Robert Taggart
rtaggart@cc.usu.edu
Work: (801) 621-3535

Brandon Wilson
blwilson@cc.usu.edu
Work: 787-0100

Brent Haslem
brenthas@cc.usu.edu
Work: 755-9848

Monday Apr. 28th, 2003.

Paul Israelsen
Department of Electrical Engineering
Utah State University

Dr. Israelsen,

The following document contains Our Final Design Report. We have truly had an educational experience in a variety of ways all of which will all be discussed in this paper. We have been limited in our contact with L3 since the middle of the semester, but have continued to program and try to finish the project for the original delivery date. This paper will discuss the overall approach we have taken to undertake the project and the process of implementing that approach. We will be happy to answer any questions and concerns you may have and are even happier to be almost finished with the project.

Thank you,

Brent Haslem

Robert Taggart

Brandon Wilson

TABLE OF CONTENTS

Final Design Report for the VHDL

Implementation of a Data Equalizer

ECE 4850
Apr. 28th, 2003

Brent Haslem
Robert Taggart
Brandon Wilson

Instructor Approval _____ Date _____

Dr. Kevin Moore Professor in Electrical and
Computer Engineering Dept. at Utah State University

1.0. INTRODUCTION:

1.1. Subject and Purpose.

L3 Communications, based out of Salt Lake City, has finished work on a project and are now finishing plans for the upgrade of that product. They have created a wireless telephone/Ethernet system that is in operation in a few countries. Currently the system is unable to effectively transmit signals to receivers that are not in a line of sight path with the transmitter. L3 has almost finished the project of upgrading their system to be able to handle non-line of sight transmissions. Our design project has been to assist them in that upgrade, which specifically is to program a data equalizer in the VHDL coding language. When this is complete and L3 has made the other necessary changes to the system it will be capable of communicating with all the receivers in the transmission area (both line of sight and non-line of sight). L3 is marketing to countries where a land based cable telephone system does not exist, and will eventually expand the market to all nations.

1.2. Brief Problem Statement

For a transmitter that broadcasts a signal over a 25 miles radius there are countless obstacles between the transmitter and receiver that create a multi path environment. The problem with this is that many of the telephone subscribers in this affected area will not be able to receive a direct signal and establish a reliable connection. An effective system needs to be able to accommodate for those situations. This final report shows how, through spectral spreading and equalization, we were able to help improve the wireless system so that it is

capable of receiving these reflected and energy damped signals, and make the connection more reliable.

L3 has built the system using VHDL and has already installed the hardware needed for the system upgrade. Since we were dealing with an already functioning system we had the task of trying to integrate our design of the data equalizer into an already existing unit. As a result all of our coding was done in VHDL and needed to be written to meet all the required timing specifications, as well as accommodate the hardware that had been installed. Writing the code was not nearly as difficult as trying to ensure that the coding for the data equalizer met the hardware and timing specifications of the system. Most of our time was spent, once our code was written, trying to integrate our project with L3's existing design and has proven to be an unbelievably difficult task, much more difficult than we had previously anticipated.

1.3. Summary of Design Process Executed.

The body of the report will cover the solution in more detail, but the following is a brief summary. Each subscriber signal being broadcast from the base station transmitter is multiplied by a unique pseudo noise code (these PN codes will be explained in more detail later). The resulting signals are then mixed together and broadcast in regular radio communication fashion. The transmitted signal then suffers various distortions due to the effects of the environment through which it is traveling. When the signals reach the receivers they are equalized to accommodate for timing offsets. Each receiver then multiplies the received signal by an assigned PN code. If the PN code at a given

receiver is identical to the PN code that the signal was multiplied by at the transmitter, the subscriber unit is able to decode the transmitted information.

If the PN codes are not identical at the transmitter and receiver than the receiver will only acknowledge the signal as noise. If the received signal does not have a direct line of sight with the transmitting base station the equalizer will begin to adapt its parameters to combine the time delayed signals and suppress the noise. Once the receiver has learned the channel properties it is able to adjust a pre-equalizer for transmission back to the base station. The resulting communication is not as fast as a line of sight connection, but is stable and can adapt to future changes in the channel.

1.4. Summary of Final Results.

The first part of this paper will show how we approached this design project using Matlab. We were able to affectively set up a simplified stand-alone transmitter and receiver system to mimic the properties of spread spectrum and equalization. By doing this we were able to see and understand the benefits of equalization and the problems that might occur later in the project with L3's existing system. This proved to be very beneficial when it came to actually writing the code in VHDL for the design blocks.

By gaining an overall understanding of a simplified system it helped us in integrating our design with L3's. We were able to write effective and working code that gave us positive results when testing the blocks functions. We accomplished this by using L3's test bench in VHDL and applying the same principles we did in our Matlab simulations. Once again this was a simplified

version of what the final result will be once our code is integrated with L3's, but it did show that our code worked correctly.

1.5. Organization and Summary of Report

The remainder of the report will show how we analyzed the problem and the decisions we made within the specifications given to us. The report then addresses the solution of the design problem using a Matlab simulation to demonstrate functionality. The paper will also include our VHDL code of the blocks we were required to program to finish the equalization process. Figure 1 of the appendix shows the overall block diagram of the wireless system and the work breakdown structure outlines the six blocks we had to program. Finally the paper will address any changes that occurred over the time period the project took place, project management aspects, and a conclusion including what we've accomplished and the educational benefits of undertaking this project.

2.0. PROBLEM ANALASYS:

2.1. Review of Problem.

As mentioned in the introduction our design has been to assist in the upgrading of a wireless telephone/Ethernet system. The problem with the current system is that in the real world all receivers for the system do not have a direct line of sight to the transmitter. For L3's transmitters, which have the ability to transmit 25 miles in radius, it is reasonable to say that a large percentage of the receivers will not have a direct line of sight with the transmitter. This would eliminate a wide client base that would be unable to establish a communication link. To avoid losing such a large potential for subscriber unit sales the existing

unit must be modified to extend service to all who want it. For this reason L3 opted to upgrade their design and make it possible for non-line of sight receivers to detect signals sent by the transmitter through a multi-path environment. In order to make the upgrade a reality L3 must include as part of their design a data equalizer. We were responsible for integrating this equalization block into their existing design. Through this and other changes to the system L3 will be able to create a wireless communication system to reach all subscribers in a given transmitter area.

2.2. Summary of Specifications.

The existing software for the current telephone/Ethernet system has been coded in VHDL, as a result we are required to use VHDL as our programming language and have been given the following hardware specifications. There are currently eight data channels in the existing system, this has been extended to twelve data channels. This provides a higher bandwidth for the system operation. The equalizer/pre-equalizer must be designed to fit in a portion of a Xilinx Spartan lle300 FPGA (block diagram for in figure 2.1). The existing channel circuitry will require one thousand Logic Cells of the FPGA leaving 5144 for this design. One thousand and fifty of these remaining logic cells (a logic cell is a 4 input logic function that can perform any 4 input combinational function) will be used for the Microblaze processor (which will be used for the processing of values in the LMS engine). This leaves us with four thousand and ninety four logic cells of the Xilinx to work with. The system also needs to fit the within the number of input and output pins available on the Xilinx chip. While these

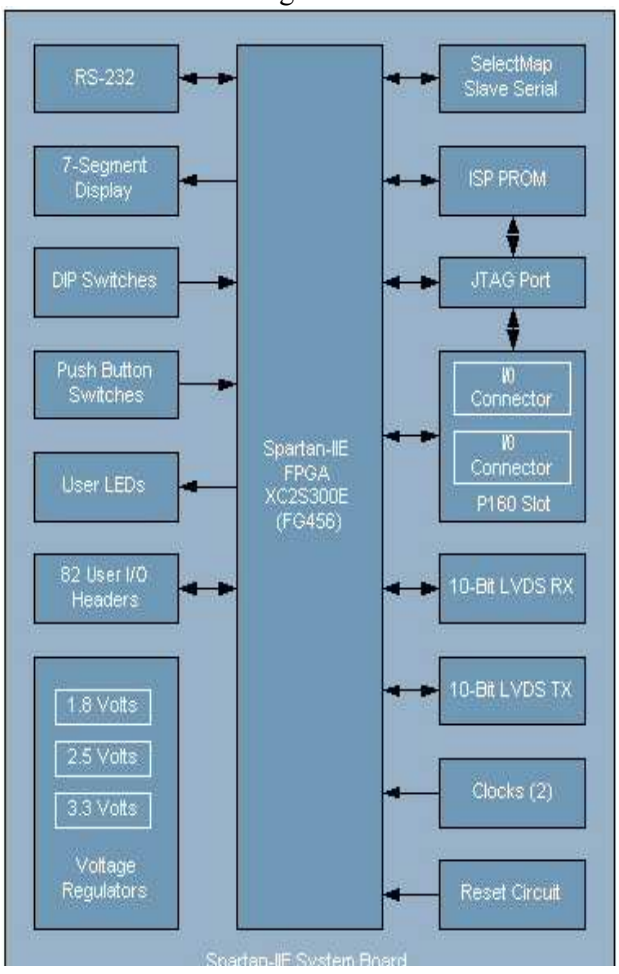
specifications are still a requirement we were unable to test our code to ensure that it fit into these specifications. Once our code is effectively integrated into L3's system whomever implements a hardware test of the project will be able to take the code we have written and make any minor necessary changes to ensure that it fits into the timing and hardware specifications. We were not able to test the hardware because Primewave is no longer going forward with the project because their division has been temporarily put on hold for financial reasons. As a result we did not have access to L3's labs or hardware.

Whenever the project is continued and our code implemented timing restrictions will need to be taken into account when changing the VHDL code. Once the design is placed and routed into the Xlinx Spartan 300, it is timed using a maximum 92 Mhz clock. Our chip rate is 5.44 Mhz making the operating clock for the Xilinx 87.04 Mhz, 32 X 5.44 Mhz. Meeting the clock timing requirement of 92 Mhz will allow the FPGA to operate within temperature requirements, which will save production costs of approximately $5.00 per unit. So our code for the equalizer (specifically comp.-multiply/summer, dispreader, adjusting already existing code for equalized dispreader, delay-line multiplex, scalar, equalizer sum) must fit into the Xlinx chip and run within the specified 92 Mhz.

The unit must also accomplish an effective data rate of 384 mega bits per second for the equalized system. This can be accomplished through splitting the data transmission over the expanded twelve channels and in the future could possibly be expanded to sixteen channels.

Initially L3 wanted to have the complete, working system by no later than April. However, in the middle of March L3 terminated primewave (the subsidiary of L3 that was sponsoring our design) and as a result the marketing of the project has been pushed back temporarily and possibly permanently. Because of this our project now must be completed by the end of April before the school semester ends.

Figure 2.1



2.3. Discussion of main features and Summary of Engineering Approach.

A Brief History of Spread Spectrum:

While spread spectrum technology may seem a newcomer to the digital communication scene, it has actually been around for nearly fifty years. However, it has only been within the last ten years that this technology has been introduced into the commercial market on a large basis. The development of spread spectrum has occurred primarily in the military, which has made use of several valuable characteristics of spread spectrum transmission namely, low probability of intercept and strong anti-jamming properties. Recently, spread spectrum techniques have been used in a wide range of commercial applications ranging from wireless local area networks (LAN), to digital cellular telephone networks. To better understand how the properties of spread spectrum have been used in some of these applications it is useful to see just how spread spectrum works.

An Overview of Spread Spectrum:

The term spread spectrum has been given to several forms of data transmission including frequency hopping, time hopping, and direct sequence spreading. For the purposes of this project spread spectrum will refer to the technique of direct sequence spreading. The defining characteristic of spread spectrum is that the bandwidth of the transmitted signal is much greater than the bandwidth of the information signal. To achieve this bandwidth expansion the information signal is multiplied by a sequence of bits known as pseudo-noise

code (PNcode). This stream of seemingly random bits is then modulated to a given frequency carrier signal and transmitted.

The result is a signal whose bandwidth is no longer determined by the information that is actually being sent. It may seem counterproductive to greatly increase the bandwidth of a data signal, but it is in the expansion of the signal that the benefits of spread spectrum are achieved. The first of these benefits is to make the sent signal look like simple noise in a channel. Anyone trying to identify a normal transmission would be able to tune a receiver to spikes seen in normal AM modulation. However, spreading causes the transmitted signal to look like random noise and would not appear as anything noticeable on a spectrum analyzer.

This leads to the property of low probability of intercept, which suggests a high level of security to a transmission channel. This has been of particular interest to military applications. Another result of this noise-like signal is the difficulty in jamming or interfering. To jam a signal another signal of similar bandwidth is transmitted to mask or confuse the information within a given bandwidth. Since spread spectrum is difficult to detect and is spread over such a wide spectrum it is very resilient to jamming efforts.

While these properties have their place in commercial applications, they have mainly been exploited by the military. The greatest attraction of spread spectrum is the ability to increase bandwidth utilization. Traditionally, time and frequency division multiplexing have been implemented to allow users to simultaneously transmit information. With spread spectrum a new “dimension” is

introduced to allow those same benefits. If each user has a distinct, orthogonal PNcode they can transmit information on the same band at the same time. This code division multiple access (CDMA) has become a major component in communication systems because it drives down costs while increasing performance. Another benefit is that spread spectrum has little effect on narrowband transmissions which share the same bandwidth since they appear as simple noise. The spread signal, however, can be negatively affected by a strong superimposed narrowband signal, so they do not usually share the same bandwidth.

The previously mentioned properties all depend on the processing gain of a spread spectrum system. This processing gain is defined as the ratio of the transmission and information width. Hence, the longer the pseudo noise code sequence the greater the processing gain of the system. This processing gain is the main factor that determines the number of users on a CDMA system, the difficulty to jam a signal, the difficulty to intercept a signal, and the reduction of multi-path effects on a transmission. Since this gain depends directly on the PNcode used it is useful to view several different families of PNcodes.

Pncodes:

The PNcodes that we have explored for this project are Walsh-Hadamard, Kasami, and Gold codes. Kasami and Gold codes are both produced by linear feedback shift registers. A register of length ‘n’ is fed by taps at several locations in the register creating a PN code of length 2^(n-1). A length of 2^n is not

possible because the register state with all cells being zeroes would fail to produce anything other than a zero sequence. Obviously the larger the shift registers, the larger the length of the code produced.

Shift register sequences have the property of producing nearly equal numbers of ones and zeroes (the number of ones exceed the number of zeroes by one). This helps to avoid a bias that would show a low frequency component spike in the frequency spectrum. These sequences also have strong auto-correlation, but are unfortunately not orthogonal. Gold codes are produced when two sequences exhibit only 3 cross correlation values: -1, 2^((n+2)/2), and -2^((n+2)/2). The codes are then produced by delaying the possible starting values in the register, this yields 2^n+1 different code sequences. Kasami codes are a subset of Gold codes, and are formed by taking spaced samples from a Gold code. This preserves the properties of the Gold codes but allows a much larger set of possible codes from which to select. A sample Kasami code that has been used in the design of the equalizer is shown in Figure 2.2.

The Walsh-Hadamard codes are not produced from shift registers, but are designed specifically to have the property of being orthogonal. This single property greatly simplifies the design process of a CDMA system because there will be zero interference between subscribers. This can be compromised by a multi-path environment due to the interference with time delayed signals. This interference is the basis for the equalizer that we are beginning to design. Accounting for PNcode matching of various delayed signals will make it possible to reconstruct the original signal, including the zero cross correlation property of

orthogonal codes. In the spreader of our system we have converted the bits to ±1 to create a zero mean code, eliminating any DC component in the PNcode spectrum.

The following are plots exhibiting the correlation properties of Kasami and Hadamard PN codes. Note that the Kasami codes have a single strong auto-correlation peak as compared with several shifts in the PN code sequence (Figure 2.3). However they show varying degrees of correlation with other Kasami codes (Figure 2.4). The Hadamard codes exhibit different degrees of autocorrelation (Figure 2.5) but very low cross correlation with other Hadamard codes (Figure 2.6).

