process to complete. Once complete, *stwaitPLLlock* waits for a high signal from the external PLL to indicate the PLL has locked onto the 60 MHz crystal and the 1.6 GHz clock signal is being sent to the ADC. All this time, the two ADC clock DCM components have been held in a reset state to ready them for the clock signal from the ADC. This is done since the ADC DCMs rely on the clock from the ADC which relies on the clock from the PLL. Enabling the ADC DCMs prior to the external PLL being locked results in the DCMs intermittently being able to lock. Once the PLL signals a lock, the state machine enters *stwaitDCMlock*. In this state, the ADC DCM reset signal is cleared and the ADC DCMs are allowed to detect the incoming clock signals. Once both ADC DCM components signal they have locked onto their respective clock, the state machine moves onto *strst_if_fifo*. The initialization process (as noted in Figure 65) is now complete.

Strst_if_fifo marks the first state of the "Sample, Read, Data Check" loop. This loop repeats the main process of the program. As the name suggests, the interface components, FIFOs, and DSP data chain components are reset during *strst_if_fifo*. Note: the DSP data chain components do not exist under DataSort.vhd, but rather ReceiverFinal.vhd. After a single clock cycle, the state machine progresses to *stwait* to allow the reset signal to disable and the component to become ready. After another single clock cycle, the state machine progresses to *stwrite*. The FIFOs receive a write signal and continue to store data until they

are all full in *stwrite*. Once all the FIFOs have enabled "full" indicators, the state machine proceeds to *stclktransition*. This state's only role is to smoothly synchronize the transition from the FIFO write clock (derivative of the ADC clock) to the read clock (derivative of the 100 MHz clock). The next state is *stread*, where the FIFOs are read from until all the FIFOs have enabled "empty" indicators. Finally, the state machine enters into *st_w8_Rxcheck*. During this state the DataSort component has no role except to wait for the ReceiverFinal.vhd component to send a "check done" signal. This signal is sent if either the receiver has checked a data block for errors or has run through all the data without being able to find the start of a data block. Once that "check done" signal has been detected, the state machine repeats back to *strst_if_fifo* and begins the loop once again.

Control Signal	Function	State(s) When Enabled ('1')
pllstart	Triggers PLL_init.vhd to begin	*stclkinit*
pll_ce	Allows PLL registers to be altered	ALWAYS
dclk_rst	Resets ADC DCM components	*stclkinit, ststart, stwaitPLLlock*
rst_if_fifo	Resets the input buffers, FIFOs, and the DSP data chain (external component)	*strst_if_fifo*
we_net	Enables write to FIFO process	*stwrite*
rd_net	Enables read from FIFO process	*stread*
set_net	Sets all input buffers and registers to '1' (debugging purposes)	NEVER

Table 12 - DataSort.vhd Control Signals

Table 12 lists and describes the control signals which exist within DataSort.vhd and in which state(s) they are enabled. The signals **pllstart** and **dclk_rst** are active during the initialization process, then sit dormant. The signals **rst_if_fifo**, **we_net** (we=write enable) , and **rd_net** (rd=read) are used to control the DataSort FIFOs and are enabled during their respective states.

ReceiverFinal.vhd

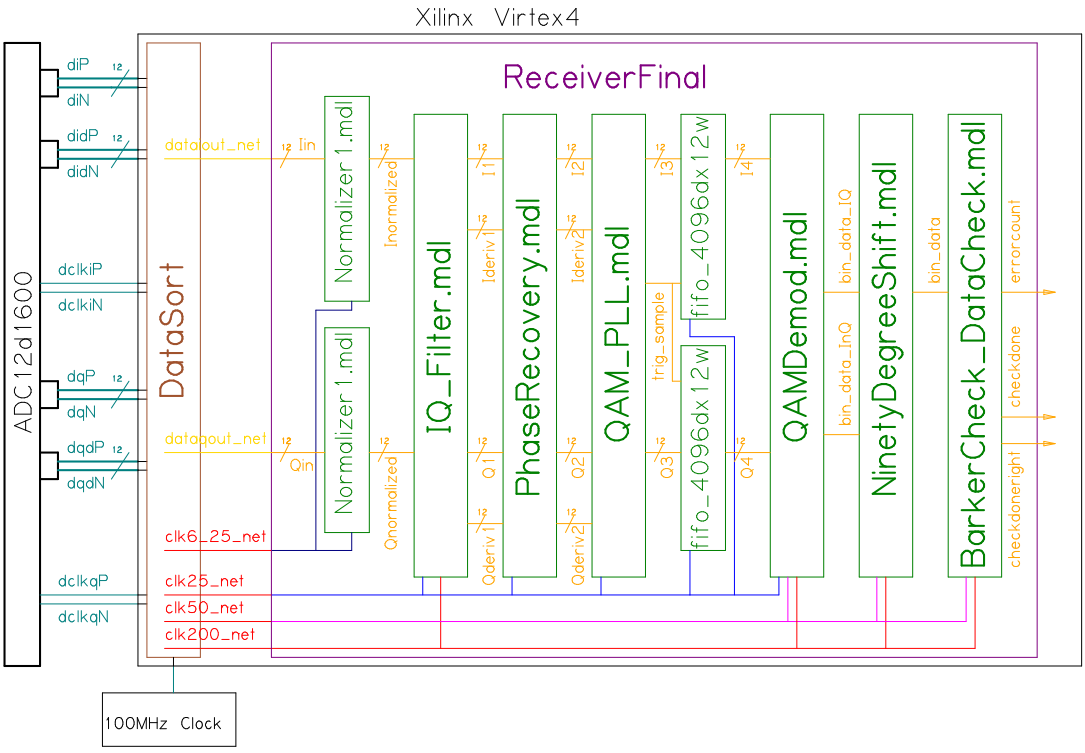


Figure 66 - ReceiverFinal.vhd Block Diagram

ReceiverFinal.vhd instantiates and connects all the Simulink generated components. The only non-Simulink components contained in this entity are two FIFOs separating the reclocking stage and the binary decoder. The clock signals to run these components originate from the DataSort.vhd component. The clock connections can be viewed in Figure 66, but were discussed in detail previously on in this paper.

The signal chain in this QAM, Nyquist-WDM receiver is very straight forward. The two data inputs are the I and Q channels: 12-bit data at 6.25 MHz data rate. The data is normalized (Normalizer1.mdl) to utilize the 12 bit width completely and output to

Inormalized and *Qnormalized*. A match and derivative filter (IQ_Filter.mdl) manipulate the data and output to *I1, Ideriv1, Q1, and Qderiv1*. Then, the phase recovery process (PhaseRecovery.mdl) is executed and results in *I2, Ideriv2, Q2,* and *Qderiv2.* The next block (QAM_PLL.mdl) reclocks the signal, resampling at (ideally) only the QAM symbol extremes. The outputs from the reclocking process are the I and Q symbol points, *I3* and *Q3*, and the trigger, *trig_sample*, to notify the next block when to sample the data.

Two FIFOs break up the data chain to store the data. Without the FIFOs between the clock recovery and the binary decoder section, the binary section of the process would have to be clocked using the trigger from the reclocking block. Instead, the FIFOs store a block of data and then empty their contents to the remaining components of the data chain. The FIFOs feed into a QAM demodulator (QAMDemod.mdl) block. Two binary data streams are created to be used by the next component (NinetyDegreeShift.mdl) to appropriately rotate the binary constellation to mimic the transmitted signal. Finally, the last component (BarkerCheck_DataCheck.mdl) detects the beginning of a data block and checks the following binary string for errors.