Figure 2.2

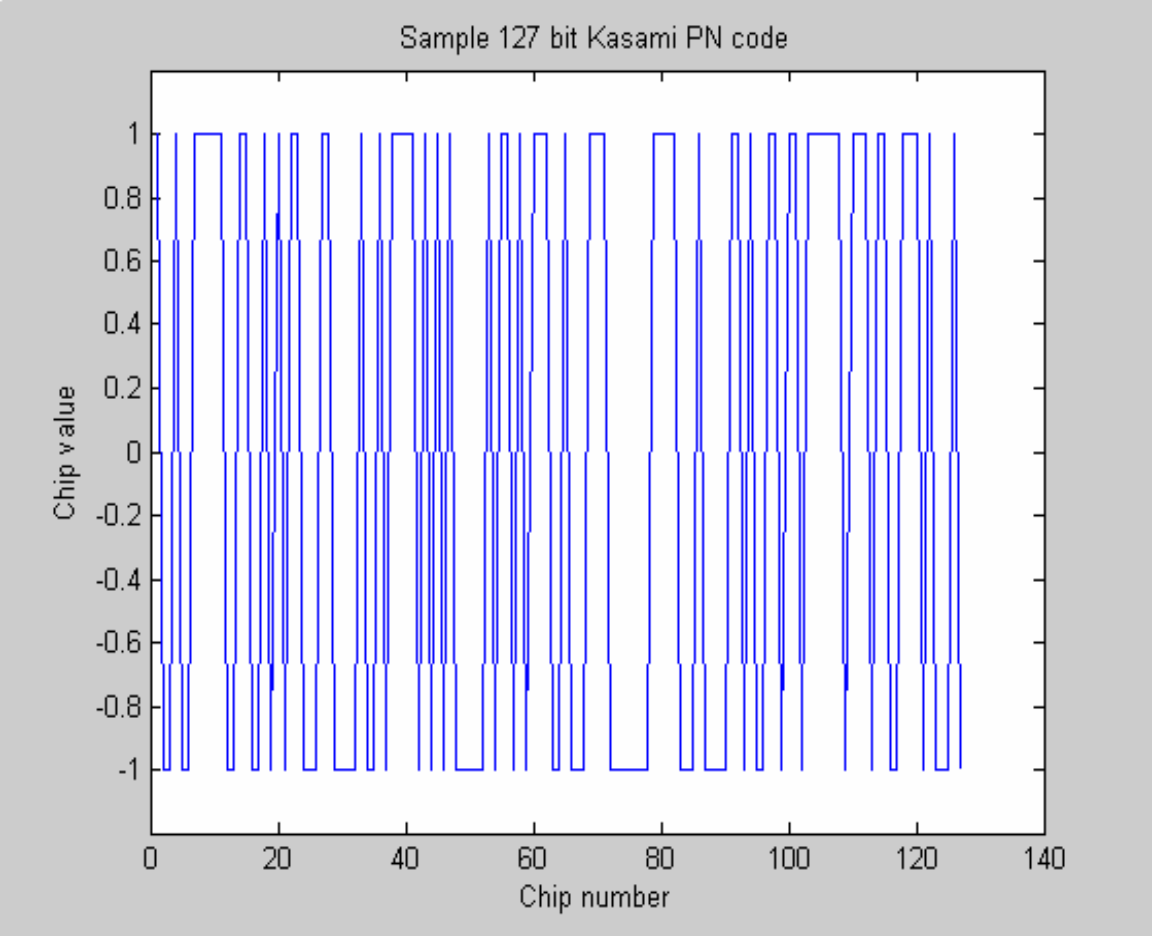


Figure 2.3

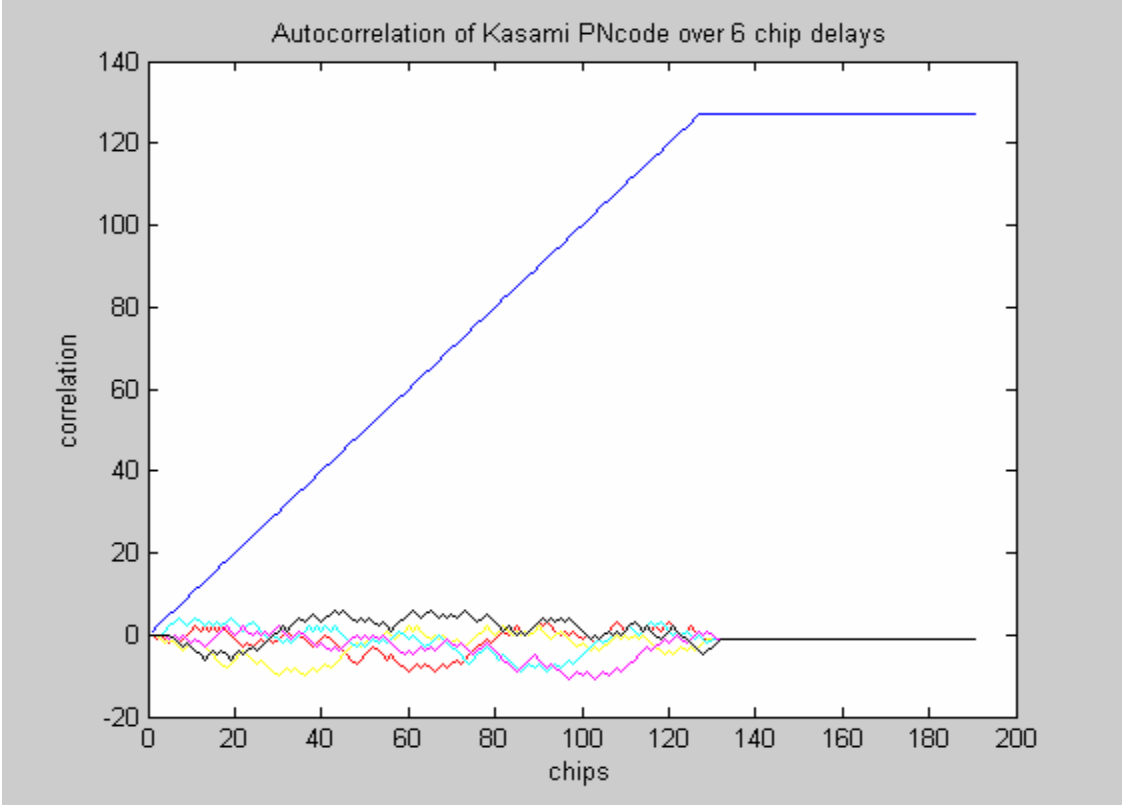


Figure 2.4

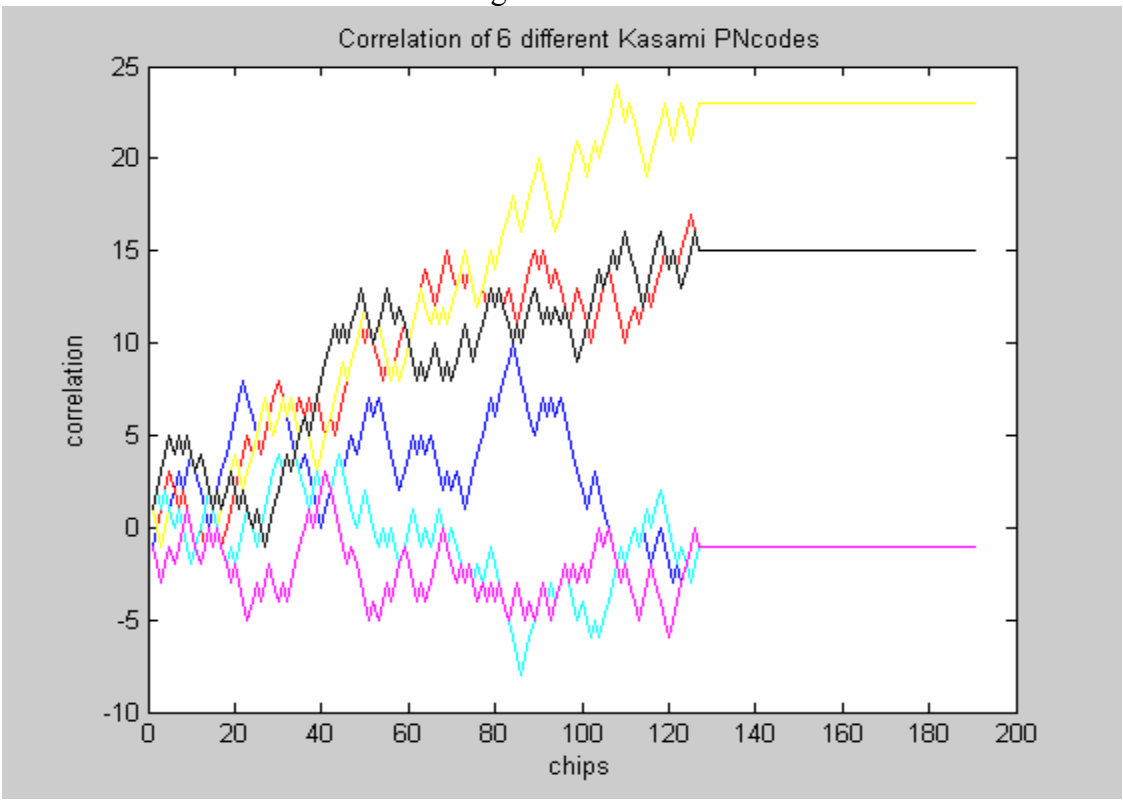


Figure 2.5

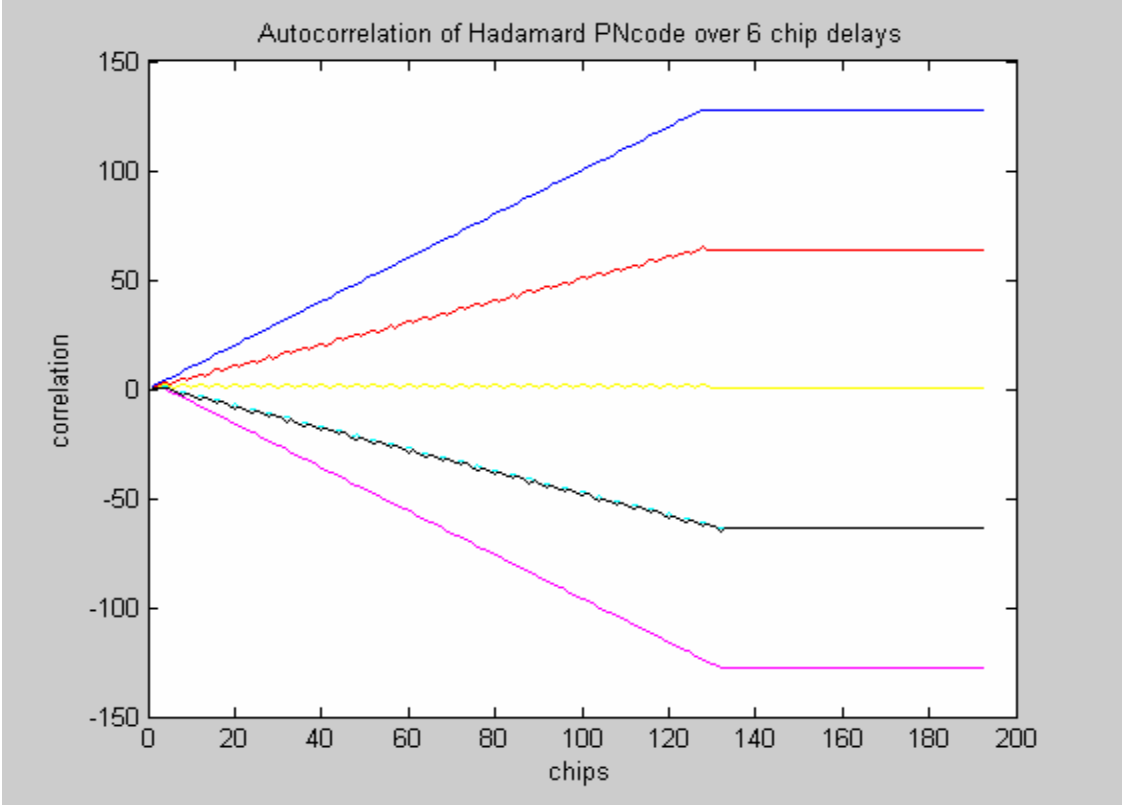
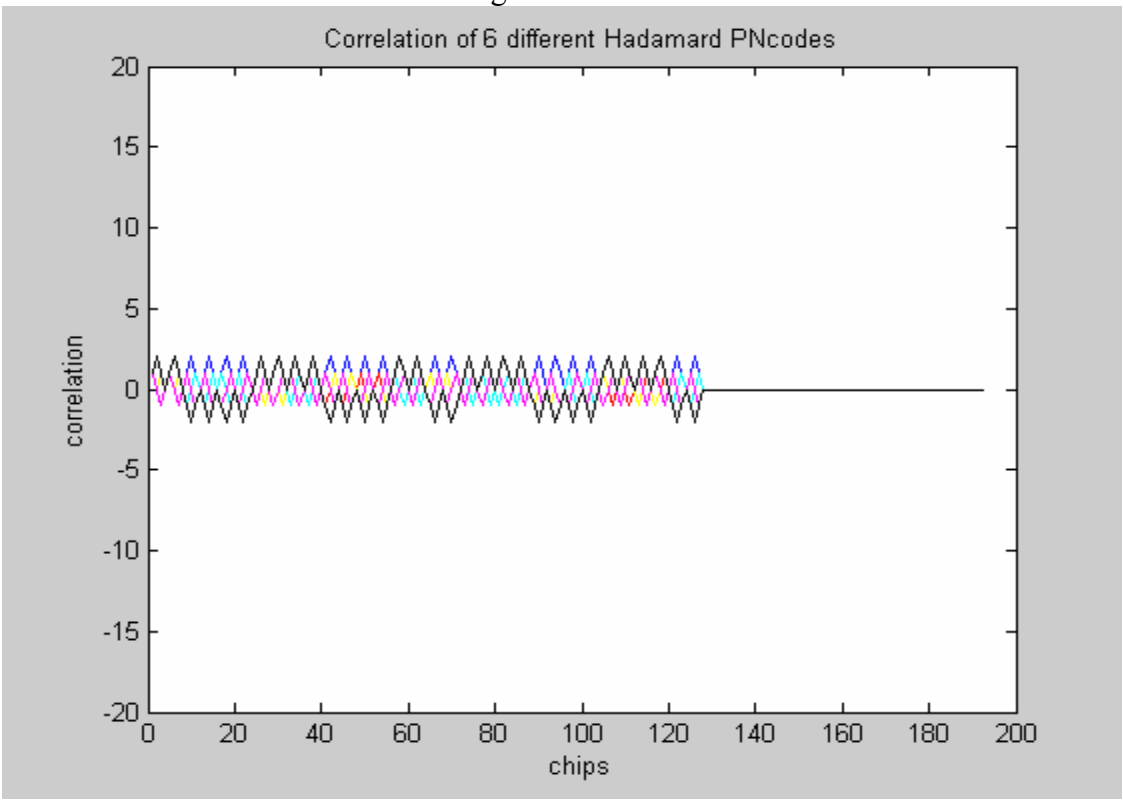


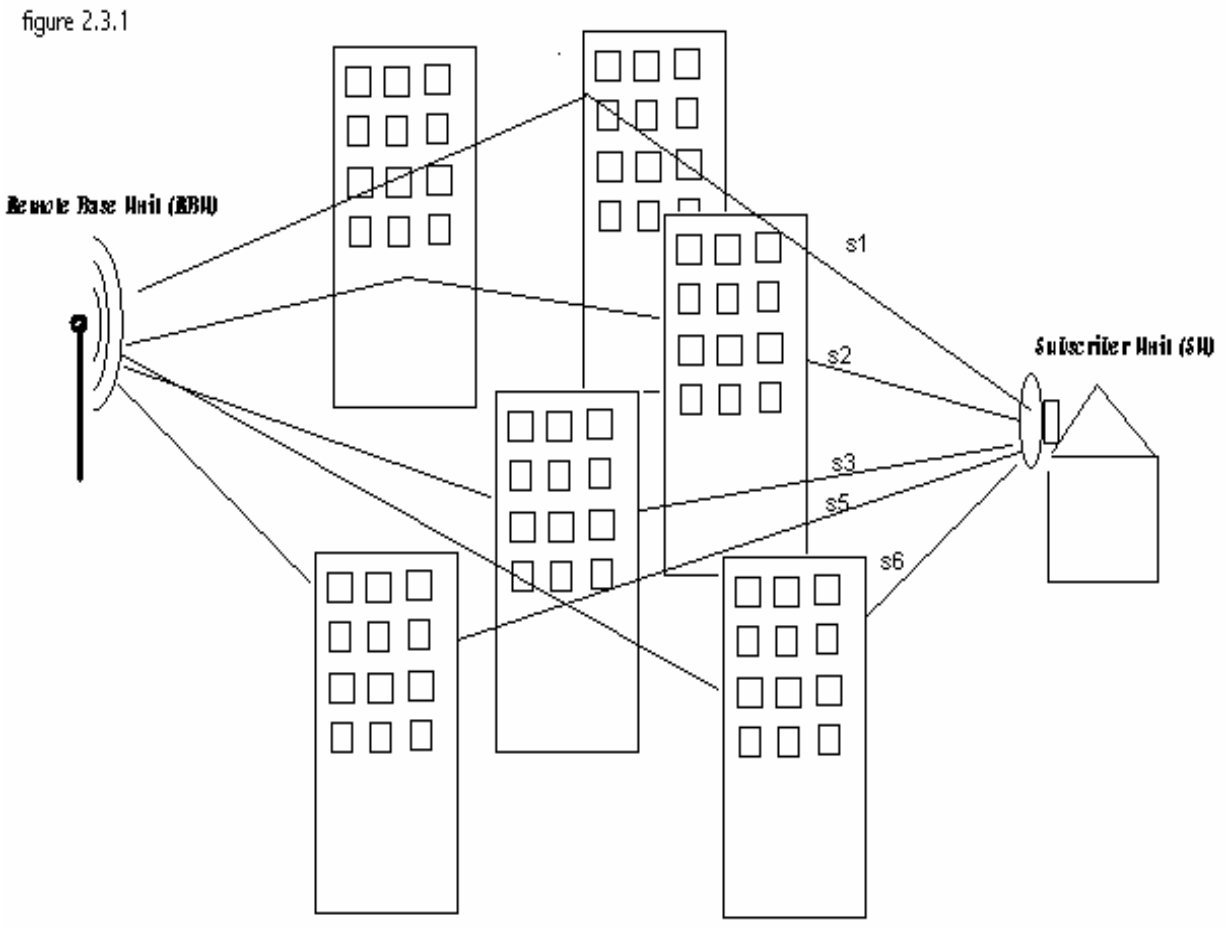
Figure 2.6



Equalization:

Figure 2.3.1 is an example of why equalization is important and the role that it plays in the real world. The need for equalization in our project arises from implementing wireless receivers that are not in a line of sight with the transmitter, and environments that create multiple copies of a signal at the receiver. Figure 2.3.1 shows how received signal becomes a collection of time delayed and scaled versions of the original signal. Later in the report we show, through the use of matlab simulations, how we implement equalization as a solution for the design

figure 2.3.1



2.4. Decision Analysis

The opportunities we had to make decisions or go different directions with this project were quite limited. Since the project was a company-sponsored project we needed to ensure that our design met the criteria that L3 delivered to us. The main restriction was the fact that our design/code is going to be integrated into an already built and functioning system. Some of the decisions regarding how our design can be implemented have already been determined. Most of the decisions made came during the coding and Matlab testing process of the design.

The biggest decision made from our Matlab implementation of the design was the choice of the PNcode we used for the spreading of the signal. We made a final decision to use a Hadamard code because of its orthogonal properties, which were previously exhibited. Other decisions during the coding process were simply decisions based on how to write the code and more decisions will need to be made when completing a successful integration with L3's overall system.

A final decision was made to extend the delay block from 12 taps to 16 taps. This allows a larger number of delayed signals to correlate, improving the quality of the received signal. Also a final decision was to stick with 12 data channels and not reduce them back to eight. Even though our code has been written to accommodate both of these decisions, they could be changed when a hardware test is implemented if timing and fitting specifications are not met. To the best of our knowledge, however, there should not be any problems.

3.0. Performance Optimization and Design of System Components

In discussing the performance of our project there are two key processes involved. The first process involves the application of spread spectrum. One of the most key decisions or changes that can be made to impact the overall effectiveness of spread spectrum is what PN code is being used. Originally we were under the impression that we were to choose the PN code used in the spread spectrum process. Later we found that one of the doctorates at L3 was "inventing" a new PN code that combined the benefits of orthogonality and shift orthogonality to give the system the best results. Since the breakdown of Primewave however, no such PN code has been made available. To finish our design and obtain desired results we needed to implement a PN code and chose a Hadamard code because of its orthogonal properties. This gives us peak performance for the spread spectrum and also simplifies, and makes more effective, the equalization process. Equalization is the second key process for performance.

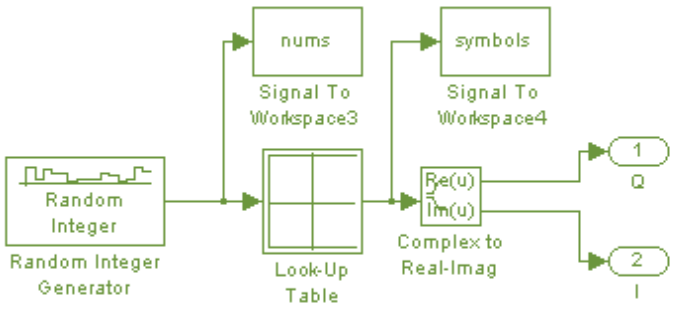
The next several pages of the report go over first the performance of the equalization process in our Matlab simulations. Second we will transition from the Matlab design to our design in VHDL and include simulation and results from our testing. We are including the Matlab simulations because that was the basis for our VHDL coding and made the actual implementation of the project much easier. Timing and fit into L3's design was in the beginning a big performance issue. After Primewave was demolished and the scope of our project changed drastically we decided to try and keep timing and fit as part of our performance

goals so that if L3 decides to continue this project our efforts could be of some use to them. The nest section will show that the performance, to the best of our knowledge of the system to this point, went quite well.

4.0. Discussion of Technical Approach Used and Project Implementation.

In section 2.0 we discussed the concepts of spectral spreading and equalization and the following graphs, schematics, and text give further explanation and show our solution to this design problem.

Figure 4.1 – Source data generator

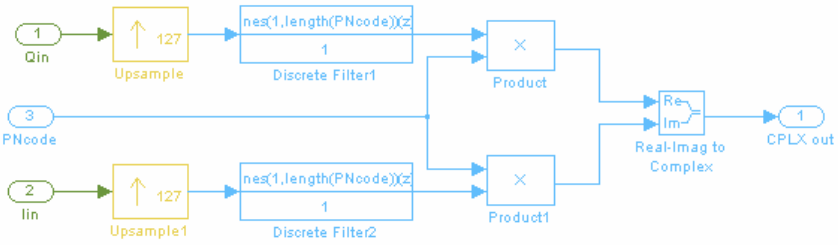


Source (See Figure 2.1):

The source block effectively converts an input stream of bits into a complex number for quaternary phase shift keying (QPSK). The data source here is actually a random integer generator that produces values from 0 to 4 with equal probability. These integers represent a two bit slice of the actual binary input. These integers are then passed into a look up table (LUT) that maps the

integer to one of four possible complex valued symbols at (±√2, ±√2). This symbol is then split into real and imaginary parts representing the in-phase signal and the quadrature signal. The in-phase signal effectively conveys the data in the first bit of the two bit slice and the quadrature signal carries the second bit.

Figure 4.2 – Signal Spreader



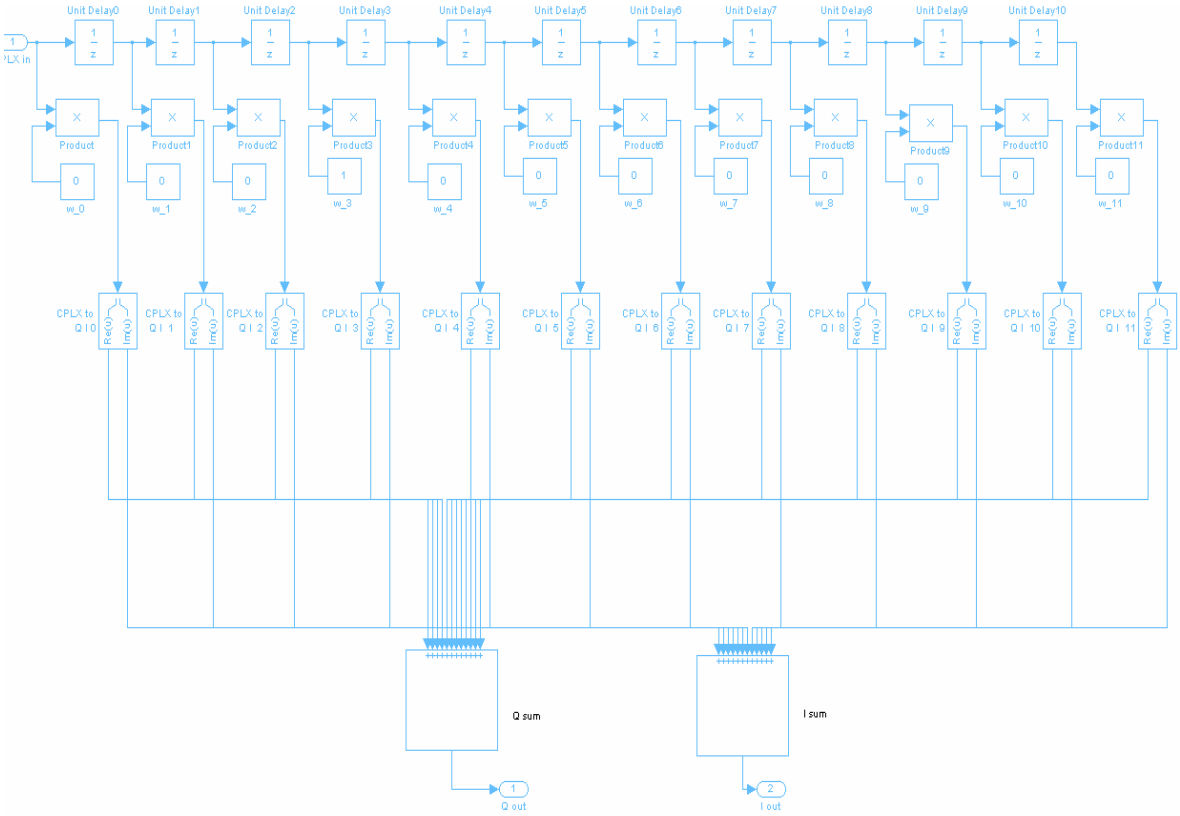
Spreader (See Figure 4.2):

The key component of any spread system communication system is obviously the spreading of a signal over a larger bandwidth. To accomplish this, the I and Q signals are up sampled by the length of the pseudo noise code that is to be utilized. This converts the signal into a digital impulse train with a period equal to the length of the PNcode. This impulse train is then fed into a pulse shaping filter.

At this point the filter implements a non-return to zero (NRZ) pulse train but may be adapted in the future for a more realizable filter. Again the resultant pulse train has a period equal to the length of the PNcode. This pulse train is

then modulated by the PNcode (whether Gold, Kasami, or Hadamard doesn't matter at this point), which produces a series of chip values of ±1 that uniquely represents the I and Q symbol values. These chip signals are then sent to the modulator for transmission over the channel.

Figure 4.3 – Channel Simulator



Pre-Equalizer/Channel Simulator (See Figure 4.3):

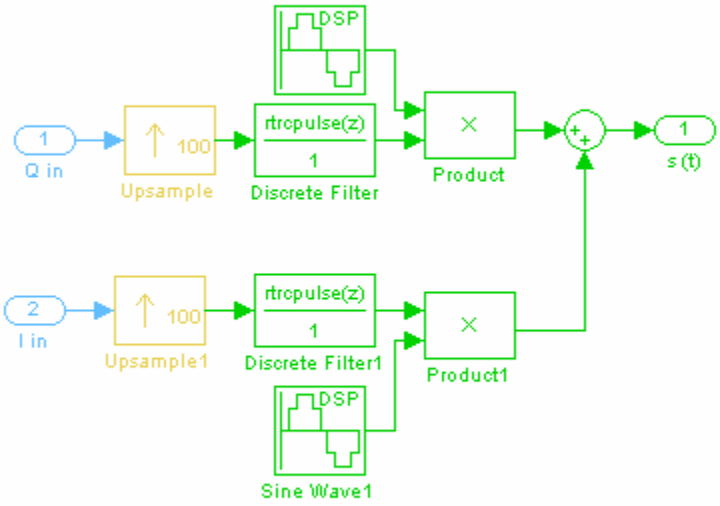
A problem arises for orthogonal spread signals in a multi-path environment. Due to the varying time delays inherent in the channel the pseudo noise codes begin to lose their orthogonal properties. This can lead to inter-

symbol interference that can make decoding the signal difficult if not impossible. An equalizer may be used to sort out the time delays and adjust itself to compensate for the adverse effects of the channel on the signal. This allows the received signal to be processed as if little or no distortion has occurred. Thus an equalizer may be used at the subscriber end to decrease the probability of error in decoding sent signals.

This could also be implemented on the base station downlink but a separate equalizer would be required for each subscriber. To remedy this problem a pre-equalizer may be implemented at the subscriber end to pre compensate for the expected channel distortions. As the equalizer learns the channel an inverted equalizer can calculate the necessary gains so that the signals received at the base station appear to be orthogonal. In this design we have also used a pre-equalizer to simulate a multi-path channel by specifying the time delays to be used.

The pre-equalizer is implemented by passing the I and Q signals into a delay line multiplexer. This delay line may be adjusted to have full or half chip delays. Each tap of the delay line is then multiplied by a weight to determine the strength of the signal at that particular time delay. The weighted, time delayed I and Q signals are then summed and are ready for carrier modulation. Whether the design is used for a pre-equalizer or a channel simulator the design and implementation are nearly identical. The only difference being that the pre-equalizer weight values would update as the equalizer learns the channel properties.

Figure 4.4 – Carrier Modulator



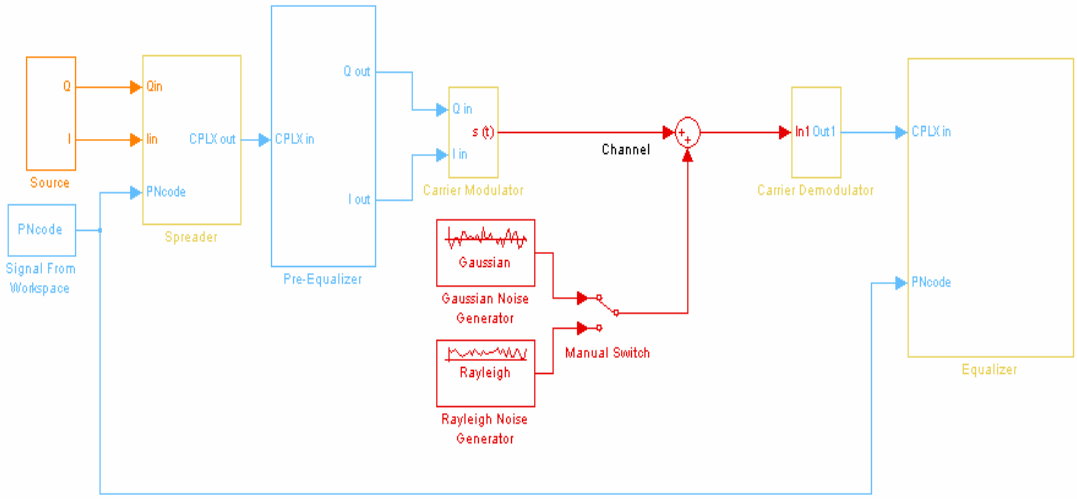
Carrier Modulator (See Figure 4.4):

The carrier modulator prepares the bipolar chip values for transmission over a channel. It does this by first pulse shaping and then amplitude modulating to the desired carrier frequency. To shape the pulse the I and Q signals are again up sampled by a factor equal to an integer multiple length of the desired pulse symbol period. For a square root raised cosine pulse that extends over twelve symbol periods we used an upsampling factor of 100. The signals are then passed through the pulse shaping FIR filter.

The square root pulse is used so that the output of the matched filter in the demodulator exhibits zero inter-symbol interference. The I and the Q pulses are then modulated by cosine and sine carrier waves. The frequency of these sinusoidal carriers may be adjusted to comply with regulatory specifications. The

sine wave is then subtracted from the cosine wave and the combined signal is ready for transmission. Due to the fact that the sinusoids are orthogonal they may occupy the same bandwidth without destructive interference.

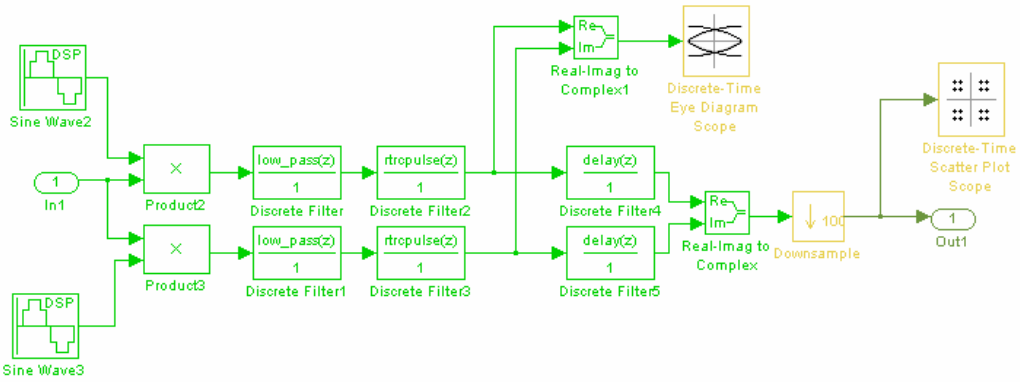
Figure 4.5 – Channel Simulation



Channel (See Figure 4.5):

A channel can be simulated by simply adding white Gaussian noise to the transmitted signal. If a channel simulator is not used as described above another FIR filter can be used to simulate a multi-path environment. The result is a varied time delayed, scaled, and noisy signal that simulates a realistic transmission channel. The model shown allows the user to manually toggle between a simple AWGN channel and a Reyleigh multi-path interference.

Figure 4.6 – Carrier Demodulation



Carrier Demodulator (See Figure 4.6):

The carrier demodulator brings the signal back to base band and then reproduces the chip values by sampling at the correct times. To accomplish this, the signals are first multiplied by synchronous same frequency sine waves. The phase correction must occur in a synchronization circuit such as a phase lock loop. For the purposes of this simulation the carrier waves were assumed to be fully synchronous. This carrier wave multiplication results in a reconstructed base band signal as well as components at double the carrier frequency. To remove these components the signals are passed through a low pass filter, preserving only the base band component.

At this point the signals are ready to be passed through a matched pulse shaping filter v (T-t). Since the root raised cosine pulse is symmetric the identical filter that is used in the modulator may be used here. The signals exhibit zero inter-symbol interference and are able to be sampled to restore a chip train. If

two samples per chip are required an additional matched filter may be used with a half symbol delay. This would produce a resultant signal with half chip spacing. This will reduce the chip span of the equalizer, but is more robust in determining the actual chip value sent. The eye diagrams and scatter plots show that the correct chip values were in fact detected with zero inter-symbol interference. These chip values are then ready to be passed to the equalizer for correction of the distortion caused by the channel.

Figure 4.7 is the schematic of a signal undergoing the process of spread spectrum. The system starts with the signal as bits, converts the bits to symbols, and creates a signal of the desired length by use of upsampling and filtering (signal shown in figure 4.8). At this point the signal is multiplied by our chosen PNcode (labeled as code 0), and transmitted. Once this occurs our signal should look exactly like noise, as shown in Figure 4.9. In order to do this a gain block has been inserted. The purpose of this block is to match the average spectral magnitude of the spread signal to that of the noise in the channel. This is only relevant and needed to avoid interference with other narrow band signals and to avoid detection. The gain of this block will be determined by L3 to set the appropriate probability of error. The Gaussian Noise is then added to the signal for real world purposes before the receiver picks up the signal (figure 4.10).

The first step of the receiver is to multiply the signal by the exact PNcode used at the transmitter. Once this operation is complete our signal should then look exactly like the transmitted signal, as revealed in figure 4.11. From there it is a simple matter of down sampling the signal and converting it back to bits to

process the data. Once to this point it was not too difficult to take these concepts implemented in Matlab and transfer them to VHDL code. This code will be available at the end of this section 4.