The three outputs from the final component make up the only three main (non-debugging) output signals: *errorcount, checkdone,* and *checkdoneright*. The signal *errorcount* indicates the number of errors in the current data check cycle. Internally in ReceiverFinal.vhd, there exists two signals which indicate the receiver has finished: *checkdoneright* and *checkdonewrong*. If either of these two signals are enabled, *checkdone* becomes enabled. *Checkdone* is used by the higher level entity and DataSort.vhd to trigger a reset for the whole system. The signal *checkdonewrong* is enabled when the

ReceiverFinal.vhd FIFOs empty their data storage without the final block detecting a start of data block. In other words, no error count could be conducted because the receiver could not detect the known binary sequence placed at the front of the data. Conversely, *checkdoneright* is enabled at the end of a received and triggered data block, signaling the receiver was able to detect the start of the data block and count the errors of the block.

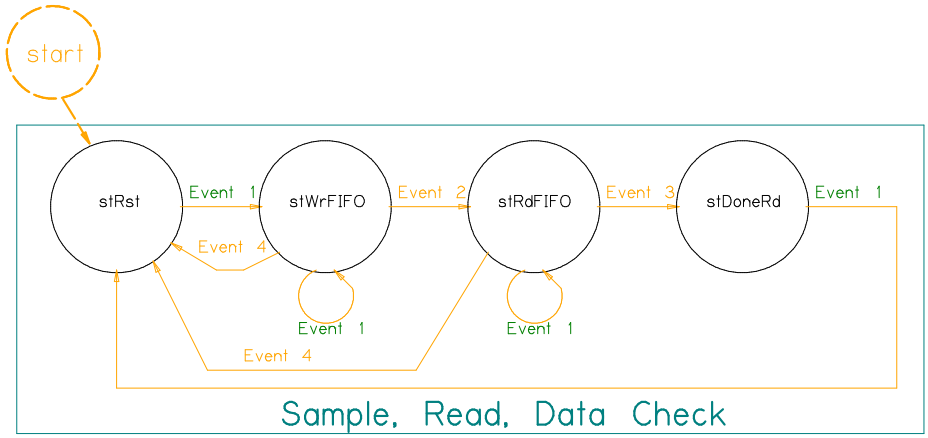


Figure 67 - ReceiverFinal.vhd State Machine

Event	Trigger
1	Rising edge of 6.25 MHz clock
2	Event 1 & Both FIFOs Full
3	Event 1 & Both FIFOs Empty
4	Event 1 & Reset Signal Received from DataSort.vhd

Table 13 - ReceiverFinal.vhd Block Diagram

Due to the existence of the FIFOs and clock shift in ReceiverFinal.vhd, a small state machine exists. This state machine can be thought of as existing under the rule of the DataSort state machine. Beginning at *stRst*, the state machine continues onto *stWrFIFO* to store the data from the reclocking component. Once the FIFOs are full, the state machine proceeds onto *stRdFIFO* to read the data from the FIFOs. If a reset signal is detected from DataSort during either the FIFO write or read states the state machine will return back to *stRst*. The state machine advances to *stDoneRd* if the FIFOs empty all their data. After a single clock cycle, the state machine returns to *stRst*.

Control Signal	Function	State(s) When Enabled ('1')
wr_en	Enables write to FIFO process	*stwrite*
rd_en	Enables read from FIFO process	*stread*
checkdonewrong	Signals that no error checking was completed	*stDoneRd*

Table 14 - ReceiverFinal.vhd Control Signals 1

Control Signal	Function	Enabled When
checkdoneright	Signals that error checking was completed	BarkerCheck_DataCheck.mdl signaled end of received desired data block
checkdone	Signals to the higher level entity that receiver process has completed	Either **checkdonewrong OR checkdoneright** enabled

Table 15 - ReceiverFinal.vhd Control Signals 2

Tables 14 and 15 list the internal control signals enable by specific states or other reasons, respectively. The **wr_en** and **rd_en** signals enable the FIFOs to be written to and read from, respectively. As explained previously, **checkdonewrong** indicates the FIFOs have emptied before an error counting process has completed and **checkdoneright** indicated a successful error counting process has completed. The role of **checkdone** is to notify the external blocks that the receiver has completed a cycle, whether or not an error count could be accumulated.

TopLevelDesign.vhd

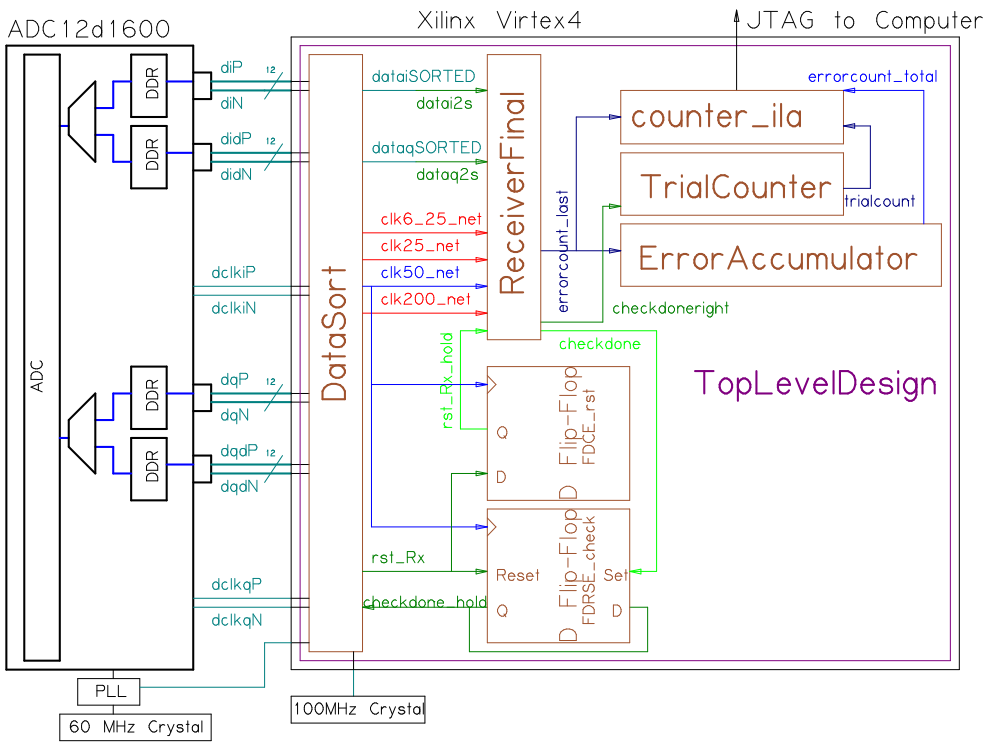


Figure 68 - TopLevelDesign.vhd Block Diagram

The overarching entity to combine all the elements of this project is TopLevelDesign.vhd. This entity connects the external hardware directly to the DataSort component and outputs an error count and trial count to the computer using an ILA core. The necessary connections between DataSort and ReceiverFinal, as discussed above, are completed in this designed. These signals are: *clk200_net, clk50_net, clk25_net, clk6_25_net,* and the I and Q channel data lines. The sorted data output is unsigned binary, but the input of the receiver requires 2's complement. To do this conversion simply and quickly, the most significant bit of each line is flipped, while the rest of the bits are left alone.