Figure 4.7

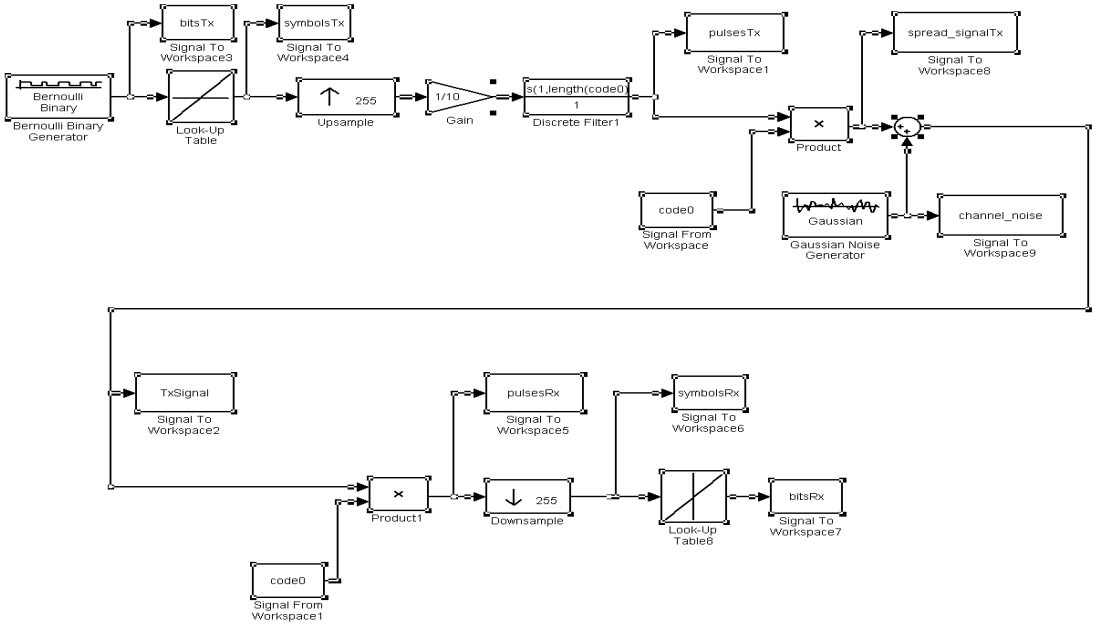


Figure 4.8

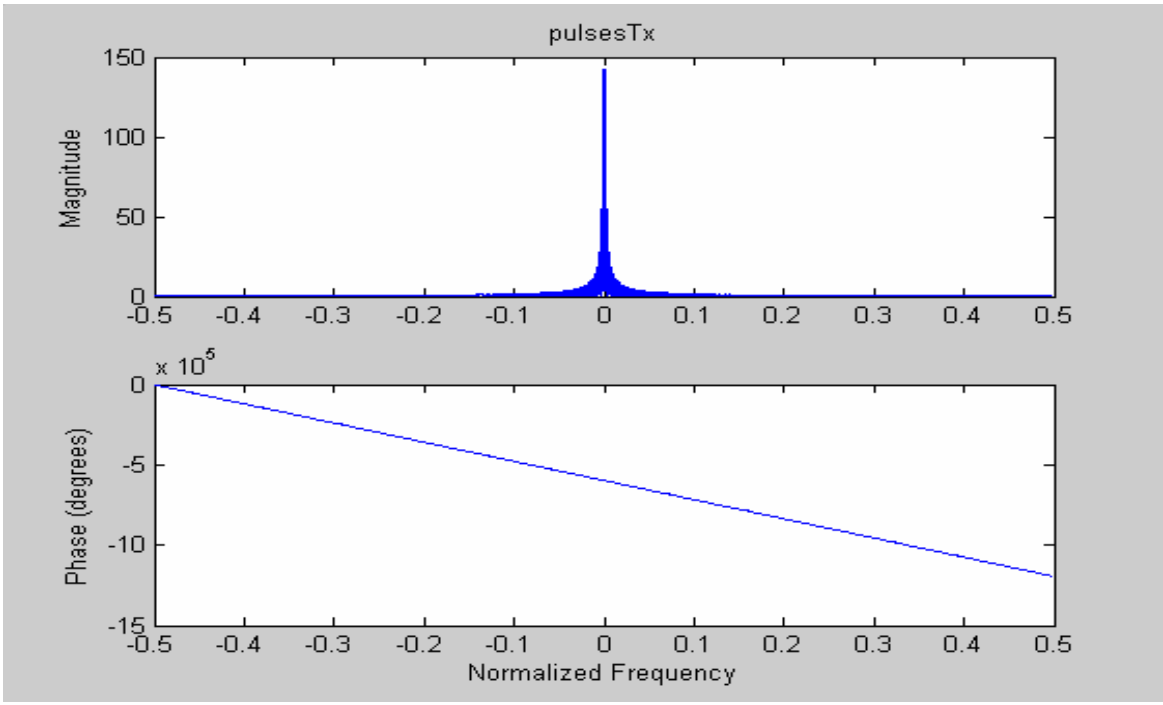


Figure 4.9

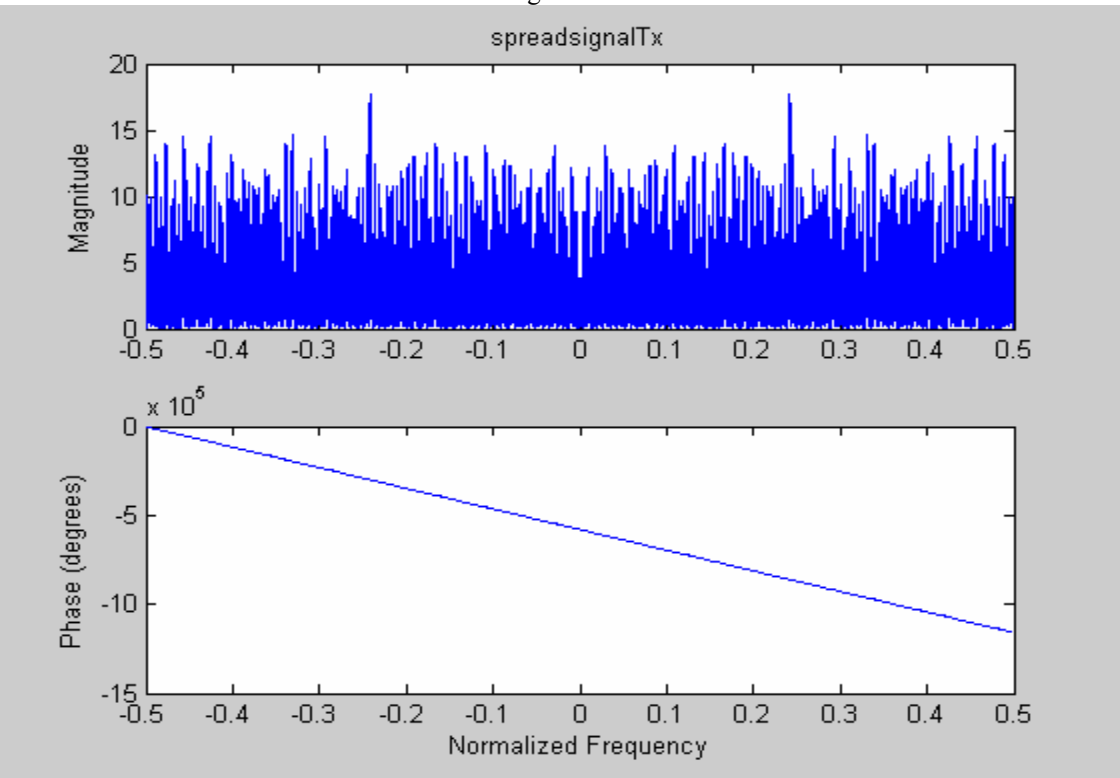


Figure4.10

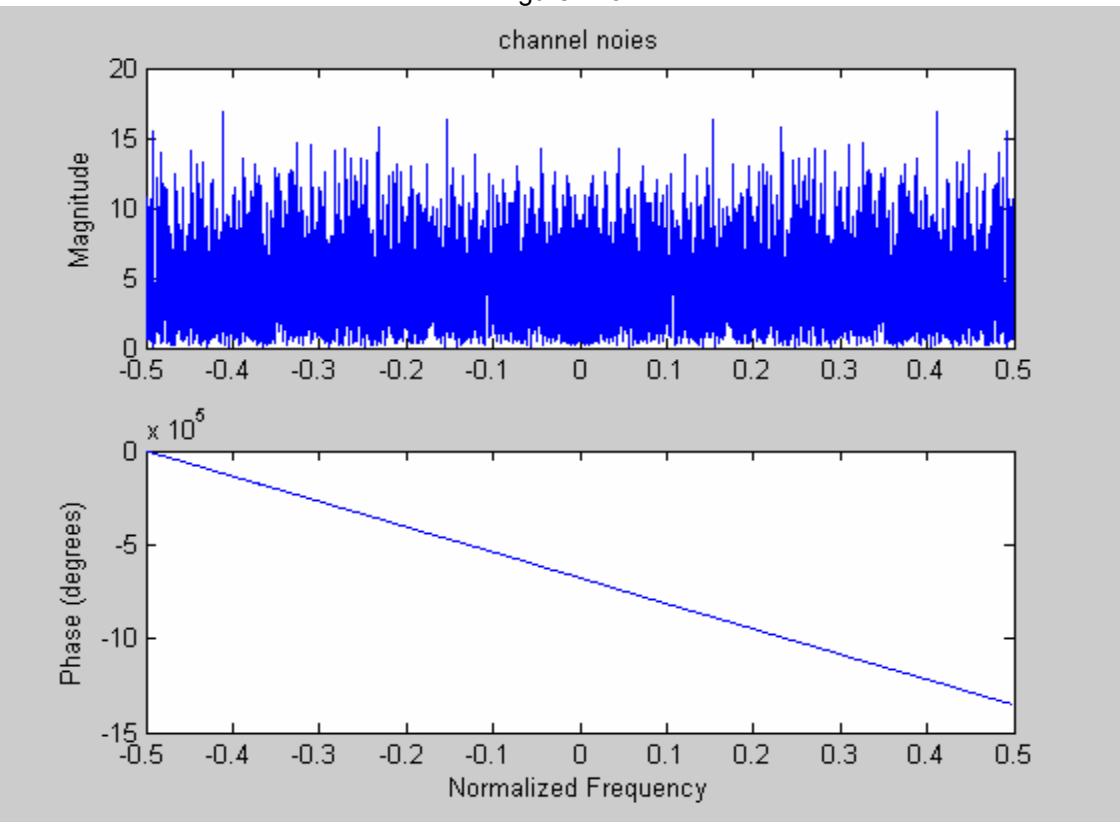
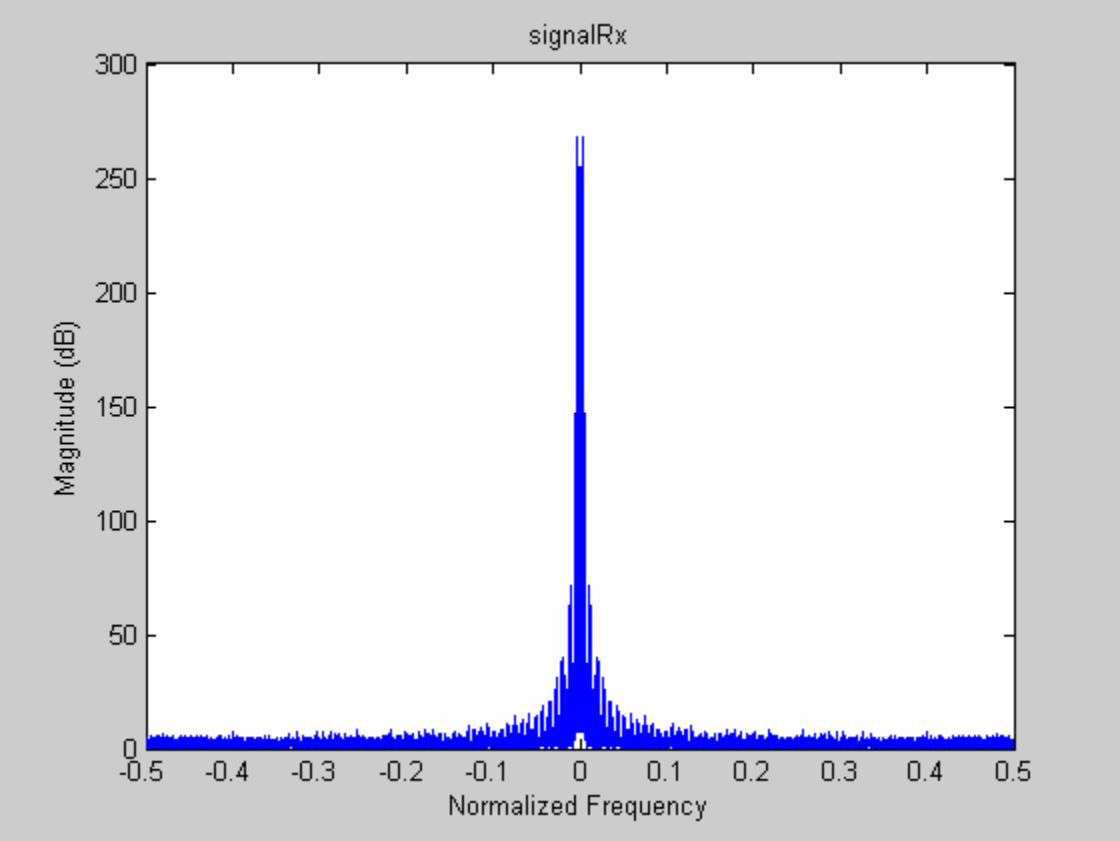


Figure 4.11

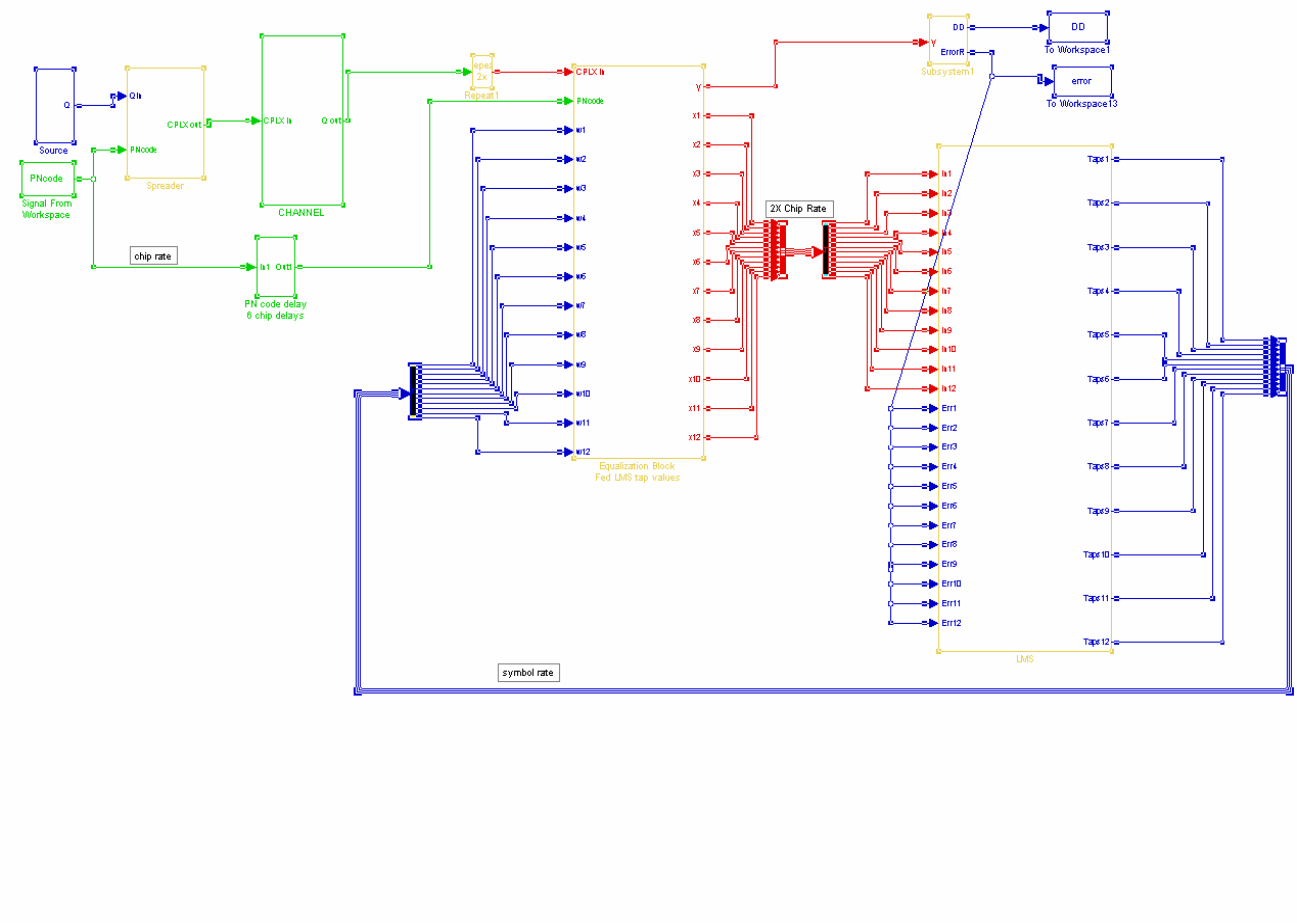


Equalization Execution:

Next we will show the implementation of equalizing a signal through the use of an LMS engine (least mean squared error). Figure 4.12 is the top-level diagram of our equalization simulation and represents all the blocks and there interconnections. It is a representation of a multi-path environment where $S_{Rx}=\sum ai*(S_{Tx}(t-di))$ (For all paths indicated by subscript i). Here ai represents the gain of a given received signal and di represents the corresponding delay factor. In this diagram (Blue = symbol rate, Green = chip rate, Red = 2X chip rate), the source is producing random data, which is then converted into our transmitted signal. This data is up sampled and multiplied by our PNcode, this block is

producing spread spectrum. The spread data is than passed into the next block, which is simulating the channel. This block is a 6 tap FIR filter where the tap values represent the delayed versions of the signals.

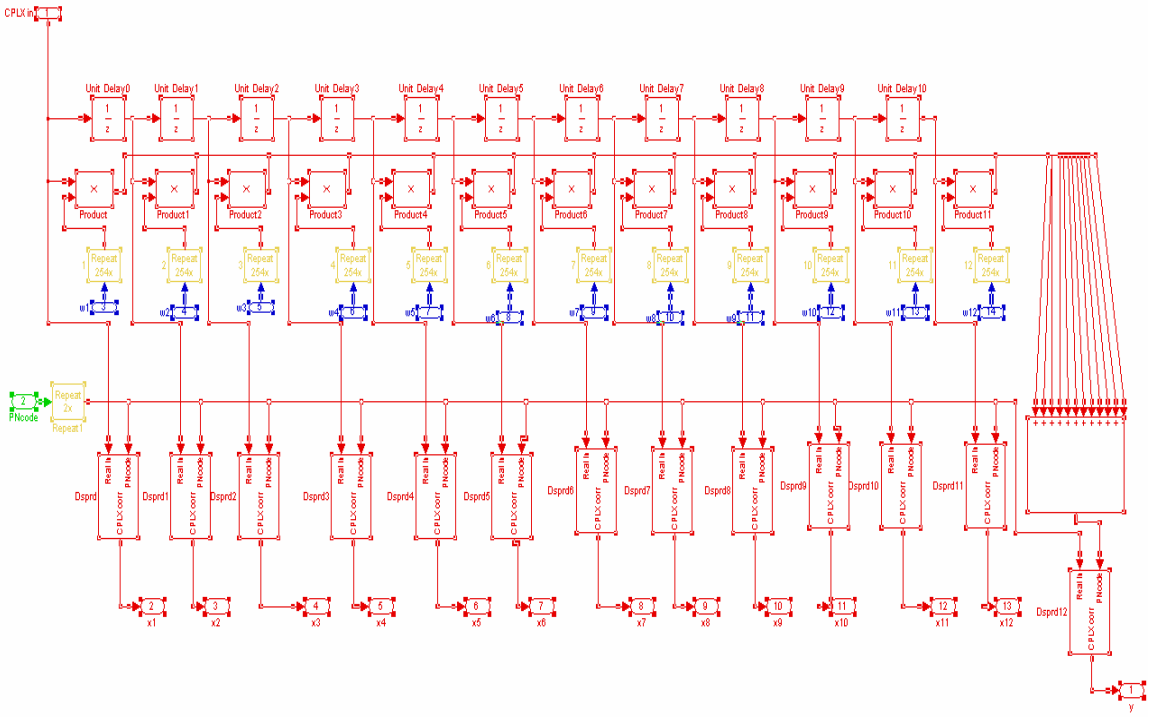
Figure 4.12



We will now explain the contents of the equalizer block, which are represented in figure 4.7. The equalization block is also a fir filter. Because of design constraints we are sampling 2 times per chip (represented in the top level diagram). This will give us are required 12 tap filter. These taps are fed with the values given by the LMS engine into the equalizer block.

The PNcode input has been delayed by 6 chips in the upper level diagram. This is in order to synchronize the filter, and center where our tap values represent the delayed versions of the signal. Without this we would get no correlation values for the output because we would start multiplying our PNcode by the delayed versions before the delayed versions have time to fill the delay blocks. There are two outputs from the equalization block, both of which represent correlation values. These values are the correlation at each tap (xi) and the summed correlation of all the taps (See also Figures 21 to 24).

Figure 4.13



The outputs of the equalizer are run into the LMS block (figure 4.15) and an error block (figure 4.14). The decision/error block is taking the summed correlation given in the Equalizer block and produces two outputs. The outputs are the decided symbol (DD) and the error. First we will discuss how the error block computes the decided symbol, which is simple sign recognition. If the correlation is positive then the received symbol is a 1, if the correlation is negative then the symbol is a -1. This result is then down sampled at 2X the chip rate in order to take it to the symbol rate.

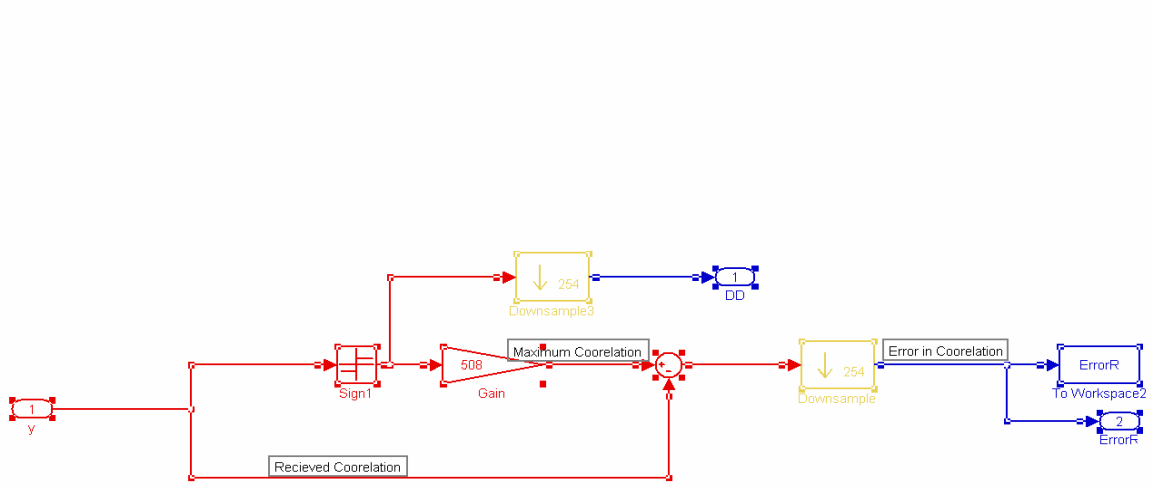
This simple solution may not produce the best results. If we happened to get a bad correlation value and it is the value that was down sampled then we will get an error. A way to reduce the possibility of an error would be to integrate the correlation and dump it every symbol period. At that point passing the signal through a sign block would greatly reduce the probability of error. We did not implement this in our design because up to this point we have not had these problems. We will no doubt have to account for that when transferring the design to VHDL.

As was mention the error blocks second output is the error. We calculate the error by taking the decided signal and multiplying it by a “max correlation” gain. This number can be tricky to figure out, because it depends on the length of the PNcode and the samples per chip of the equalizer. In our case the length of the PNcode=128 and we are running at 2 samples/chip. Because we are sampling twice per chip we must extend our number of taps to 12 . Each pair of these taps represents one delay on the channel. Therefore our maximum

correlation = length (PNcode)*2(samples/chip)*2(taps/delay). This maximum correlation allows us to subtract our decided signal from our received signal to give us an error value.

Note: This formula only works if the magnitude of the sent signal is normalized to 1.

Figure 4.14



The error output of our error block is fed, along with the equalized data, into our LMS block (figure 4.15). In this block we down sample the signal by 256 to bring it to the symbol rate. Once this is done the signal and the error are run into the LMS engine block of figure 4.15. The LMS engine (figure 4.16) is a very simple way of updating the tap values. It relies on 2 inputs and one parameter.

The formula for the LMS is wi[k+1]=wi+mu*error*xi. The new weight values at each tap are determined using a feed back loop where the past weight

value is summed to the product of the error of the system and the correlation value at the given tap. This is then multiplied by a step value mu. Based on the step value of mu we can damp the system as we see fit.

Figure 4.15

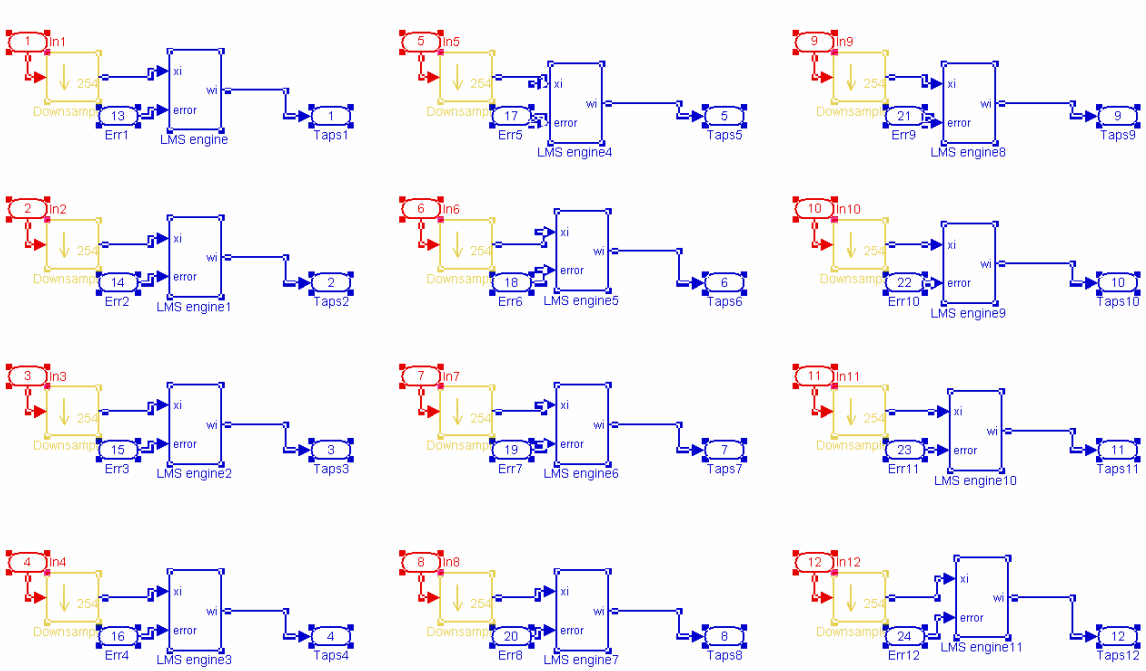
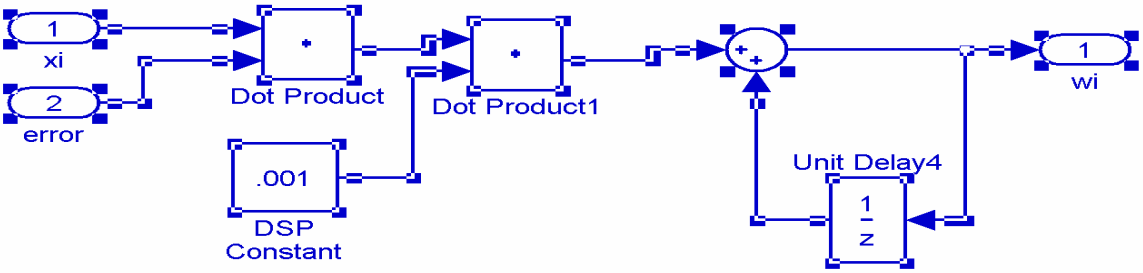


Figure 4.16



The following graphs show the influence mu has on the effectiveness of the LMS engine reducing error. From figure 4.17 and 4.18 we see that the amount of time it takes to reduce the error is dramatically decreased as mu is increased. However, figure 4.19 shows that if mu is increased too much than the amplitude of the error shoots up. This reduces the accuracy of the LMS engine. It is important to choose a value for mu that will give us minimum symbol time for correction, while at the same time giving us reasonable values for our error magnitude. These factors are why we chose mu to be equal to .01, the reasons for which are illustrated in graphs 4.17, 4.18, 4.19.

Figure 4.17

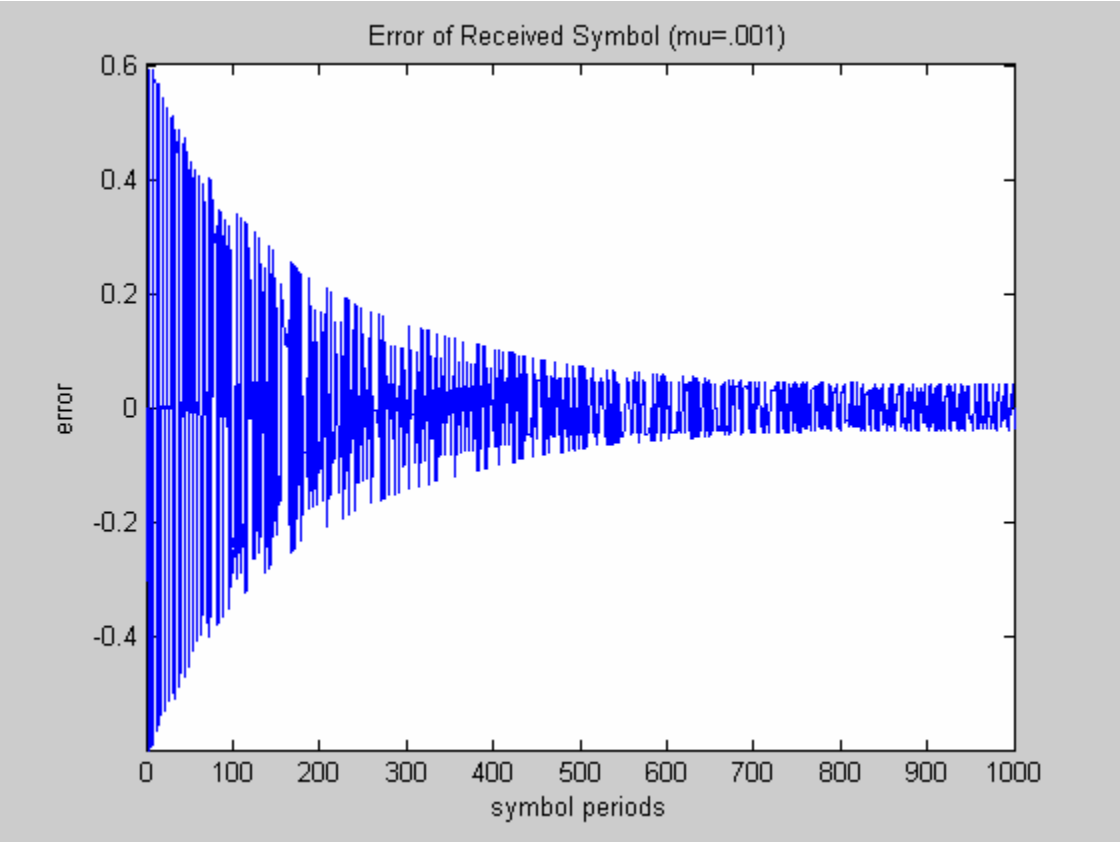


Figure 4.18

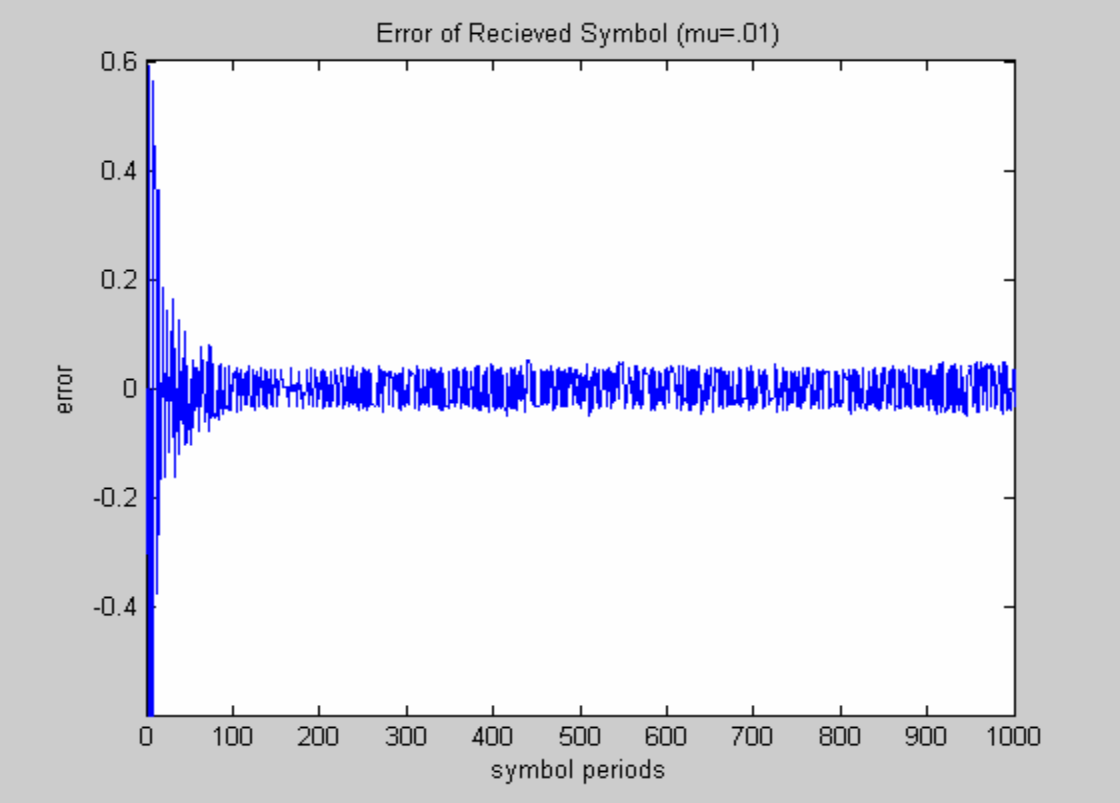
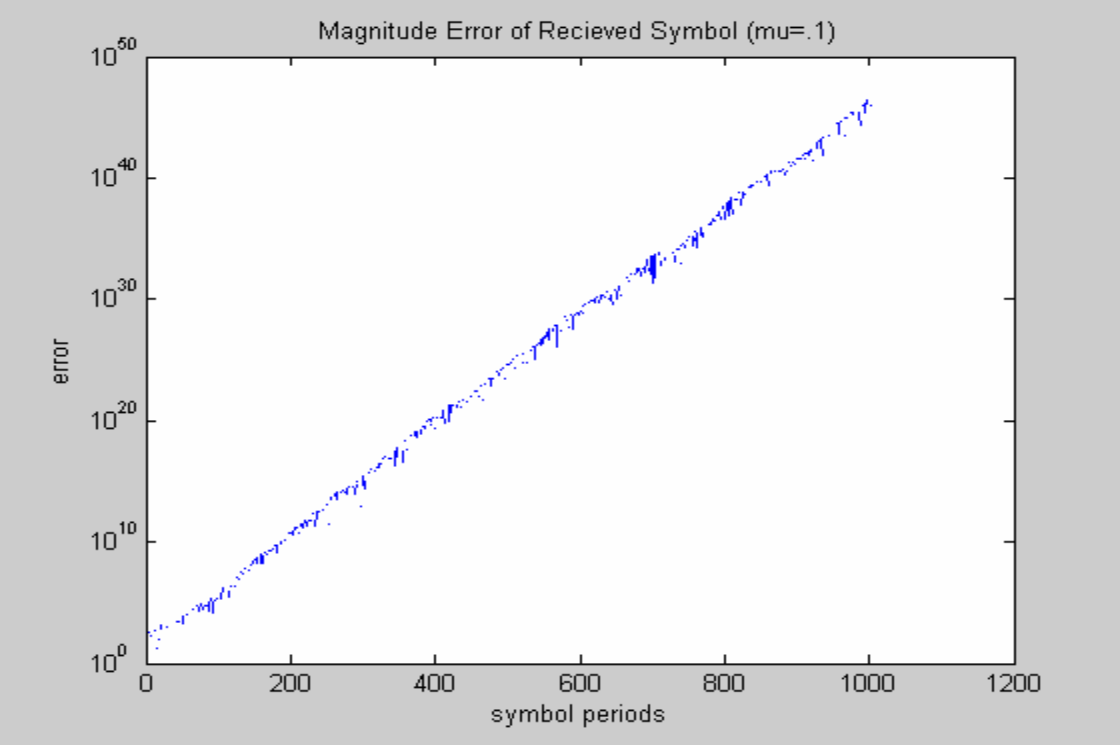
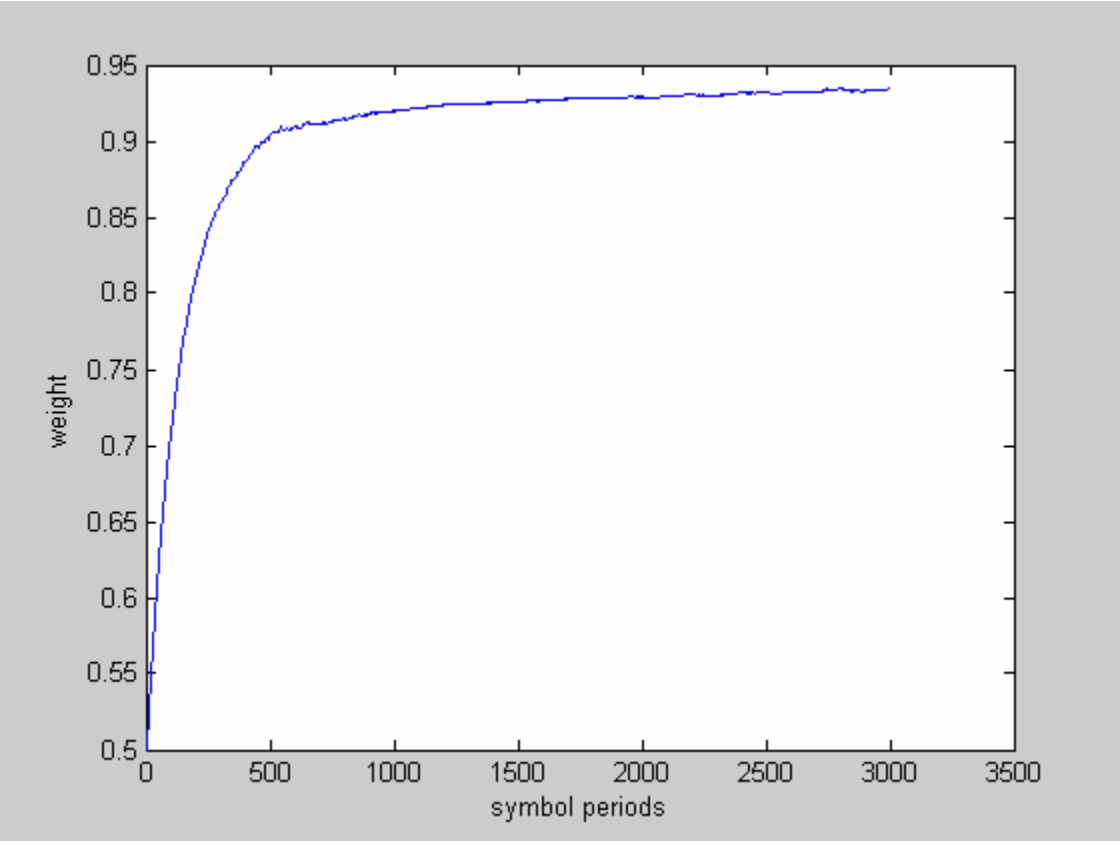


Figure 4.19



The final graph in this section shows the entire equalization process starting with our source and running through the entire design to produce desired outputs for our signals. Figure 4.20 shows a typical weight value progression of taps with good correlation. If the system would not have produced a good progression of taps than we would have known that the signal being received was not the signal for this given receiver. Once again going through the entire simulation in Matlab made the large task of coding the design in VHDL much easier. This was the case because we had to simply transfer the knowledge and process over to VHDL, and this code is included at the end of this section.

Figure 4.20



To be able to decode the symbols that were transmitted, the decision block must compare the additive correlation to an expected correlation value. The following plots show the correlation for both in-phase and quadrature channels for Kasami and Hadamard PN codes. Again note the autocorrelation properties for each code. Although the correlation values differ in each delay of the equalizer their cumulative effect is obvious in the additive result. The symbols that were sent are clearly detectable and are sufficient to reconstruct the bit stream that was transmitted.