Two D-flip-flops included in this design ensure the reset and "check done" signals are timed and held correctly. *FDCE_rst* holds the reset signal an extra clock cycle before resetting the receiver component. This extra time ensures the accumulators receive their respective signals and accurately record the data. The second flip-flop, *FDRSE_check,* has a more complicated wiring scheme. The "set" input is controlled by the *checkdone_hold* signal. The reset input is controlled by the reset signal from the DataSort component. The output, Q, feeds back into the input, D, and also to DataSort.vhd to inform the component the receiver has completed its process. The following is a typical chain of events after to complete a data check cycle:

Order	Event	Set	Reset	D	Clk	Q
1	ReceiverFinal processing data; DataSort waiting for *checkdone_hold*	0	0	0	X	0
2	Receiver completes process	1	0	1	↑	1
3	DataSort detects *checkdone_hold*	0	0	1	X	1
4	DataSort advances its state machine to the reset state	0	1	0	↑	0
5	DataSort advances its state machine to the write FIFO state	0	0	0	X	0
6	Repeat to 1	0	0	0	X	0

Table 16 - FDRSE_reset Logic Sequence

The errors after each successful receiver cycle are accumulated in the *ErrorAccumulator* block. A signal named *errorcount_last* is always connected to the

accumulator, but only adds to the successive sum when a *checkdoneright* signal is enabled. Therefore, when the receiver empties its self-contained FIFOs without detecting the beginning of a data block, the *checkdone* signal triggers the reset process, but does not accumulate error data from that trial. But, when the receiver did detect the beginning of the data block, the accumulator adds the current *errorcount_last* term to the existing error count while triggering the reset process. A *checkdoneright* signal also causes the *TrialCounter* component to add 1 to its count. The user can access the error accumulation and the total of successful trials accounted for via the ILA core. With this information, a simple division can compute the BER.

$$BER = \frac{ErrorAccumulation}{TrialCount * BitsCheckedPerTrial} \qquad \text{Equation 55}$$

Hardware Results

The Nyquist-WDM signal could not successfully be demodulated by the DSP system on the FPGA. The signals were sampled, sorted, and filtered, but were unable to be correctly resampled and demodulated. The problem most likely occurred in one or two areas. The binary point shifts could have proved to confuse and disrupt the system. Also, it was noticed that the compiler dropped components due to inaccurately discerning they were unused, leaving a non-working, illogical system behind. The MUXs proved to be especially problematic, even when correctly initialized.

Future Work and Possible Improvements

The demo board's serial connection to the computer could produce a continuous BER calculation. The current system has upper limits as to how long it can run, although the upper limit is very high. This would also remove the Chipscope cores, which would free up a decent amount of resources which, of course, are scarce. This would open the door for BER vs Time implementation. Additional studies could be done by altering the environment of the optical system and noting the effect on the BER on a time plot.

The system would really benefit from a phase tracking and correction. This component would be complex, given the packet-based nature of this project. It would probably be near impossible to alter the current design to add this feature. A system with a non-burst, memory based front end would be necessary.

The most obvious end goal of this project is to compare OFDM and N-WDM resources with same taps. The study would take on an OFDM system with a 128 tap DFT and an N-WDM system with a 128 match filter. Due to the design of this project, the front end is interchangeable, so a simple swap out of the first component of the data demodulation chain between OFDM and N-WDM could be done. With that, a direct comparison of resource use and BER could be done.

Appendix

A. Digital PLL with Proportional plus Integrator loop filter derivation

$$V(z) = K_p F(z)\left(\theta(z) - \hat{\theta}(z)\right) = \frac{1}{K_0}\frac{1-z^{-1}}{z^{-1}}\hat{\theta}(z)$$

Rearranging,

$$\hat{\theta}(z)\left(\frac{1}{K_0}\frac{1-z^{-1}}{z^{-1}} + K_p F(z)\right) = K_p F(z)\theta(z)$$

$$H_d(z) = \frac{\hat{\theta}(z)}{\theta(z)} = \frac{K_p F(z)}{\left(\frac{1}{K_0}\frac{1-z^{-1}}{z^{-1}} + K_p F(z)\right)} = \frac{K_p K_0 F(z) z^{-1}}{1 - z^{-1} + K_p K_0 F(z) z^{-1}}$$

Substituting,

$$F(z) = K_1 + K_2\frac{1}{1-z^{-1}}$$

$$H_d(z) = \frac{K_p K_0\left[K_1 + K_2\frac{1}{1-z^{-1}}\right]z^{-1}}{1 - z^{-1} + K_p K_0\left[K_1 + K_2\frac{1}{1-z^{-1}}\right]z^{-1}} * \frac{1-z^{-1}}{1-z^{-1}}$$

$$H_d(z) = \frac{K_p K_0[K_1(1-z^{-1}) + K_2]z^{-1}}{1 - 2z^{-1} + z^{-2} + K_p K_0[K_1 1 - z^{-1} + K_2]z^{-1}}$$

Finally,

$$H_d(z) = \frac{K_p K_0(K_1 + K_2)z^{-1} - K_p K_0 K_1 z^{-2}}{1 - 2\left(1 - \frac{1}{2}K_p K_0(K_1 + K_2)\right)z^{-1} + (1 - K_p K_0 K_1)z^{-2}}$$

B. Bilinear Transformation

$$H_a(s) = \frac{2\zeta\omega_n s + {\omega_n}^2}{s^2 + 2\zeta\omega_n s + {\omega_n}^2}$$

$$H_d(z) = H_a(s)_{|s=\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} = \frac{2\zeta\omega_n\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right) + {\omega_n}^2}{\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + 2\zeta\omega_n\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right) + {\omega_n}^2}$$

Multiplying by $\frac{\frac{1}{{\omega_n}^2}}{\frac{1}{{\omega_n}^2}}$,

$$H_d(z) = \frac{2\zeta\frac{1}{\omega_n}\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right) + 1}{\left(\frac{1}{\omega_n}\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + 2\zeta\frac{1}{\omega_n}\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right) + 1}$$

Substituting $\theta_n = \frac{\omega_n T}{2}$,

$$H_d(z) = \frac{2\zeta\frac{1}{\theta_n}\frac{1-z^{-1}}{1+z^{-1}} + 1}{\left(\frac{1}{\theta_n}\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + 2\zeta\frac{1}{\theta_n}\frac{1-z^{-1}}{1+z^{-1}} + 1}$$

Multiplying by $\frac{{\theta_n}^2(1+z^{-1})^2}{{\theta_n}^2(1+z^{-1})^2}$,

$$H_d(z) = \frac{2\zeta\theta_n(1-z^{-1})(1+z^{-1}) + {\theta_n}^2(1+z^{-1})^2}{(1-z^{-1})^2 + 2\zeta\theta_n(1-z^{-1})(1+z^{-1}) + {\theta_n}^2(1+z^{-1})^2}$$

$$H_d(z) = \frac{\left(2\zeta\theta_n + {\theta_n}^2\right) + \left(2{\theta_n}^2\right)z^{-1} + \left(-2\zeta\theta_n + {\theta_n}^2\right)z^{-2}}{\left(1 + 2\zeta\theta_n + {\theta_n}^2\right) - 2\left(1 - {\theta_n}^2\right)z^{-1} + \left(1 - 2\zeta\theta_n + {\theta_n}^2\right)z^{-2}}$$