Figure 4.21

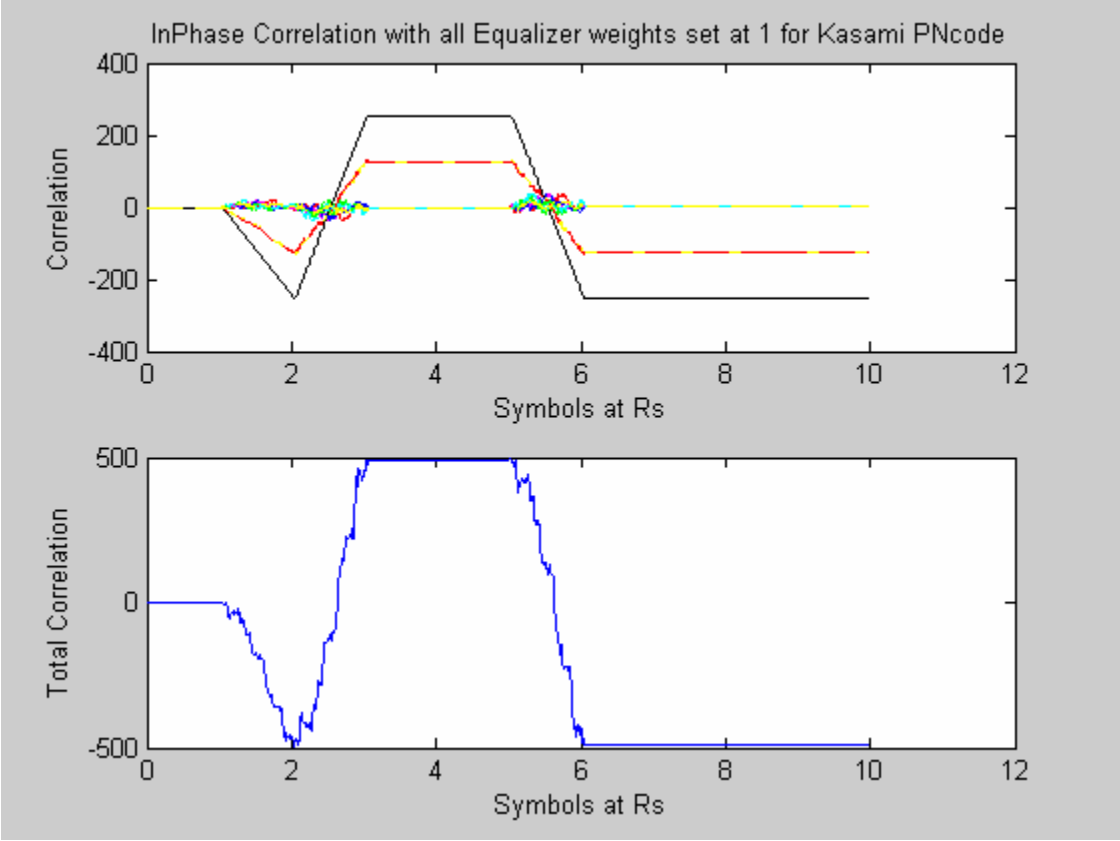


Figure 4.22

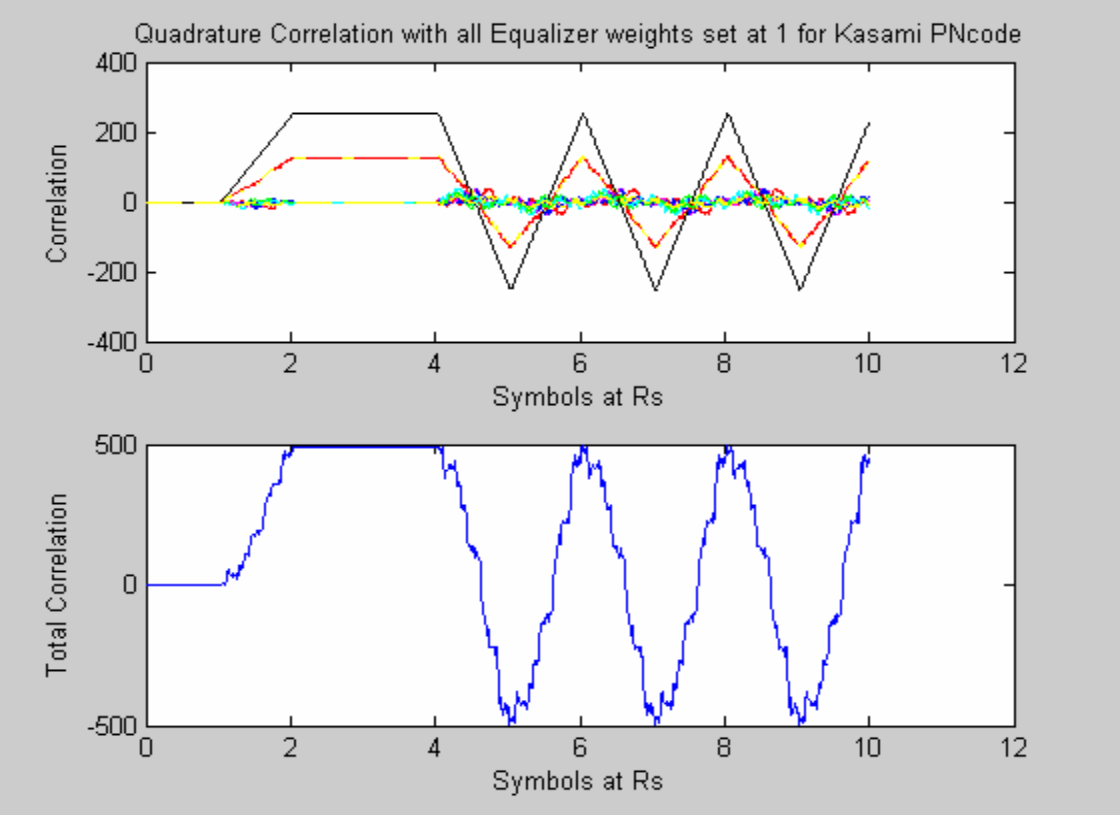


Figure 4.23

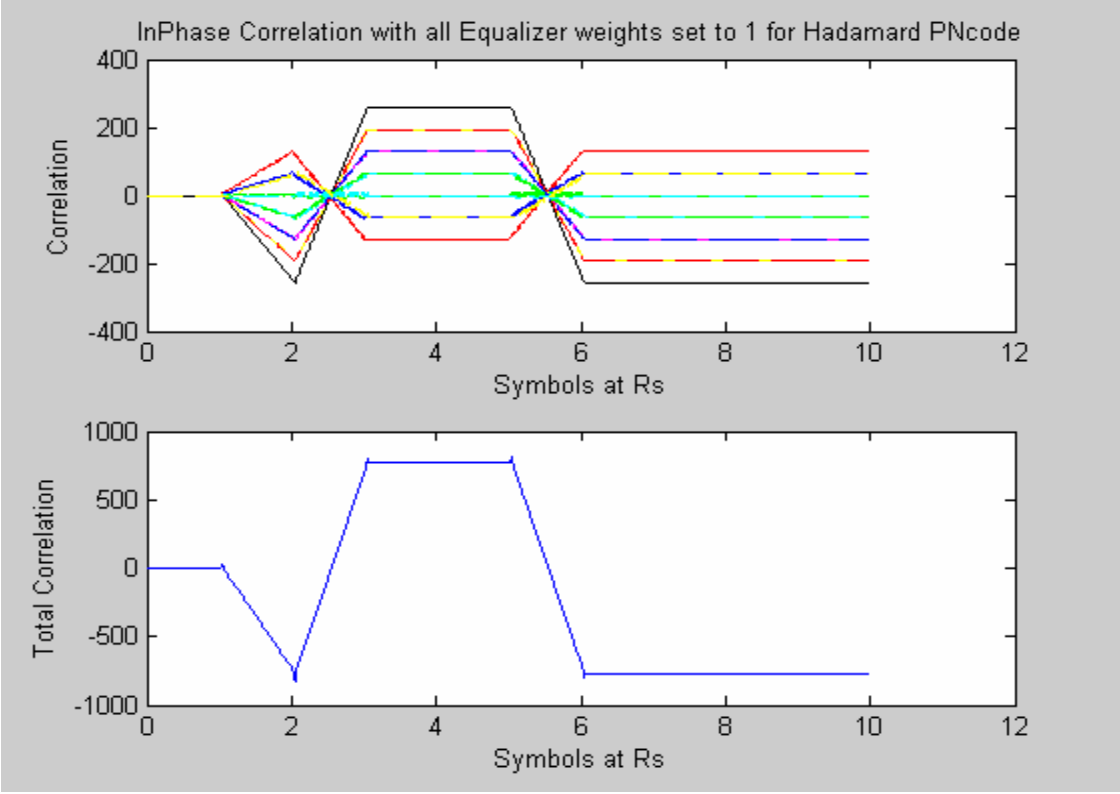
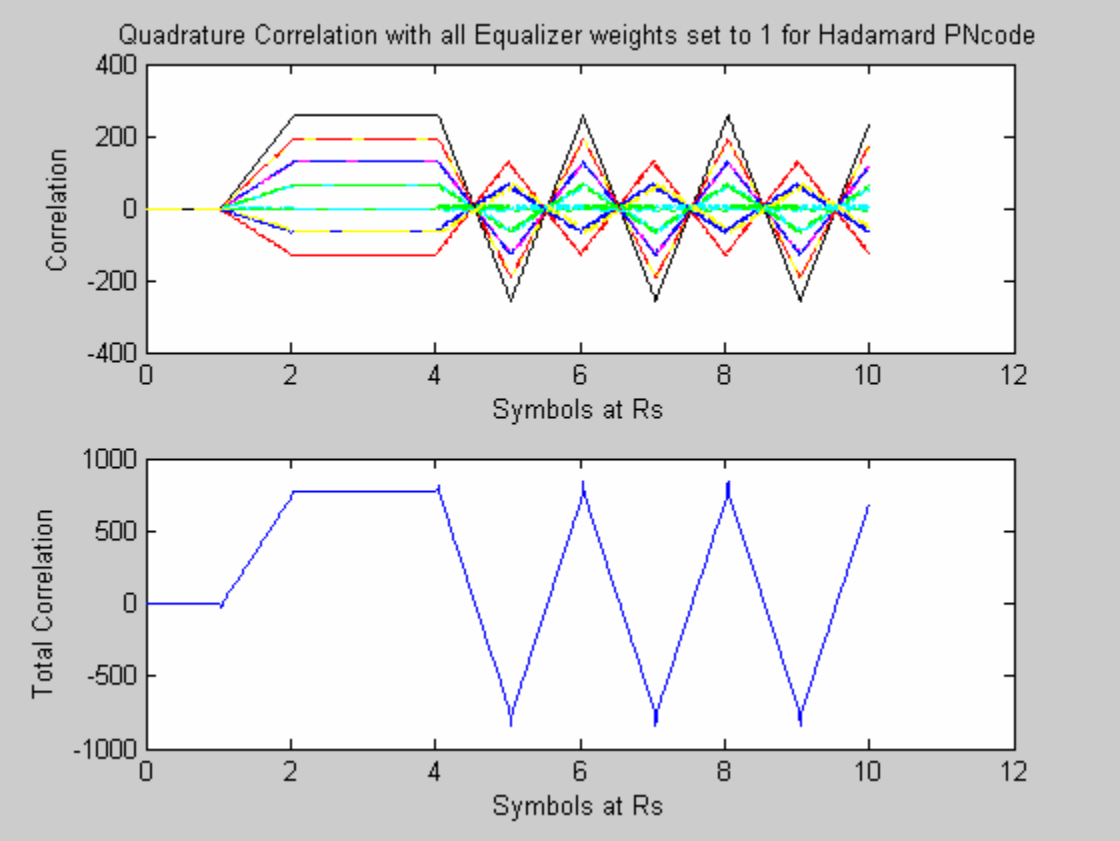


Figure 4.24



4.1. VHDL Coding

BINARY MULTIPLIER

```
-- this is the model for a 8x8 multiplier

library ieee;
use ieee.std_logic_1164.all;

entity Binary_Multiplier is

port( MPLR    :in std_logic_vector (7 downto 0);
      MPCD    :in std_logic_vector (7 downto 0);
      RESULT  :out std_logic_vector (15 downto 0));
end Binary_Multiplier;

architecture Compact of Binary_Multiplier is
```

```
  signal ACarry         :std_logic_vector (7 downto 0);

  type Sum_Type is array (7 downto 0) of
                  std_logic_vector (7 downto 0);

  signal ASum           :Sum_Type;
  signal OPD1,OPD2  :Sum_type;
  signal RES            :std_logic_vector (15 downto 0);

 function RESIZE (A  :in std_logic;
                  Size    :in Natural)
  return std_logic_vector is

  variable RES:         std_logic_vector (Size - 1 downto 0);

 begin
  RES:=(others=>A);
  return (RES);
 end;

 component Generic_Full_Adder
  port(
    A             :in std_logic_vector (7 downto 0);
    B             :in std_logic_vector (7 downto 0);
    Sum           :out std_logic_vector (7 downto 0);
    Cout          :out std_logic);
 end component;

begin

 G2: for K in 1 to 7 generate

   G3: if K=1 generate
         Asum(0)<=RESIZE(MPLR(0),8) and MPCD;
         RESULT(0)<=ASUM(0)(0);
         ACarry(0)<='0';
   end generate;

   OPD2(K)<=ACarry(K-1)&ASum(K-1)(7 downto 1);
   OPD1(K)<=RESIZE(MPLR(K),8) and MPCD;

   GFA:Generic_Full_Adder
         port map(A=>OPD1(K),B=>OPD2(K),Sum=>ASum(K),Cout=>ACarry(K));

   RESULT(K)<=ASum(K)(0);
```

```
  end generate;

  RESULT(15 downto 7)<=
    ACarry(7)&ASum(7)(7 downto 0);

end Compact;
```

MUX

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
PORT(I_data_in_mux:in is array (15 downto 0) of std_logic_vector (7 downto 0);
    Q_data_in_mux     :in is array (15 downto 0) of std_logic_vector (7 downto 0);

    I_data_out_mux    :out std_logic_vector (7 downto 0);
    Q_data_out_mux     :out std_logic_vector (7 downto 0);

    selector               :in std_logic_vector (3 downto 0));
end;

architecture behav of mux is
begin
  process(selector)

   signal I_temp  :std_logic_vector (7 downto 0);
   signal Q_temp  :std_logic_vector (7 downto 0);

   begin
        case selector is

         when "0000" =>  I_temp <=I_data_in_mux(0)(7 downto 0);
                         Q_temp <=Q_data_in_mux(0)(7 downto 0);

         when "0001" =>  I_temp <=I_data_in_mux(1)(7 downto 0);
                         Q_temp <=Q_data_in_mux(1)(7 downto 0);

         when "0010" =>  I_temp <=I_data_in_mux(2)(7 downto 0);
                         Q_temp <=Q_data_in_mux(2)(7 downto 0);

         when "0011" =>  I_temp <=I_data_in_mux(3)(7 downto 0);
                         Q_temp <=Q_data_in_mux(3)(7 downto 0);

         when "0100" =>  I_temp <=I_data_in_mux(4)(7 downto 0);
```

```
                    Q_temp <=Q_data_in_mux(4)(7 downto 0);

          when "0101" =>  I_temp <=I_data_in_mux(5)(7 downto 0);
                    Q_temp <=Q_data_in_mux(5)(7 downto 0);

          when "0110" =>  I_temp <=I_data_in_mux(6)(7 downto 0);
                    Q_temp <=Q_data_in_mux(6)(7 downto 0);

          when "0111" =>  I_temp <=I_data_in_mux(7)(7 downto 0);
                    Q_temp <=Q_data_in_mux(7)(7 downto 0);

          when "1000" =>  I_temp <=I_data_in_mux(8)(7 downto 0);
                    Q_temp <=Q_data_in_mux(8)(7 downto 0);

          when "1001" =>  I_temp <=I_data_in_mux(9)(7 downto 0);
                    Q_temp <=Q_data_in_mux(9)(7 downto 0);

          when "1010" =>  I_temp <=I_data_in_mux(10)(7 downto 0);
                    Q_temp <=Q_data_in_mux(10)(7 downto 0);

          when "1011" =>  I_temp <=I_data_in_mux(11)(7 downto 0);
                    Q_temp <=Q_data_in_mux(11)(7 downto 0);

          when "1100" =>  I_temp <=I_data_in_mux(12)(7 downto 0);
                    Q_temp <=Q_data_in_mux(12)(7 downto 0);

          when "1101" =>  I_temp <=I_data_in_mux(13)(7 downto 0);
                    Q_temp <=Q_data_in_mux(13)(7 downto 0);

          when "1110" =>  I_temp <=I_data_in_mux(14)(7 downto 0);
                    Q_temp <=Q_data_in_mux(14)(7 downto 0);

          when "1111" =>  I_temp <=I_data_in_mux(15)(7 downto 0);
                    Q_temp <=Q_data_in_mux(15)(7 downto 0);

        end case;

        I_data_out_mux <= I_temp;
        Q_data_out_mux <= Q_temp;

  end process;
end behav;
```

COMPLEX MULTIPLY

```
--Algorithm
--(I_data+jQ_data)*(I_weight+jQ_weight)=
--(I_data*I_weight -Q_data*Q_weight)+j(I_data*Q_weight+Q_data*I_weight)

library ieee;
use ieee.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity complex_mult is
PORT(r_clk              :IN std_logic;
    r_Reset_N           :IN std_logic;

    I_Rx_Weight         :IN std_logic_vector(7 downto 0);
    Q_Rx_Weight         :IN std_logic_vector(7 downto 0);

    I_data_in_CM        :IN std_logic_vector(7 downto 0);
    Q_data_in_CM        :IN std_logic_vector(7 downto 0);

    I_data_out_CM       :OUT std_logic_vector(14 downto 0);
    Q_data_out_CM       :OUT std_logic_vector(14 downto 0));
end;

architecture behav_complex_mult of complex_mult is

  component Binary_Multiplier
    port( MPLR:in std_logic_vector (7 downto 0);
          MPCD         :in std_logic_vector (7 downto 0);
          RESULT:out std_logic_vector (15 downto 0));
  end component;

signal  R1      :std_logic_vector (15 downto 0);
signal  R2      :std_logic_vector (15 downto 0);
signal  I1      :std_logic_vector (15 downto 0);
signal  I2      :std_logic_vector (15 downto 0);

begin

M1:     Binary_Multiplier port map
(MPLR=>I_data_in_CM,MPCD=>I_Rx_Weight,RESULT=>R1);--R1=I_data*I_weight
M2:     Binary_Multiplier port map
(MPLR=>Q_data_in_CM,MPCD=>Q_Rx_Weight,RESULT=>R2);--
R2=Q_data*Q_weight
```

```
M3:     Binary_Multiplier port map
(MPLR=>I_data_in_CM,MPCD=>Q_Rx_Weight,RESULT=>I1);--I1=jI_data*Q_weight
M4:     Binary_Multiplier port map
(MPLR=>Q_data_in_CM,MPCD=>I_Rx_Weight,RESULT=>I2);--I2=jQ_data*I_weight

I_data_out_CM<=R1-R2;
Q_data_out_CM<=I1+I2;


end behav_complex_mult;
```

COMPLEX MULTIPLY SUM

```
entity complexmultiply_sum IS
PORT(r_clk              :IN std_logic;
    r_Reset_N           :IN std_logic;

    I_Rx_Weights        :IN std_logic_vector(11 downto 0);
    Q_Rx_Weights        :IN std_logic_vector(11 downto 0);

    I_Delay_bits        :IN std_logic_vector(11 downto 0);
    Q_Delay_bits       :IN std_logic_vector(11 downto 0);

    I_Summed            :OUT std_logic;
    Q_Summed                    :OUT std_logic);
END complexmultiply_sum;

ARCHITECTURE behav of complexmultiply_sum IS

begin
        process(r_clk,r_Reset_N)
```

FULL ADDER

```
-- This is an entity for a 1-bit full_adder

library ieee;
use ieee.std_logic_1164.all;

entity Full_Adder is
Port(X,Y,Cin:in std_logic; S,C:out std_logic);
end Full_Adder;

architecture DataFlow of Full_Adder is
begin
```

```
S<=(X xor Y) xor Cin;
C<=(X and Y) or (X and Cin) or (Y and Cin);

end DataFlow;
```

GENERIC FULL ADDER

```
--This is a 8 bit full Adder

library ieee;
use ieee.std_logic_1164.all;

entity Generic_Full_Adder is
port(A          :in std_logic_vector (7 downto 0);
    B           :in std_logic_vector (7 downto 0);
    Sum         :out std_logic_vector (7 downto 0);
    Cout        :out std_logic);

end Generic_Full_Adder;

architecture CONC of Generic_Full_Adder is

component Full_Adder
  Port(X,Y,Cin:in std_logic; S,C:out std_logic);
end component;

signal CARRY          :std_logic_vector(8 downto 0);

begin

CARRY(0)<='0';
Cout<=CARRY(8);

G1: for K in 0 to 7 generate
 FA:Full_Adder port map (X=>A(K),Y=>B(K),Cin=>CARRY(K),
                         S=>Sum(K),C=>CARRY(K+1));
  end generate;

end CONC;
```

DELAY LINE MULTIPLY

```
library ieee;
use ieee.std_logic_1164.all;

entity delayline_multiply_sum is
port(r_clk              :in std_logic;
```

```
    r_Reset_N           :in std_logic;

    Q_in                :in std_logic_vector(7 downto 0);
    I_in          :in std_logic_vector(7 downto 0);

    I_Rx_Weight_top  :in is array (15 downto 0) of std_logic_vector (7 downto 0);
    Q_Rx_Weight_top    :in is array (15 downto 0) of std_logic_vector (7 downto 0);

    I_out               :out std_logic_vector(19 downto 0);
    Q_out               :out std_logic_vector(19 downto 0);

    Sum_rdy_stb         :out std_logic);

end delayline_multiply_sum;

architecture behav of delayline_multiply_sum is
  component complex_mult
    port(r_clk          :IN std_logic;
     r_Reset_N          :IN std_logic;

    I_Rx_Weight         :IN std_logic_vector(7 downto 0);
    Q_Rx_Weight         :IN std_logic_vector(7 downto 0);

    I_data_in_CM        :IN std_logic_vector(7 downto 0);
    Q_data_in_CM        :IN std_logic_vector(7 downto 0);

    I_data_out_CM       :OUT std_logic_vector(14 downto 0);
    Q_data_out_CM       :OUT std_logic_vector(14 downto 0));
  end component;

  component mux
    port(I_data_in_mux:in is array (15 downto 0) of std_logic_vector (7 downto 0);
     Q_data_in_mux      :in is array (15 downto 0) of std_logic_vector (7 downto 0);

    I_data_out_mux      :out std_logic_vector (7 downto 0);
    Q_data_out_mux      :out std_logic_vector (7 downto 0);

     selector           :in std_logic_vector (3 downto 0));
  end component;

  component delay_line
    port(r_clk          :IN std_logic;
     r_Reset_N          :IN std_logic;
    I_data_in_DL        :IN std_logic_vector (7 downto 0);
    Q_data_in _DL       :IN std_logic_vector (7 downto 0);
```

```
    I_data_out_DL      :OUT is array (15 downto 0) of std_logic_vector (7 downto 0);
    Q_data_out_DL      :OUT is array (15 downto 0) of std_logic_vector (7 downto 0);

  end component;

begin

  variable select_line   :std_logic_vector(3 downto 0) =:"0000";
  signal I_weights       :std_logic_vector(7 downto 0);
  signal  Q_weights      :std_lgoic_vector(7 downto 0);

  CM1:complex_mult
    port
map(r_clk,r_Reset_N,I_weights,Q_weights,I_data_out_mux,Q_data_out_mux,I_out,Q_o
ut);

  MUX1:mux
    port map(I_data_out_DL,Q_data_out_DL,I_data_in_CM,Q_data_in_CM,select_line);

  MUX2:mux
    port map(I_Rx_Weight_top,Q_Rx_Weight_top,I_weights,Q_weights,select_line);

  DL1:delay_line
    port map(r_clk,r_Reset_N,I_in,Q_in,I_data_in_mux,Q_data_in_mux);


  variable I_total,Q_total :integer :=0;

initialize:process(r_Reset_N)

          if(r_Reset_N='0') then
            select_line:="0000"
            I_total,Q_total:=0;

end process;

Summation:process(r_clk'event and r_clk='1')

          variable k        :integer:=15;

          Sum_rdy_stb=>'0';

          SUM_LOOP:while k>0 loop

             I_total:=I_total+integer(I_data_out_CM);
             Q_total:=Q_total+integer(Q_data_out_CM);
```

```
            select_line=>std_logic_vector(3 downto 0)(k+1);

          end loop;

          I_out=>std_logic_vector(19 downto 0)(I_total);
          Q_out=>std_logic_vector(19 downto 0)(I_total);
          Sum_rdy_stb=>'1';

end process;
end behav;
```

DISPREADER

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
-------------------------------------------------------------------------
--                    ENTITY DECLARATION                               --
-------------------------------------------------------------------------
ENTITY desprd IS
PORT(r_clk                : IN  std_logic;
    r_Reset_N             : IN  std_logic;
    r_Rx5p44Stb           : IN  std_logic;
    -------------------------------------------------------------------------
    r_IRxDPSScaled_mc8    : IN  std_logic_vector(7 DOWNTO 0);
    r_QRxDPSScaled_mc8    : IN  std_logic_vector(7 DOWNTO 0);
    -------------------------------------------------------------------------
    r_uP_ForcePNSelNow    : IN  std_logic;
    r_uP_SelFwdSidePNCode_N : IN  std_logic;
    r_uP_ChanPNCodeEnable : IN  std_logic_vector(7 DOWNTO 0);
    -------------------------------------------------------------------------
    r_uP_PNCodeNewWrStb   : IN  std_logic;
    r_uP_PNCodeWord       : IN  std_logic_vector(15 DOWNTO 0);
    r_uP_PNCodeAddr       : IN  std_logic_vector(15 DOWNTO 0);
    -------------------------------------------------------------------------
    r_uP_PNCodeRdBank     : IN  std_logic_vector(1 DOWNTO 0);
    r_uP_PNCodeNewRdStb   : IN  std_logic;
    r_uP_PNCodeReadValue  : OUT std_logic_vector(15 DOWNTO 0);
    -------------------------------------------------------------------------
    r_ScrmRAMRdAddr       : IN  std_logic_vector(10 DOWNTO 0);
    r_ScrmRAMData         : OUT std_logic;
    -------------------------------------------------------------------------
    r_Chan4PN             : OUT std_logic;
    r_IChan_sign_inv      : OUT std_logic;
```

```
    r_QChan_sign_inv       : OUT std_logic;
    ------------------------------------------------------------------------
    r_DesprdSFEpoch        : IN  std_logic;
    r_DesprdSymbStb        : IN  std_logic;
    r_SymbExtractStb       : IN  std_logic;
    r_SymbDataReadyonFall  : OUT std_logic;
    r_IDesprdData          : OUT std_logic_vector(15 DOWNTO 0);
    r_QDesprdData          : OUT std_logic_vector(15 DOWNTO 0));
END desprd;

------------------------------------------------------------------------
--                    ARCHITECTURE DECLARATION                        --
------------------------------------------------------------------------
ARCHITECTURE behav of desprd IS

  CONSTANT PNCodeSideStartAddr : std_logic_vector(15 DOWNTO 0) := X"0640";
  CONSTANT PNCodeNormStartAddr : std_logic_vector(15 DOWNTO 0) := X"0660";

  -- first create a block ram to hold the PN codes

  TYPE PNRAM2048x1Type IS ARRAY (0 TO 2047) OF std_logic;
  SIGNAL r_PNRAM2048x1 : PNRAM2048x1Type;
  ATTRIBUTE syn_ramstyle : string;
  ATTRIBUTE syn_ramstyle OF r_PNRAM2048x1 : SIGNAL IS "no_rw_check";
  SIGNAL r_ScrmRAM2048x1 : PNRAM2048x1Type;
  ATTRIBUTE syn_ramstyle OF r_ScrmRAM2048x1 : SIGNAL IS "no_rw_check";

  SIGNAL r_uP_PNRAMAddr_r : std_logic_vector(10 DOWNTO 0);
  SIGNAL r_uP_PNRAMRdData : std_logic;
  SIGNAL r_PNRAMRdAddr_r  : std_logic_vector(10 DOWNTO 0);
  SIGNAL r_PNRAMRdData    : std_logic;
  ------------------------------------------------------------------------
  SIGNAL r_uP_ScrmRAMAddr   : std_logic_vector(10 DOWNTO 0);
  SIGNAL r_uP_ScrmRAMAddr_r : std_logic_vector(10 DOWNTO 0);
  SIGNAL r_uP_ScrmRAMRdData : std_logic;
  SIGNAL r_ScrmRAMRdAddr_r  : std_logic_vector(10 DOWNTO 0);

  -- now create some Shiftreg
  TYPE ShiftReg7x16Type IS ARRAY (6 DOWNTO 0) OF std_logic_vector(15
DOWNTO 0);
  SIGNAL r_IDesprdShftReg7x16 : ShiftReg7x16Type;
  SIGNAL r_QDesprdShftReg7x16 : ShiftReg7x16Type;

  SIGNAL r_FinalStb  : std_logic;
  TYPE ShiftReg8x16Type IS ARRAY (7 DOWNTO 0) OF std_logic_vector(15
DOWNTO 0);
```

```
  SIGNAL r_ISymbolShftReg8x16 : ShiftReg8x16Type;
  SIGNAL r_QSymbolShftReg8x16 : ShiftReg8x16Type;

  -- despread8chan process signals
  SIGNAL r_PNRAMRdAddr     : std_logic_vector(10 DOWNTO 0);
  SIGNAL r_DesprdStb       : std_logic;
  SIGNAL r_CyCnt           : std_logic_vector(3 DOWNTO 0);
  SIGNAL r_Rx2p72Period    : std_logic;
  SIGNAL r_PNCodeDt_R      : std_logic;
  SIGNAL r_SymbStbPeriod   : std_logic;
  SIGNAL r_PNCnt           : std_logic_vector(2 DOWNTO 0);
  SIGNAL r_SymbStbPeriod_r : std_logic;
  SIGNAL r_IDesprdData_mc2 : std_logic_vector(15 DOWNTO 0);
  SIGNAL r_QDesprdData_mc2 : std_logic_vector(15 DOWNTO 0);
  SIGNAL r_IRxDPSScaled_r  : std_logic_vector(7 DOWNTO 0);
  SIGNAL r_QRxDPSScaled_r  : std_logic_vector(7 DOWNTO 0);

  -- pn ram read write process
  TYPE RAMWrRdStateType IS
(WaitForWrRdStb,DoWrite,GoWrite16,DoSecWrite,DoRead,
                    GoRead16,GetLastBitOfRead);
  SIGNAL r_RAMWrRdState : RAMWrRdStateType;
  attribute syn_encoding                : string;
  attribute syn_encoding of r_RAMWrRdState : signal is "safe";

  SIGNAL r_WriteShiftReg       : std_logic_vector(15 DOWNTO 0);
  SIGNAL r_uP_PNRAMAddr        : std_logic_vector(10 DOWNTO 0);
  SIGNAL r_NumWrites_ms1       : std_logic;
  SIGNAL r_uP_RAMWrStb         : std_logic;
  SIGNAL r_uP_PNCodeReadValue_i : std_logic_vector(15 DOWNTO 0);

BEGIN

  -- PN RAM creation
  PNRAMP: PROCESS (r_clk)
  BEGIN
    IF r_clk'event AND r_clk = '1' THEN
      IF r_uP_RAMWrStb = '1' THEN
        r_PNRAM2048x1(conv_integer('0' & r_uP_PNRAMAddr)) <= r_WriteShiftReg(0);
      END IF;
      r_uP_PNRAMAddr_r <= r_uP_PNRAMAddr;
      r_PNRAMRdAddr_r  <= r_PNRAMRdAddr;
    END IF;
  END PROCESS PNRAMP;
```

```
  r_uP_PNRAMRdData <= r_PNRAM2048x1(conv_integer('0' &
r_uP_PNRAMAddr_r));
  r_PNRAMRdData    <= r_PNRAM2048x1(conv_integer('0' & r_PNRAMRdAddr_r));

  r_uP_ScrmRAMAddr <= r_uP_PNRAMAddr;

  -- Scrm RAM creation
  ScrmRAMP: PROCESS (r_clk)
  BEGIN
    IF r_clk'event AND r_clk = '1' THEN
      IF r_uP_RAMWrStb = '1' THEN
        r_ScrmRAM2048x1(conv_integer('0' & r_uP_ScrmRAMAddr)) <=
r_WriteShiftReg(0);
      END IF;
      r_uP_ScrmRAMAddr_r <= r_uP_ScrmRAMAddr;
      r_ScrmRAMRdAddr_r  <= r_ScrmRAMRdAddr;
    END IF;
  END PROCESS;

  r_uP_ScrmRAMRdData <= r_ScrmRAM2048x1(conv_integer('0' &
r_uP_ScrmRAMAddr_r));
  r_ScrmRAMData      <= r_ScrmRAM2048x1(conv_integer('0' &
r_ScrmRAMRdAddr_r));


-- SLR creation
  SLRDesprdP: PROCESS (r_clk)
  BEGIN
    IF r_clk'event AND r_clk = '1' THEN
      IF r_DesprdStb = '1' THEN
        r_IDesprdShftReg7x16 <= r_IDesprdShftReg7x16(5 DOWNTO 0) &
                                              r_IDesprdData_mc2;
        r_QDesprdShftReg7x16 <= r_QDesprdShftReg7x16(5 DOWNTO 0) &
                                              r_QDesprdData_mc2;
      END IF;
    END IF;
  END PROCESS SLRDesprdP;

  r_FinalStb <= (r_SymbStbPeriod_r AND r_DesprdStb) OR r_SymbExtractStb;

  SLRFinalP: PROCESS (r_clk)
  BEGIN
    IF r_clk'event AND r_clk = '1' THEN
      IF r_FinalStb = '1' THEN
        r_ISymbolShftReg8x16 <= r_ISymbolShftReg8x16(6 DOWNTO 0) &
                                          r_IDesprdShftReg7x16(6);
```

```
    r_QSymbolShftReg8x16 <= r_QSymbolShftReg8x16(6 DOWNTO 0) &
                                        r_QDesprdShftReg7x16(6);
   END IF;
  END IF;
END PROCESS;

r_IDesprdData <= r_ISymbolShftReg8x16(7);
r_QDesprdData <= r_QSymbolShftReg8x16(7);

Despread8ChanP: PROCESS (r_clk, r_Reset_N)
  VARIABLE PNSign_int_v : integer range -1 TO 1;
  VARIABLE ISpreadData_v : std_logic_vector(15 DOWNTO 0);
  VARIABLE QSpreadData_v : std_logic_vector(15 DOWNTO 0);
BEGIN
  IF r_Reset_N = '0' THEN
    r_PNRAMRdAddr     <= (OTHERS => '0');
    r_DesprdStb       <= '0';
    r_CyCnt           <= (OTHERS => '0');
    r_Rx2p72Period    <= '0';
    r_PNCodeDt_R      <= '0';
    r_SymbStbPeriod   <= '0';
    r_PNCnt           <= (OTHERS => '0');
    r_SymbStbPeriod_r <= '0';
    r_IDesprdData_mc2 <= (OTHERS => '0');
    r_QDesprdData_mc2 <= (OTHERS => '0');
    r_IRxDPSScaled_r  <= (OTHERS => '0');
    r_QRxDPSScaled_r  <= (OTHERS => '0');
    r_Chan4PN         <= '0';
    r_IChan_sign_inv  <= '0';
    r_QChan_sign_inv  <= '0';
  ELSIF r_clk'event AND r_clk = '1' THEN
    PNSign_int_v := 0;
    IF r_DesprdSFEpoch = '1' OR r_uP_ForcePNSelNow = '1' THEN
      r_PNRAMRdAddr(10) <= NOT r_uP_SelFwdSidePNCode_N;
    END IF;

    r_DesprdStb <= NOT r_DesprdStb;
    r_CyCnt <= r_CyCnt + 1;
    IF r_Rx5p44Stb = '1' THEN
      r_CyCnt <= (OTHERS => '0');
      r_DesprdStb <= '0';
      r_Rx2p72Period <= NOT r_Rx2p72Period;
      IF r_Rx2p72Period = '1' THEN
        r_PNRAMRdAddr(9 DOWNTO 3) <= r_PNRAMRdAddr(9 DOWNTO 3) + 1;
      END IF;
    END IF;
```

```
IF r_DesprdStb = '1' THEN
  r_PNRAMRdAddr(2 DOWNTO 0) <= r_PNRAMRdAddr(2 DOWNTO 0) + 1;
END IF;
IF r_DesprdSymbStb = '1' THEN
  r_Rx2p72Period <= '0';
  r_PNRAMRdAddr(9 DOWNTO 0) <= (OTHERS => '0');
END IF;
-- reg PN code data from ram
r_PNCodeDt_R <= r_PNRAMRdData;

IF r_DesprdSymbStb = '1' THEN
  r_SymbStbPeriod <= '1';
ELSIF r_Rx5p44Stb = '1' THEN
  r_SymbStbPeriod <= '0';
END IF;

-- now if the channel is enable despread the data
IF r_DesprdStb = '1' THEN
  r_PNCnt <= r_PNCnt + 1;
  IF r_uP_ChanPNCodeEnable(conv_integer('0' & r_PNCnt)) = '1' THEN
    IF r_PNCodeDt_R = '1' THEN
      PNSign_int_v := -1;
    ELSIF r_PNCodeDt_R = '0' THEN
      PNSign_int_v := 1;
    ELSE
      PNSign_int_v := 0;
    END IF;
  ELSE
    PNSign_int_v := 1;
  END IF;

  IF r_PNCnt = "000" THEN
    r_Chan4PN <= r_PNCodeDt_R;
    r_IChan_sign_inv <= NOT r_IRxDPSScaled_r(r_IRxDPSScaled_r'high);
    r_QChan_sign_inv <= NOT r_QRxDPSScaled_r(r_QRxDPSScaled_r'high);
  END IF;

  ISpreadData_v := conv_std_logic_vector(PNSign_int_v *
                                conv_integer(r_IRxDPSScaled_r),16);
  QSpreadData_v := conv_std_logic_vector(PNSign_int_v *
                                conv_integer(r_QRxDPSScaled_r),16);

  r_SymbStbPeriod_r <= r_SymbStbPeriod;

  IF r_SymbStbPeriod_r = '1' THEN
```

```
      r_IDesprdData_mc2 <= ISpreadData_v;
      r_QDesprdData_mc2 <= QSpreadData_v;
     ELSE
      r_IDesprdData_mc2 <= r_IDesprdShftReg7x16(6) + ISpreadData_v;
      r_QDesprdData_mc2 <= r_QDesprdShftReg7x16(6) + QSpreadData_v;
     END IF;
    END IF;

    -- reg despread data so aligned with PN Code Data
    IF r_CyCnt = X"1" THEN
     r_PNCnt <= (OTHERS => '0');
     r_IRxDPSScaled_r <= r_IRxDPSScaled_mc8;
     r_QRxDPSScaled_r <= r_QRxDPSScaled_mc8;
    END IF;
  END IF;
END PROCESS Despread8ChanP;

r_SymbDataReadyonFall <= r_SymbStbPeriod_r;

-- now do the goofy uP PN read and write stuff to simplify life for the uP
PNRAMuPWriteP: PROCESS (r_clk, r_Reset_N)
  VARIABLE PNCodeNorAddr_v : std_logic_vector(15 DOWNTO 0);
  VARIABLE PNRAMAddr_v     : std_logic_vector(3 DOWNTO 0);
BEGIN
  IF r_Reset_N = '0' THEN
    r_RAMWrRdState         <= WaitForWrRdStb;
    r_WriteShiftReg        <= (OTHERS => '0');
    r_uP_PNRAMAddr         <= (OTHERS => '0');
    r_NumWrites_ms1        <= '0';
    r_uP_RAMWrStb          <= '0';
    r_uP_PNCodeReadValue_i <= (OTHERS => '0');
  ELSIF r_clk'event AND r_clk = '1' THEN
    CASE r_RAMWrRdState IS
      WHEN WaitForWrRdStb =>
        IF r_uP_PNCodeNewWrStb = '1' THEN
          r_RAMWrRdState <= DoWrite;
        ELSIF r_uP_PNCodeNewRdStb = '1' THEN
          r_RAMWrRdState <= DoRead;
        END IF;
      WHEN DoWrite =>
        -- calculate the write address
        r_WriteShiftReg <= r_uP_PNCodeWord;
        PNCodeNorAddr_v := r_uP_PNCodeAddr - PNCodeSideStartAddr;
          -- this will have to
          -- be changed when more channels added ie set from chan No. Pins
        -- the word of the PN code goes in bit 9:7 these are bits 4:2 of the
```

```
  -- Normalized Address
  r_uP_PNRAMAddr(9 DOWNTO 7) <= PNCodeNorAddr_v(4 DOWNTO 2);
  IF PNCodeNorAddr_v(8 DOWNTO 6) = "000" THEN
    -- pn code is for chan 4 (0) either upper or lower 0 is upper
    r_uP_PNRAMAddr(10) <= NOT PNCodeNorAddr_v(5);
    r_uP_PNRAMAddr(2 DOWNTO 0) <= "000";
    r_NumWrites_ms1 <= '0';
  ELSE
    PNRAMAddr_v := PNCodeNorAddr_v(8 DOWNTO 5) - 1;
    r_uP_PNRAMAddr(2 DOWNTO 0) <= PNRAMAddr_v(2 DOWNTO 0);
    r_NumWrites_ms1 <= '1';
    r_uP_PNRAMAddr(10) <= '0';
  END IF;

  r_uP_PNRAMAddr(6 DOWNTO 3) <= "0000";
  r_uP_RAMWrStb <= '1';
  r_RAMWrRdState <= GoWrite16;
 WHEN GoWrite16 =>
  r_uP_PNRAMAddr(6 DOWNTO 3) <= r_uP_PNRAMAddr(6 DOWNTO 3) + 1;
  r_WriteShiftReg <= '0' & r_WriteShiftReg(15 DOWNTO 1);
  IF r_uP_PNRAMAddr(6 DOWNTO 3) = X"f" THEN
    r_uP_RAMWrStb <= '0';
    IF r_NumWrites_ms1 = '0' THEN
      r_RAMWrRdState <= WaitForWrRdStb;
    ELSE
      r_NumWrites_ms1 <= '0';
      r_RAMWrRdState <= DoSecWrite;
    END IF;
  END IF;
 WHEN DoSecWrite =>
  r_uP_PNRAMAddr(10) <= '1';
  r_uP_RAMWrStb <= '1';
  r_uP_PNRAMAddr(6 DOWNTO 3) <= "0000";
  r_WriteShiftReg <= r_uP_PNCodeWord;
  r_RAMWrRdState <= GoWrite16;
-- now do a read from ram when requested same funny way
 WHEN DoRead =>
  PNCodeNorAddr_v := r_uP_PNCodeAddr - PNCodeNormStartAddr;
  -- the word of the PN code goes in bit 9:7 these are bits 4:2 of the
  -- Normalized Address
  r_uP_PNRAMAddr(10)        <= r_uP_PNCodeRdBank(0);
  r_uP_PNRAMAddr(9 DOWNTO 7) <= PNCodeNorAddr_v(4 DOWNTO 2);
  r_uP_PNRAMAddr(2 DOWNTO 0) <= PNCodeNorAddr_v(7 DOWNTO 5);
  r_uP_PNRAMAddr(6 DOWNTO 3) <= "0000";
  r_RAMWrRdState <= GoRead16;
 WHEN GoRead16 =>
```

```
      r_uP_PNRAMAddr(6 DOWNTO 3) <= r_uP_PNRAMAddr(6 DOWNTO 3) + 1;
      IF r_uP_PNCodeRdBank(1) = '0' THEN
        r_uP_PNCodeReadValue_i <= r_uP_PNRAMRdData &
                          r_uP_PNCodeReadValue_i(15 DOWNTO 1);
      ELSE
        r_uP_PNCodeReadValue_i <= r_uP_ScrmRAMRdData &
                          r_uP_PNCodeReadValue_i(15 DOWNTO 1);
      END IF;
      IF r_uP_PNRAMAddr(6 DOWNTO 3) = X"f" THEN
        r_RAMWrRdState <= GetLastBitOfRead;
      END IF;
     WHEN GetLastBitOfRead =>
      IF r_uP_PNCodeRdBank(1) = '0' THEN
        r_uP_PNCodeReadValue_i <= r_uP_PNRAMRdData &
                          r_uP_PNCodeReadValue_i(15 DOWNTO 1);
      ELSE
        r_uP_PNCodeReadValue_i <= r_uP_ScrmRAMRdData &
                          r_uP_PNCodeReadValue_i(15 DOWNTO 1);
      END IF;
      r_RAMWrRdState <= WaitForWrRdStb;
     WHEN OTHERS => NULL;
    END CASE;
   END IF;
 END PROCESS PNRAMuPWriteP;

 r_uP_PNCodeReadValue <= r_uP_PNCodeReadValue_i;

END  behav;
```