Denote $\psi = \left(1 + 2\zeta\theta_n + {\theta_n}^2\right)$ for organization,

$$H_d(z) = \frac{\frac{\left(2\zeta\theta_n + {\theta_n}^2\right)}{\psi} + \frac{\left(2{\theta_n}^2\right)}{\psi}z^{-1} + \frac{\left(-2\zeta\theta_n + {\theta_n}^2\right)}{\psi}z^{-2}}{1 - \frac{2\left(1 - {\theta_n}^2\right)}{\psi}z^{-1} + \frac{\left(1 - 2\zeta\theta_n + {\theta_n}^2\right)}{\psi}z^{-2}}$$

C. ReceiverSimulink.m

```
clear all
%load data and module filenames
filenames

%constants
Fs=21.4186e9*(1-2.8181e-005);
bitpersymbol=32; %each symbol = 2 bits, 4QAM
ADCFs=1.6e9;
interpfactor=4;
chanspace=1.05;
mfilterorder=100;
%%
%make 10 digital signal, resample, apply Nyquist filter, and
%combine
[tol,spe1,t,h,DataIn1]=TX_func([1 2 3 4 5 6 7 8 9 10...
11],chanspace,bitpersymbol,Fs);

%apply random/known phase change
phchange=2*rand()*pi;phchange=2*0.083*pi();%comment second for random
tol=tol.*exp(1j*phchange);

TOL=fftshift(fft(tol));
SPE1=fftshift(fft(spe1));

f1=linspace(-(Fs/2),(Fs/2),length(tol));
%%
%resampling Tx data rate (~21 GHz) to Rx data rate (1.6 GHz)
upsamprate=7*2;%344;
downsamprate=94*2;%4605;

%make analog signal
y=interp([tol],upsamprate);
fy=linspace(-(Fs/2)*upsamprate,(Fs/2)*upsamprate,length(y));
Y=fftshift(fft(y));

analogsig=[y];
ANALOGSIG=fftshift(fft(analogsig));

%load analog anti-aliasing filter response
importfile(filter_550); filter550=data; clear colheaders textdata data
%apply analog anti-aliasing filter
[analogsig,ANALOGSIG]=chebysfilter(ANALOGSIG,fy,filter550);
%%
%shift phase and sample

%resample to 1.6 GHz
insig=downsample(analogsig,downsamprate);

INSIG=fftshift(fft(insig));
```

```
f2=linspace(-
upsamprate*(Fs/2)/downsamprate,upsamprate*(Fs/2)/downsamprate,length(insig));

%repeat the input signal to mimic the real Tx, shift by random integer sample
insig_repeat=[insig insig insig insig insig insig insig
insig]/(max(insig)*1.2);
insig_repeat=circshift(insig_repeat,[0
round(rand()*bitpersymbol*upsamprate/downsamprate)]);
%%
%matched and derivative filting process

%retime the input signal to be 32 clocks per sample
tt=(0:32:32*length(insig_repeat)-1)';

%organize and normalize the I Q signals
reinsig=[tt real(insig_repeat)'];
reinsig(:,2)=reinsig(:,2)/max(abs(reinsig(:,2)))/2;%/2
iminsig=[tt imag(insig_repeat)'];
iminsig(:,2)=iminsig(:,2)/max(abs(iminsig(:,2)))/2;%/2

% simple enable signal to keep the filters running always
nd=[tt ones(length(tt),1)];

R=Fs*chanspace/(bitpersymbol);beta=.1;
Fpass=R/(ADCFs*interpfactor*chanspace);%match filter pass freq
Fstop=R*(2-(1-beta))/(ADCFs*interpfactor);%match filter stop freq
mf = matchfiltermaker(mfilterorder,Fpass,Fstop);%create match filter

Fpass=R/(ADCFs*interpfactor);%deriv filter pass freq
Fstop=R*(2-2*(1-beta)/1.9)/(ADCFs*interpfactor);%deriv filter stop freq
df=derivfiltermaker(Fpass,Fstop);%create deriv filter

%run the IQ filter Simulink module
sim(IQ_Filter_mdl);

%capture outputs
matchI=downsample(yout(:,2),32/interpfactor);
derivI=downsample(yout(:,3),32/interpfactor);
matchQ=downsample(yout(:,6),32/interpfactor);
derivQ=downsample(yout(:,7),32/interpfactor);

%calculate fft of the match filter output
MATCH=fftshift(fft(matchI));
f3=linspace(-upsamprate*(Fs/2)/downsamprate*interpfactor,upsamprate*(Fs/2)
... /downsamprate*interpfactor,length(MATCH));

%plot the -original analog signal (TOL),
%         -baseband analog signal (SPE1),
%         -signal after resampling and anti-alias filting (ANALOGSIG)
%         -signal sampled by the ADC (INSIG)
%         -inphase (I) signal after match filtering (MATCH)
```

```
figure(1)
plot(f1,mag2db(abs(TOL))-max(mag2db(abs(TOL)))...
    ,f1,mag2db(abs(SPE1))-max(mag2db(abs(SPE1)))...
    ,fy,mag2db(abs(ANALOGSIG))-max(mag2db(abs(ANALOGSIG))),...
    f2,mag2db(abs(INSIG))-max(mag2db(abs(INSIG)))...
    ,f3,mag2db(abs(MATCH))-max(mag2db(abs(MATCH))))
legend('tol','spe1','analogsig','insig','match');
xlabel('freq (GHz)'),ylabel('magnitude (dB)')
%%
%phase recovery

% keep same timing from IQ_Filter_mdl output
tt=0:1:length(matchI)-1;

%organize the I Q and derivative signals
Ifilt=[tt' matchI];Id1=[tt' derivI];
Qfilt=[tt' matchQ];Qd1=[tt' derivQ];

%apply reset signal at beginning of signal, then never again
rst=zeros(length(matchI),1);rst(1:2)=[1 1];
rst=[tt' rst];

%run Phase Recovery Simulink module