(IMPORTANT NOTICE: The dispreader code was an already existing code from L3 and as part of the project we needed to extend it to a 16 channel dispread.
We've included the code because we used it in the top level design and extended it their although the code is L3's. All other VHDL code included was what we have written.)

5.0. Final scope of work statement.

To say that this project was or has been educational would be an understatement. Coming up to date on the concepts behind the project, such as spread spectrum and equalization was quite the chore itself. Not to mention understanding the system L3 already had in place and trying to program

upgrades for that system in a language that we had never even used. In retrospect though, that learning process was the most rewarding part of this design experience. It gave us a taste of what a real working environment will be like. Most likely wherever we start our careers at, we will be experiencing a repeated exercise of undertaking projects that involve an understanding of foreign concepts. So this Senior Design has given us a heads up on what to expect.

It has also been beneficial to see how a real work environment functions and how common drastic changes can be. It was rather frustrating to find out 70 percent of the way through our project that the upgrade for L3 had been put on hold. In fact the whole subsidiary of L3 (Primewave) had been done away with for financial reasons. So not only had we lost the support and much needed updates for our design, but our design in essence would never be finished while we were involved. Even though we were able to code our required blocks in VHDL and come to an understanding of their functions and see them be successful with our own weights and PN codes, we never had, and probably never will have, the opportunity to see them used successfully in a real world application. Of course our frustrations do not compare to those of the employees of Primewave who have been working for years on this project and now have been moved to the defense side of L3. Once again this helped to prepare us for the real world and what to expect.

While the majority of our project was successful, not only for educational purposes but also for the actual programming in VHDL (and Matlab), we were

unable to finish everything we started out wanting to do. It would have been most rewarding to see our code integrated with L3's and burned onto the Xilinx chip and then have it tested in a real world environment. One thing not having the access to the necessary hardware to test our design did for us was make our preliminary research more useful. At the beginning of the project we were expecting to use L3's PN codes and weights for simulation. Since we no longer had that luxury we had to implement our coding using our own PN codes. Since we had researched these codes and understood their individual strengths and weaknesses we were able to effectively choose a code that helped us get positive results we would not have otherwise had. This gave us some measure of satisfaction to see that our time was spent working on a successful system even though it was not in the environment it was originally intended for.

In looking back on the project there is definitely some things we would do differently, starting with the actual choosing of the project. In the future it would be nice to start a project that did not require us to be so dependant on L3 for help, especially since L3 was 2 hours away. This made it very difficult to find times when they would be able to help us and give needed support. Also we would have chosen a project that we knew at least a little bit about so we could spend some large amounts of time working on the project, without external help, and see drastic improvements. Probably most important would have been to choose a project where if the company did go under we would still be able to accomplish all of our original goals. There seemed to be many times where we felt way in over our heads and even though help did come eventually it was a

stressful experience. It was a good experience but bad enough to make us weary of wanting to undertake a similar project.

It became apparent that most other students doing senior designs approached it in a much more beneficial way. Either including it with another class where they received credit for both cases, or more importantly as part of an internship where they got paid for their project and on the job training for their project. If we had the chance to pick a project again we would definitely try and take full advantage of either one of the previously mentioned processes.

6.0. COST ESTIMATION:

One of the nicest things about this project is that money is not a big factor. At the beginning of the project L3 was providing the funding for a test board and the hardware needed for the upgrade. After we lost access to their laboratory and there was no longer a board to test the code on this was not an issue, but still just as cheap. The labor for the project is primarily educational (translation = free), so L3 had no expenses for the labor. Since the program was put on hold L3 of course will not receive a finalized working prototype (but will receive our code), which proves the theory that you get what you pay for. The only materials we had to pay for were the printouts for the preliminary and final design reports and the poster printout. These expenses totaled to be about $20.00.

Our only additional cost for the project has been traveling back and forth from Logan to Salt Lake City for updates and information at L3. We did not travel down to L3 as often as we had first anticipated, mainly because we did not have

the opportunity to test the design. Still we went down to Salt Lake a half a dozen times. The average expense of each trip was $15.00 now that gas prices are so high. So between the printouts and the gas for travel our project only cost us a little over $110.00, about $60.00 more than was necessary (fortunately there was no call for psychiatric assistance or the illegal us of drugs).

7.0. PROJECT MANAGEMENT SUMMARY:

7.1. Tasks.

Gaining an understanding of the system and the concepts behind equalization and spread spectrum was the first task that needed to be accomplished. Doing this gave us the ability to create a simplified working version in Matlab that deepened our understanding of the design. The next task was to learn the programming language we would be coding the design in, VHDL. Once completed it was a matter of molding the two learning processes and transferring what we had accomplished in Matlab to VHDL, which involved more than just a simple data transfer. We needed to program each block we had been assigned individually and then compile them into one working system with the help of test benches and other blocks in the design from L3. Originally the biggest obstacle in accomplishing this task was fitting the timing and design specifications. One of the reasons for this is that the code that L3 wrote was simulated in a different version of Model Sim than the version that we have been using. This has created some unique problems in trying to integrate our design with L3's. Also without the complete working L3 design it was impossible to ensure that all the timing and fitting specifications were met.

The biggest disappointment in loosing access to L3's laboratories and hardware was not being able to burn the software into the Xilinx chip and run a hardware test of the design. Although not very probable it would be nice if in the future we had the opportunity to test our design in the PrimeWave lab and see if we could get it to communicate with a base station to verify that our equalization implementation worked in the real world.

The following is an outline of the mentioned tasks, both complete and incomplete, and the schedule we followed to bring the project to its current point.

7.1.1. Solidify learning of VHDL, equalization and spread spectrum.

7.1.2. Write code for six blocks outlined in work breakdown structure that we are responsible for.

7.1.3. Synthesize individual blocks to verify they are working

7.1.4. Synthesize entire design in VHDL to verify blocks are working together correctly.

7.1.5. Burn VHDL code into the Xlinx Spartan 300 (incomplete)

7.1.6. Testing and debugging of complete hardware system at L3 to ensure design is successful (incomplete).

7.1.7. Prepare Final Design Report and Give Final Design Presentation.

The Gant chart below is split up into week intervals (except for April and May), and the tasks are crossed out for their latest possible completion time.

Task/week	DEC2	DEC3	DEC4	JAN1	JAN2	JAN3	JAN4	FEB1	FEB2	FEB3	FEB4	MAR1	MAR2	MAR3	MAR4	APR	MAY
7.1.1	XXX	XXX	XXX	XXX	XXX												
7.1.2				XXX	XXX	XXX	XXX	XXX	XXX	XXX							
7.1.3								XXX	XXX	XXX	XXX						
7.1.4												XXX					
7.1.5												XXX					
7.1.6													XXX	XXX	XXX		
7.1.7																XXX	XXX

7.2. Facilities.

For the entire project we used the computer laboratories, including the Sun laboratories, at USU for our work. Since primewave was shut down we did not have the opportunity of using the facilities down at L3.

7.3. Personnel.

In order for our team to have been able to complete this project we needed to obtain the following skills. Most of the concepts of the design, like spectral spreading and equalization, were foreign to us. We have gained an understanding of these terms through the help of two Professors at USU, Dr. Jacob Gunther and Dr. Tamal Bose. Their help has made it possible for us to move from the conceptual design into the actual design.

Since the entire design has been coded in VHDL we needed to become familiar with this language. Dr. Alan Shaw, also a professor at Utah State

University, gave us some books on the VHDL language that proved to be invaluable resources. Robert Taggart was the team member in charge of the programming design in Matlab. He compiled the block diagrams of all the team members and performed the simulations that gave us the final working results. He was also responsible for our Poster presentation.

Brent Haslem served as the information specialist for the project. He continued to be our primary liaison with L3. He was also responsible for combining all of the team members VHDL code and transferring the entire design from Matlab to VHDL. His main task is to ensure that are design integrates properly with the existing design of L3.

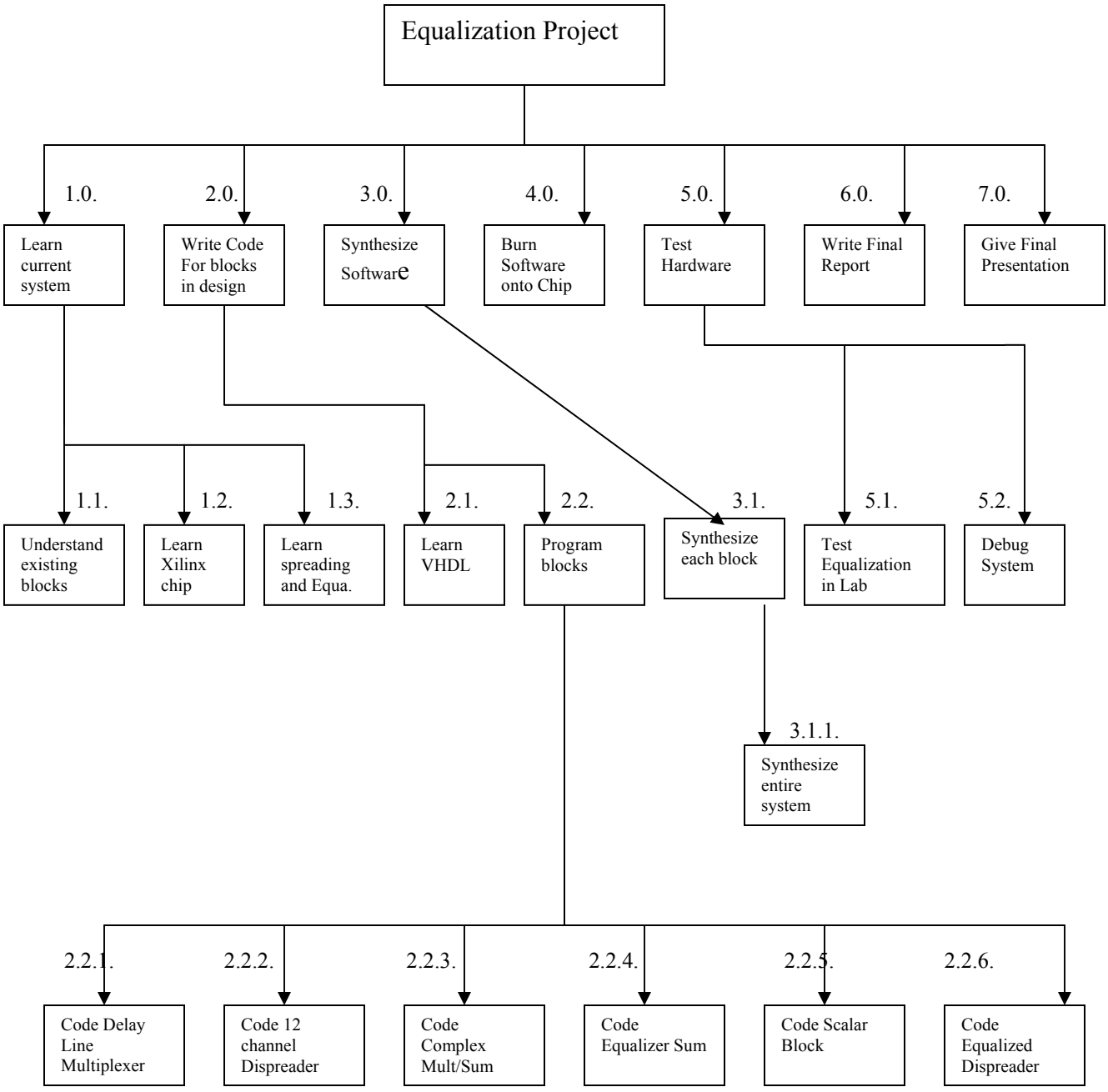
As our coding progresses we will need to continually ensure our design is meeting the specifications for the Xilinx Spartan 300. This involved obtaining from L3 compact disks that provided coding of the Xilinx and the Microblaze. Since the project spanned over such a long period of time it was important that we stay on schedule with tasks leading up to the completion of the project. Brandon Wilson, who is serving as the project manager, is also responsible to make certain all time constraints are met. This includes project presentations and reports, deliverables of the final design, and all other project management responsibilities.

As each team member successfully fulfills his responsibility, we will successfully complete the design. To guarantee this, each member of the team was involved in all three of specific personnel tasks, creating a system of checks

and balances. This also accounted for times when one task needed more work than another one.

The conclusion of the project management section is displayed in the two figures below which give the individuals tasks that will have to be completed to finish this project.

WORK BREAKDOWN STRUCTURE CHART FORM



WORK BREAKDOWN STRUCTURE OUTLINE FORM

Integration of Equalizer into existing Wireless Ethernet system.

1.0. Learn the details of existing system.

- 1.1. Understand functions of existing blocks.
- 1.2. Become familiar with Xilinx Spartan 300 chip.
- 1.3. Learn concepts of Spectral Spreading and Equalization.

2.0. Write Code for blocks we are responsible for.

- 2.1. Learn VHDL coding language.
- 2.2.Program blocks we are responsible for.
  - 2.2.1. Code delay line multiplexer.
  - 2.2.2. Code 12 channel dispreader.
  - 2.2.3. Code Complex Mult/Sum.
  - 2.2.4. Code Equalizer Sum.
  - 2.2.5. Code Scalar block.
  - 2.2.6. Code Equalized Dispreader.

3.0. Synthesize Software.

- 3.1. Synthesize each block individually.
  - 3.1.1. Synthesize entire system once every block is written.

4.0. Burn Software onto Xilinx Spartan 300.

5.0. Test Hardware to ensure system functioning properly.

- 5.1. Use L3 Laboratory to signals to see if equalization works properly.
- 5.2. Debug system until test is successful.

6.0. Write Final Design Report.

7.0. Give Final Design Presentation.

8.0. Conclusion

In this report we have communicated the purpose and process of updating the wireless telephone/Ethernet system L3 has created. Our objective has been to show why equalization is necessary in providing a more reliable and marketable product. Equally important was showing that the solution outlined in this report is not only realistic, but also an optimal choice for this design.

Spectral spreading (especially using QPSK, quaternary phase shift keying), allows L3 to maximize the amount of bandwidth they have at their disposal. Using Orthogonal PNcodes also enables any given transmitter the maximum amount of unique signals it can transmit. Giving receivers the ability to equalize these signals will not only benefit non-line of sight users, but it will also improve signals for receivers that are already line of sight. Sending the signal through a series of delays and going through the process of finding the correlation between these delayed signals gives receivers the most energy efficient signals possible.

Completing this design (a successful hardware test), gives L3 a full functioning wireless Ethernet system capable of providing high-speed service to anyone subscribing, regardless of his or her location. This will not only guarantee customer satisfaction, but it will also guarantee customers for this service. Even though Senior Design is over we have talked to Dan Griffin and are hoping to be able to continue work on the project. Hopefully with his help we can finish fitting our code into L3's within a couple of weeks. This will give us a

little more satisfaction with our project, however until the project is continued at Primewave we will not have access to their labs for any hardware testing.

In concluding even though we didn't have anything 'cool' to show off for our design, it was well worth our time and effort. We have gained what we would consider as valuable experience and wisdom. One of those points of wisdom being, that if a professor or specialist in a field (i.e. Dr. Bose in communications) tells us that a project is probably way over our heads, we will take his opinion seriously.

APPENDIX