sim(Phase_Recovery)

%capture outputs
Iphasecorrected=yout(:,1); Id=yout(:,2);
Qphasecorrected=yout(:,3); Qd=yout(:,4);
%%
%clock recovery

tt=0:1:length(Iphasecorrected)-1;

%organize the I Q and derivative signals, normalize the deriv signals
maxId=max(abs(Id));
Iphasecorrected=[tt' Iphasecorrected];Id=[tt' Id/maxId];
maxQd=max(abs(Qd));
Qphasecorrected=[tt' Qphasecorrected];Qd=[tt' Qd/maxQd];

%define PLL gain/noise bandwidth factors and allocate memory
BnTs=0.002511886431510;
Kp=0.001584893192461;
trigIsums=zeros(length(BnTs),length(Kp));errorCntsums=trigIsums;
N=bitpersymbol/(Fs/ADCFs)*4;invN=1/N;zeta=1/sqrt(2);K0=-1;

%calculate PLL constants
A = BnTs/(zeta+1/(4*zeta));
B = zeta/N;
D = 1+2*B*A+(A/N)^2;
K1 = 2*B*A/(D*Kp*K0);
K2 = 2/(N^2)*A^2/(D*Kp*K0);
```

```
%run QAM PLL Simulink module
sim(QAM_PLL_mdl);

%QAM_PLL time plot
loopAdjust=yout(:,11);
x1=500:1000;x1l=length(x1);
x2=x1;x2l=length(x2);

figure(2)
plot(x1+14,yout(x1,1),x1+14,yout(x1,6),x1,yout(x1,2),x1,yout(x1,3)*.2,'o',...
    x1,yout(x1,4),x1,yout(x1,5),x1,yout(x1,9),x1,loopAdjust(x1),':')
legend('I match in','Q match in','deriv in','trig','trigged match',...
    'I to FB','Q to FB','Loop Adjust')
%%
%calculate PLL clocking shift vs time
cnt=1;temp=0;shift=0;

IoutT=yout(:,4);Iin=yout(:,1);
QoutT=yout(:,10);Qin=yout(:,6);

Iout(1)=IoutT(1);Qout(1)=QoutT(1);cnt1=2;

for ii=1:length(yout(:,1))
    if yout(ii,3)==1
        if temp==0
            temp=ii;
        else
            trigL(cnt,1)=ii-temp;
            if cnt > 1
                shift(cnt,1)=trigL(cnt,1)- ...
bitpersymbol*upsamprate/downsamprate*interpfactor+shift(cnt-1,1);
                shift(cnt,2)=ii;
            end
            cnt=cnt+1;
            temp=ii;
        end

        Iout(cnt1)=IoutT(ii);
        Qout(cnt1)=QoutT(ii);
        DataIndices(1,cnt1)=ii;
        cnt1=cnt1+1;
    end
end

figure(3)
plot(shift(:,2),shift(:,1))
legend('shift amount')
%%
%scatter plot of input, phase recovered, and clock recovered signals
figure(6)
scatter(matchI,matchQ)
```

```
hold on
scatter(Iphasecorrected(:,2),Qphasecorrected(:,2))

scatter(Iout*1.5,Qout*1.5)
hold off
%%
%QAM demod

%organize data and time arrays for QAM demod
tt=0:2:2*length(Iout)-1;
I1i=[tt' Iout'];
Q1i=[tt' Qout'];
%plot input to QAM Demod
st=2;ed=1000;
xx=linspace(st,ed,ed-st+1);
figure(5)
plot(xx,Iout(xx),'o',xx,Qout(xx),'o')

%run QAM Demod Simulink model
sim(QAMDemod_mdl); %clk = clk/2
%%
%180 degree phase shift checker/executer

%reclock 4x oversampling
tt=0:4:4*length(yout)-1;
dataIQ=[tt' yout(:,1)];
dataInegQ=[tt' yout(:,2)];

%reset first 2 samples, none after
rst=[tt' [ones(2,1);zeros(length(tout)-2,1)]];

%create 50 MHz (non-oversampled data rate) signal
clk_50MHz=repmat([0 0 0 0 1 1 1 1],1,ceil((length(yout))/8*4));
t2=0:length(clk_50MHz)-1;
clk_50MHz=[t2' clk_50MHz'];

%run 180 degree shift correcter Simulink module
sim(NinetyDegreeShift_mdl)

%capture outputs and output text of any shift events
shift_pt3=yout(:,4);
shift_pt3i=find(shift_pt3==1);
fprintf('shift 180 pt3 %i\r',shift_pt3i)
shift_pt4=yout(:,3);
shift_pt4i=find(shift_pt4==1);
fprintf('shift 180 pt4 %i\r',shift_pt4i)
shift_pt7=yout(:,2);
shift_pt7i=find(shift_pt7==1);
fprintf('shift 180 pt7 %i\r',shift_pt7i)
```

```
%capture main and other outputs
tt=0:1:1*length(yout)-1;
data=[tt' yout(:,1)];
filt180in_1=yout(:,5);filt180out_1=yout(:,8);
filt180in_2=yout(:,6);filt180out_2=yout(:,9);
filt180in_3=yout(:,7);filt180out_3=yout(:,10);
%%
%BER checker

%run BER checker
load(DataIn1_2048)
sim(BarkerDataChecker_mdl); %clk = clk

%capture several intermediate outputs for debugging
barkerfilt=yout(:,1);
trigI=yout(:,2);
trigQ=yout(:,3);

%check/display beginning of data segments
wait=1;

for ii=1:length(yout(:,1))
    if wait==1
        if yout(ii,2) ==1
            fprintf('trigI %i\r',ii)
            wait=0;
        end
    end
    if wait ==1
        if yout(ii,3) ==1
            fprintf('trigQ %i\r',ii)
            wait=0;
        end
    end
    if wait ==0
        if ((yout(ii,2) == 0) && (yout(ii,3) ==0))
            wait =1;
        end
    end
end

%capture and display error count
errorCnt=downsample(yout(:,5),4);
done=downsample(yout(:,8),4);
percentDone=downsample(yout(:,9)/2176,4);

doneI=find(done==1);

fprintf('error count %i\r',errorCnt(doneI))
```

D. ReceiverPP.mat

```
clear all
filenames
di=load('D:\\Work\\Masters
Project\\Matlab\\ReceiverFinal\\Data\\C2i100000_wf1.dat');
dq=load('D:\\Work\\Masters
Project\\Matlab\\ReceiverFinal\\Data\\C3q100000_wf1.dat');

bitpersymbol=32;
Fs=21.4186e9*(1-2.8181e-005);
ADCFs=1.6e9;chanspace=1.3;interpfactor=4;
mfilterorder=100;


di=interp(di,2); di=downsample(di,25);
dq=interp(dq,2); dq=downsample(dq,25);

DI=fftshift(fft(di));
DQ=fftshift(fft(dq));
f1=linspace(-800e6,800e6,length(DI));


%%
tt=0:length(di)-1;
rst=[tt' zeros(length(di),1)];

in=[tt' di];
sim(Normalizer1_mdl)
di_normalized=yout(:,1);

in=[tt' dq];
sim(Normalizer1_mdl)
dq_normalized=yout(:,1);

%%
tt=(0:32:32*length(dq_normalized)-1)';
reinsig=[tt di_normalized];
iminsig=[tt dq_normalized];
% iminsig(:,2)=iminsig(:,2)/max(abs(iminsig(:,2)))/2;

nd=[tt ones(length(tt),1)];

R=Fs*chanspace/(bitpersymbol);beta=.1;
Fpass=R/(ADCFs*interpfactor*chanspace);%match filter pass freq
Fstop=R*(2-(1-beta))/(ADCFs*interpfactor);%match filter stop freq
mf = matchfiltermaker(mfilterorder,Fpass,Fstop);%create match filter

Fpass=R/(ADCFs*interpfactor);%deriv filter pass freq
Fstop=R*(2-2*(1-beta)/1.9)/(ADCFs*interpfactor);%deriv filter stop freq
```

```
df=derivfiltermaker(Fpass,Fstop);%create deriv filter

sim(IQ_Filter_mdl);
Imatch=yout(:,2);
matchI=downsample(yout(:,2),8); derivI=downsample(yout(:,3),8);
matchQ=downsample(yout(:,6),8); derivQ=downsample(yout(:,7),8);

MATCH=fftshift(fft(matchI));
f2=linspace(-800e6*4,800e6*4,length(MATCH));

figure(1)
plot(f1,mag2db(abs(DI))-max(mag2db(abs(DI)))...
    ,f2,mag2db(abs(MATCH))-max(mag2db(abs(MATCH))),':')
legend('Input','Filter Stage Output')


%%
%phase recovery

% keep same timing from IQ_Filter_mdl output
tt=0:1:length(matchI)-1;

%organize the I Q and derivative signals
Ifilt=[tt' matchI];Id1=[tt' derivI];
Qfilt=[tt' matchQ];Qd1=[tt' derivQ];

%apply reset signal at beginning of signal, then never again
rst=zeros(length(matchI),1);rst(1:2)=[1 1];
rst=[tt' rst];

%run Phase Recovery Simulink module
sim(Phase_Recovery)

%capture outputs
Iphasecorrected=yout(:,1); Id=yout(:,2);
Qphasecorrected=yout(:,3); Qd=yout(:,4);

%%
%clock recovery

tt=0:1:length(Iphasecorrected)-1;

%organize the I Q and derivative signals, normalize the deriv signals
maxId=1;%max(abs(Id));
Iphasecorrected=[tt' Iphasecorrected];Id=[tt' Id/maxId];
maxQd=1;%max(abs(Qd));
Qphasecorrected=[tt' Qphasecorrected];Qd=[tt' Qd/maxQd];

clk6_25=repmat([0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1],1,ceil((length(Iphasecorrected))/16));
t2=0:length(Iphasecorrected)+2;
```

```
clk6_25=[t2' clk6_25'];

loop_fb_sel=[ones(25,1);zeros(length(Iphasecorrected)-25,1)];
loop_fb_sel=[tt' loop_fb_sel];

BnTs=logspace(-3,-1,6);
% Kp=linspace(.05,.8,8);
Kp=logspace(-4,-1,6);
trigIsums=zeros(length(BnTs),length(Kp));errorCntsums=trigIsums;
N=bitpersymbol/(Fs/ADCFs)*4;invN=1/N;zeta=1/sqrt(2);K0=-1;

% for BnTsii=1:length(BnTs)
        for BnTsii=2
%
%         for Kpii=4
    for Kpii=3
    [BnTsii Kpii]


        A = BnTs(BnTsii)/(zeta+1/(4*zeta));
        B = zeta/N;
        D = 1+2*B*A+(A/N)^2;
        K1 = 2*B*A/(D*Kp(Kpii)*K0);
        K2 = 2/(N^2)*A^2/(D*Kp(Kpii)*K0);
        %run QAM PLL Simulink module
        sim(QAM_PLL_mdl);

        loopAdjust=yout(:,11);

        %QAM_PLL time plot
        x1=500:1000;x1l=length(x1);
        x2=x1;x2l=length(x2);

        figure(2)

plot(x1+14,yout(x1,1),x1+14,yout(x1,6),x1,yout(x1,2),x1,yout(x1,3)*.2,'o',...
            x1,yout(x1,4),x1,yout(x1,5),x1,yout(x1,9),x1,loopAdjust(x1),':')
        legend('I match in','Q match in','deriv in','trig','trigged
match',...
            'I to FB','Q to FB','Loop Adjust')

        %%
        %calculate PLL clocking shift vs time
        cnt=1;temp=0;shift=0;

        IoutT=yout(:,4);Iin=yout(:,1);
        QoutT=yout(:,10);Qin=yout(:,6);

        Iout(1)=IoutT(1);Qout(1)=QoutT(1);cnt1=2;

        for ii=1:length(yout(:,1))
```

```
            if yout(ii,3)==1
                if temp==0
                    temp=ii;
                else
                    trigL(cnt,1)=ii-temp;
                    if cnt > 1
                        shift(cnt,1)=trigL(cnt,1)-N+shift(cnt-1,1);
% shift(cnt,1)=trigL(cnt,1)-10+shift(cnt-1,1);
                        shift(cnt,2)=ii;
                    end
                    cnt=cnt+1;
                    temp=ii;
                end

                Iout(cnt1)=IoutT(ii);
                Qout(cnt1)=QoutT(ii);
                DataIndices(1,cnt1)=ii;
                cnt1=cnt1+1;
            end
        end

        figure(3)
        plot(shift(:,2),shift(:,1))
        legend('shift amount')

        %%
        %scatter plot of input, phase recovered, and clock recovered signals
                figure(6)
                scatter(matchI,matchQ)

                hold on
                scatter(Iphasecorrected(:,2),Qphasecorrected(:,2))

                scatter(Iout*1.5,Qout*1.5)
                hold off


        %%
        %QAM demod

        %organize data and time arrays for QAM demod
        tt=0:2:2*length(Iout)-1;
        I1i=[tt' Iout'];
        Q1i=[tt' Qout'];
        %plot input to QAM Demod
        st=2;ed=1000;
        x=linspace(st/2,ed/2,(ed-st)/2+1);x1=st:2:ed;
        xx=linspace(st,ed,ed-st+1);
        x=1:length(Iout);
        figure(5)
        % plot(x1,I1i(x,2),':o',x1,Q1i(x,2),'-o')
        plot(x,Iout,'o',x,Qout,'o')
```

```
%%
%run QAM Demod Simulink model
sim(QAMDemod_mdl); %clk = clk/2


%%
%180 degree phase shift checker/executer

%reclock 4x oversampling
tt=0:4:4*length(yout)-1;
dataIQ=[tt' yout(:,1)];
dataInegQ=[tt' yout(:,2)];

%reset first 2 samples, none after
rst=[tt' [ones(2,1);zeros(length(tout)-2,1)]];

%create 50 MHz (non-oversampled data rate) signal
clk_50MHz=repmat([0 0 0 0 1 1 1 1],1,ceil((length(yout))/8*4));
t2=0:length(clk_50MHz)-1;
clk_50MHz=[t2' clk_50MHz'];

%run 180 degree shift correcter Simulink module
sim(NinetyDegreeShift_mdl)

%capture outputs and output text of any shift events
shift_pt3=yout(:,4);
shift_pt3i=find(shift_pt3==1);
fprintf('shift 180 pt3 %i\r',shift_pt3i)
shift_pt4=yout(:,3);
shift_pt4i=find(shift_pt4==1);
fprintf('shift 180 pt4 %i\r',shift_pt4i)
shift_pt7=yout(:,2);
shift_pt7i=find(shift_pt7==1);
fprintf('shift 180 pt7 %i\r',shift_pt7i)

%capture main and other outputs
tt=0:1:1*length(yout)-1;
data=[tt' yout(:,1)];
filt180in_1=yout(:,5);filt180out_1=yout(:,8);
filt180in_2=yout(:,6);filt180out_2=yout(:,9);
filt180in_3=yout(:,7);filt180out_3=yout(:,10);

%%
%BER checker

%run BER checker
load(DataIn1_2048_50err)
sim(BarkerDataChecker_mdl); %clk = clk

%capture several intermediate outputs for debugging
```

```
        barkerfilt=yout(:,1);
        trigI=yout(:,2);
        trigQ=yout(:,3);

        %check/display beginning of data segments
        wait=1;

        for ii=1:length(yout(:,1))
            if wait==1
                if yout(ii,2) ==1
                    fprintf('trigI %i\r',ii)
                    wait=0;
                end
            end
            if wait ==1
                if yout(ii,3) ==1
                    fprintf('trigQ %i\r',ii)
                    wait=0;
                end
            end
            if wait ==0
                if ((yout(ii,2) == 0) && (yout(ii,3) ==0))
                    wait =1;
                end
            end
        end

        %capture and display error count
        errorCnt=downsample(yout(:,5),4);
        done=downsample(yout(:,8),4);
        percentDone=downsample(yout(:,9)/2176,4);

        doneI=find(done==1);

        fprintf('error count %i\r',errorCnt(doneI))

        trigIsums(BnTsii,Kpii)=sum(trigI);
        errorCntsums(BnTsii,Kpii)=sum(errorCnt(doneI));

        if length(doneI)>4
            if doneI(2)==50
        figure(7)
        subplot(length(BnTs),length(Kp),(BnTsii-1)*length(Kp)+Kpii)
        scatter(Iout,Qout)
            end
        end
    end
end
```

E. BER Data

BER vs Match Filter Order

Tx
sm/sy
32
FstpF
2
dataL
16384
V&V
16
R factor
1
InPhase
0.083

	Filt N							
	25	50	80	100	150	250	500	1000
	0.039597	0.024038	0.005874	0.001669	0.000771	0.000584	0.000546	0.00038

Data	Filt N							
	25	50	80	100	150	250	500	1000
	0.036326	0.023475	0.005628	0.001747	1.30E-03	0.00013	0.00039	0.000917
	0.034387	0.026256	0.008992	0.001747	6.48E-04	0.000648	6.51E-05	0
	0.051128	0.018496	0.005757	0.001165	7.12E-04	0.000584	0.001171	9.82E-04
	0.033676	0.025998	0.004722	0.001941	5.18E-04	0.000973	0.001431	0.000131
	0.051968	0.023928	0.006598	0.001941	0.000583	0.001621	0.001041	0.000196
	0.046927	0.023411	0.00414	0.002071	0.001036	0.000454	0	0.000262
	0.034581	0.027032	0.005628	0.001747	0	6.48E-05	0.000455	0.000524
	0.036908	0.024639	0.004269	0.000971	0.000842	0.000973	0	0
	0.03555	0.027679	0.006986	0.002459	0.001101	0.000259	0.00039	0.000393
	0.034516	0.019466	0.006016	0.000906	0.000971	0.00013	0.00052	0.000393

BER vs Match Filter Order vs Channel Spacing

Tx sm/sy
32
FstpF
2
dataL
16384
V&V
8
FiltN
100
InPhase
9.71E-02

ChanS

	FiltN 25	60	80	100	150	250	500
100%	0.24213	0.156654	0.136986	0.087695	0.061862	0.058376	0.057793
105%	0.084394	0.068227	0.049744	0.038801	0.03078	0.030107	0.0337
110%	0.041583	0.025502	0.015956	0.01486	0.011548	0.013422	0.010713
120%	0.003835	0.001423	0.001251	0.000927	0.001792	0.001535	0.001908
130%	0.00028	0.000129	2.16E-05	0	0	2.16E-05	0.00013
140%	0	0	0	0	0	0	0

Data

V&V

ChSp	0	1	2	4	8	16	24
1.00	0.133346	0.147901	0.111521	0.091427	5.69E-02	0.055437	0.055754
1.05	0.084739	0.059368	0.051814	0.039146	3.28E-02	0.030993	0.033635
1.10	0.044535	0.024963	0.017918	0.012811	8.87E-03	0.012514	0.01119
1.20	2.65E-03	0.001164	0.000517	0.001488	2.46E-03	0.002399	0.001822
1.30	6.46E-05	0	6.47E-05	0	0	6.48E-05	0
1.40	0	0	0	0	0	0	0

V&V

ChSp	0	1	2	4	8	16	24
1.00	0.294939	0.160189	0.166505	0.086962	5.99E-02	0.059392	0.059463
1.05	0.084481	0.071784	0.051038	0.039405	2.83E-02	0.028983	0.029796
1.10	0.036003	0.030783	0.016172	0.014882	1.38E-02	0.014718	0.011906
1.20	0.003749	0.00194	0.001811	0.000388	0.000907	4.54E-04	0.002017
1.30	0.000776	1.29E-04	0.00E+00	0	0	0	0.00013
1.40	0	0	0	0	0	0.00E+00	0

V&V

ChSp	0	1	2	4	8	16	24
1.00	0.298106	0.16187	0.132932	0.084698	6.88E-02	0.0603	0.058161
1.05	0.083964	0.07353	0.046381	0.037852	3.12E-02	0.030344	0.037668
1.10	0.044212	0.020759	0.013778	0.016888	1.20E-02	0.013032	0.009043
1.20	0.005106	0.001164	0.001423	0.000906	0.002007	0.001751	0.001887
1.30	0	0.000259	0	0	0	0	0.00026
1.40	0.00E+00	0	0	0	0	0	0

BER vs Percent Analog Fitler Bandwidth vs V&V Ratio

Tx
sm/sy
32
filter N
150
FstpF
2
dataL
16384
V&V
8
R factor
1
InPhase
0.083

			V&V							
			0	1	2	4	8	16	24	32
	12	159%	0.233885	0.230096	0.344094	0.319844	0.163841	0.364274	0.125171	0.321706
	16	119%	0.056885	0.239498	0.116797	0.185228	0.03721	0.009881	0.00294	0.000281
	20	95%	0.010046	0.111176	0.042128	0.021799	0.007778	0.000756	0	0
	24	80%	0.014448	0.105993	0.045265	0.026131	0.008682	0.001037	4.32E-05	4.32E-05
	28	68%	0.010599	0.109384	0.050083	0.025452	0.00762	0.000756	0	0
	32	60%	0.012886	0.097498	0.056768	0.030111	0.006497	0.000496	0.00013	0
	36	53%	0.019656	0.105508	0.061234	0.034846	0.010011	0.000734	0.000151	6.47E-05
	40	48%	0.027007	0.109882	0.054509	0.032464	0.012964	0.000884	6.47E-05	0
	50	38%	0.058277	0.103117	0.061232	0.040181	0.017017	0.002049	0.000345	0.000367
	60	32%	0.089494	0.120755	0.077615	0.0575	0.024578	0.009939	0.006339	0.006662

Data

N Tx

	V&V							
	0	1	2	4	8	16	24	32
12	0.229035	0.225528	0.378954	0.165963	1.68E-01	0.419617	0.127834	0.423774
16	0.058961	0.379062	0.113252	0.078615	3.74E-02	0.010443	0.002854	0
20	0.009398	0.10221	0.04161	0.016722	8.04E-03	0.000907	0	0.00E+00
24	0.014383	0.105993	0.044639	0.026174	9.65E-03	0.002203	0	0.00013
28	0.009973	0.110096	0.048961	0.025128	0.007836	0.000583	0	0
32	0.011526	0.097002	0.057308	0.028362	0.008224	0.001036	0.000389	0
36	0.022785	0.107062	0.066024	0.035277	0.009645	0.000647	0	0
40	0.022779	0.10341	0.057206	0.032874	0.012036	0.000777	0	0
50	0.056551	0.105209	0.063863	0.03934	0.016758	0.001229	0.000453	0.000324
60	0.091391	0.116875	0.077033	0.061768	0.022638	0.011125	0.006533	0.006533

N Tx

	V&V							
	0	1	2	4	8	16	24	32
12	0.238909	0.22897	0.377785	0.397012	1.49E-01	0.248782	0.130692	0.113024
16	0.054031	0.163196	0.114679	0.078615	3.55E-02	0.009405	0.002919	0
20	0.009398	0.117506	0.043165	0.023009	8.94E-03	0.000648	0	0
24	0.013541	0.103013	0.042306	0.025008	0.008682	1.94E-04	0.00013	0
28	0.009973	0.110874	0.049867	0.024221	0.008354	0.001554	0	0
32	0.013598	0.09752	0.057437	0.031406	0.005698	6.48E-05	0	0
36	0.019483	0.107256	0.058321	0.03217	0.010551	8.41E-04	0.000324	0
40	0.036562	0.112664	0.055717	0.031774	0.012942	0.001683	6.47E-05	0
50	0.062828	0.104885	0.058428	0.042834	0.018117	0.00317	0.000324	0.000324
60	0.088093	0.121984	0.077421	0.056659	0.022896	0.009249	0.006144	0.006144

N Tx

	V&V							
	0	1	2	4	8	16	24	32
12	0.233712	0.235791	0.275544	0.396557	1.74E-01	0.424424	0.116986	0.428321
16	0.057664	0.176234	0.122462	0.398456	3.88E-02	0.009794	0.003049	0.000843
20	0.011342	0.113812	0.04161	0.025666	6.35E-03	0.000713	0	0
24	0.01542	0.108973	0.04885	0.027211	0.00771	0.000713	0	0
28	0.011852	0.107182	0.051422	0.027006	0.006671	0.00013	0	0
32	0.013534	0.097973	0.055559	0.030564	0.005569	0.000389	0	0
36	0.0167	0.102207	0.059357	0.03709	0.009839	0.000712	0.000129	0.000194
40	0.021679	0.11357	0.050605	0.032744	0.013913	0.000194	0.000129	0.00E+00
50	0.055451	0.099256	0.061404	0.038369	0.016176	0.001747	0.000259	4.53E-04
60	0.088998	0.123407	0.078391	0.054072	0.0282	0.009443	0.006339	0.007309

BER vs Input Phase Angle vs Viterbi-Viterbi Ratio

Tx
sm/sy
32
filter
N
150
FstpF
2
dataL
16384
ChS
1.05
R
factor
1

		V&V							
		0	1	2	4	8	16	24	32
	0.000	0	0.015865	0.003734	0.001813	0.00041	0	0	0
	0.014	0	0.019793	0.005483	0.002633	0.000324	0	0	0
	0.028	0	0.025276	0.008289	0.003432	0.000453	0	0	0
	0.042	0	0.033802	0.013836	0.004921	0.000928	0	0	0
In Phase	0.056	0.000302	0.05189	0.021067	0.009001	0.001684	0	0	0
	0.069	0.002007	0.066654	0.032399	0.013534	0.003885	6.48E-05	0	0
	0.083	0.014678	0.099722	0.057804	0.02778	0.009605	0.000971	8.63E-05	0
	0.097	0.061365	0.15364	0.099204	0.067755	0.030629	0.005375	0.001209	0.000281
	0.111	0.178312	0.239699	0.200458	0.164023	0.127307	0.067237	0.032032	0.026657
	0.1249	0.375726	0.376783	0.375639	0.374107	0.367934	0.377625	0.360552	0.389799

Data

				V&V				
	0	1	2	4	8	16	24	32
0	0	0.013857	0.003302	0.001554	5.18E-04	0	0	0
0.013889	0	0.019426	0.005504	0.002461	6.48E-05	0	0	0.00E+00
0.027778	0	0.023765	0.007058	0.003367	4.53E-04	0	0	0
0.041667	0	0.03283	0.010814	0.003108	0.000583	0	0	0
0.055556	0.000194	0.05148	0.018131	0.007835	0.001813	0	0	0
0.069444	0.001619	0.069028	0.032442	0.015023	0.005116	0.000194	0	0
0.083333	0.01457	0.100434	0.054653	0.026161	0.006734	0.000324	0	0
0.097222	0.062358	0.151719	0.102312	0.069481	0.025837	0.008353	0.000648	0
0.111111	0.177686	0.235317	0.200479	0.163958	0.121673	0.062423	0.029593	0.016901
0.1249	0.376935	0.367804	0.371107	0.380431	0.357573	0.350774	0.360681	0.38613

InPhase

				V&V				
	0	1	2	4	8	16	24	32
0	0	0.017225	0.004339	0.002266	1.30E-04	0	0	0
0.013889	0	0.019621	0.005569	0.002914	5.83E-04	0	0	0
0.027778	0	0.02493	0.010166	0.003561	0.000324	0.00E+00	0	0
0.041667	0	0.035874	0.0158	0.006475	0.000648	0	0	0
0.055556	0.000453	0.052451	0.021498	0.008289	0.000971	0	0	0
0.069444	0.002202	0.064366	0.030758	0.01457	0.003756	0.00E+00	0	0
0.083333	0.015023	0.100758	0.058085	0.030888	0.01049	0.001425	0	0
0.097222	0.061776	0.152496	0.097002	0.067021	0.03283	0.00531	0.000777	0.000842
0.111111	0.174059	0.246196	0.196723	0.162339	0.132163	0.071035	0.03173	0.038529
0.1249	0.374021	0.380302	0.378942	0.367286	0.363077	0.40439	0.337693	0.410283

InPhase

				V&V				
	0	1	2	4	8	16	24	32
0	0	0.016512	0.003561	0.001619	5.83E-04	0	0	0
0.013889	0	0.020333	0.005375	0.002525	3.24E-04	0	0	0
0.027778	0	0.027132	0.007641	0.003367	0.000583	0	0	0
0.041667	0	0.032701	0.014893	0.00518	0.001554	0	0	0
0.055556	0.000259	0.051739	0.023571	0.010879	0.002266	0	0	0
0.069444	0.002202	0.066567	0.033996	0.011008	0.002784	0	0	0
0.083333	0.01444	0.097973	0.060675	0.02629	0.011591	0.001166	0.000259	0.00E+00
0.097222	0.059962	0.156705	0.098297	0.066762	0.033219	0.002461	0.002202	0.00E+00
0.111111	0.18319	0.237583	0.20417	0.165771	0.128084	0.068251	0.034773	0.024542
0.1249	0.376222	0.382244	0.37687	0.374603	0.383151	0.377712	0.38328	0.372985

InPhase

Works Cited

Baker, B. C. (1999). *Anti-Aliasing, Analog Filters for Data Acquisition Systems.* Microchip Technology Inc.

Gabriella Bosco, e. a. (2010, August 1). Performance Limits of Nyquist-WDM and. *IEEE PHOTONICS TECHNOLOGY LETTERS, VOL. 22, NO. 15*, pp. 1129-1131.

Haykin, S. (2002). *Adaptive Filter Theory.* Upper Saddle River: Prentice Hall.

Hui, R., & O'Sullivan, M. (2009). *Fiber Optic Measurement Techniques.* Burlington: Elsevier Academic Press.

Ifeachor, E., & Jervis, B. (2001). *Digital SIgnal Processing: A Practical Approach.* Prentice Hall.

Junyi Wang, C. X. (2012). Generation of Spectrally Efficient Nyquist-WDM QPSK Signals using DSP Techniques at Transmitter . *OFC/NFOEC Technical Digest*.

Kikuchi, K. (2010). Coherent optical communications: Historical perspectives and future directions. *Optical and Fiber Communications Reports 6*. Springer.

Kikuchi, K. (2011, Oct 25). Digital coherent optical communication systems: fundamentals and future prospects. *IEICE Electronics Express*, pp. 1642-1662.

Liu, M. M.-K. (1996). *Principles and Applications of Optical Communications.* Irwin.

National Semiconductor Corporation. (2011, October 11). ADC12D1000/ADC12D1600.

Q. Yang, A. A. (2011). Optical OFDM Basics. In S. Kumar, *Impact of Nonlinearities on Fiber Optic Communications* (pp. 43-87). New York: Springer.

Rice, M. (2009). *Digital Communications: A Discrete Time Approach.* Upper Saddle River: Pearson Prentice Hall.

S. Shimotsu, S. O. (2001, April). Single Side-Band Modulation Performence... Letter 3. *IEEE Photonics Technology Letters, VOL. 13, NO. 4*, p. 364.

Texas Instruments. (2013, February). LMX2531 High Performance Frequency Synthesizer System with Integrated VCO Data Sheet. Dallas, TX.