802.15.4 Low Intermediate Frequency Radio Receiver

By: Sanjay Avasarala

Senior Project

Electrical Engineering Department

Cal Poly, San Luis Obispo

2012

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my father for all the help he gave me during this project and my mother who motivated me to study and work hard and remain dedicated. I also want to thank my senior project advisor Professor Wayne Pilkington for the support and guidance he gave me these last two quarters.

List of Tables and Figures

TABLES

FIGURES

ABSTRACT

This project involves the design, building, and testing of a low intermediate frequency 802.15.4 receiver that uses an FPGA to perform final demodulation to baseband.

Section 1. INTRODUCTION

This project will demonstrate the successful demodulation of data using a low intermediate frequency (IF) digital radio receiver. A separate modulation circuit will provide the system with BPSK modulated data at 956 MHz and an RF down converter circuit will supply the receiver with the low IF signal which contains both in phase and quadrature components.

The analog section of the receiver includes a complex band pass filter with capability of tuning both the center frequency as well as the filter bandwidth, a programmable gain amplifier (PGA), and an analog to digital converter (ADC). A digital signal processor (DSP) or field programmable gate array (FPGA) provides the final demodulation of the IF data down to base band.

The product in question is an 802.15.4 compatible receiver module that utilizes a DSP/FPGA to perform final demodulation to baseband of a low-IF signal. This type of device is not intended for consumer end markets, but rather for companies who design consumer electronics. The 802.15.4 standard is intended for low power and low data rate applications such as interfacing house hold appliances to a central node and other daily, short-range wireless applications. The module is only a receiver, and thus can either be used in conjunction with a transmitter to form a transceiver, or as a stand-alone module in applications that call for only receiving data. Because the module uses a DSP and has several features described in the specifications document, the receiver is highly customizable and immune to certain levels of interference from noise and adjacent channels. The module also does not contain any clock extraction devices for synchronization since that is beyond the scope of this project.

Section 2. BACKGROUND TECHNOLOGY REVIEW

Note that this section offers the technology background at the highest possible system level. The theory of each sub-system operations is presented in their respective sections.

This project entails the design and creation of a low IF digital radio receiver. Currently, most receivers use three common system architectures: Direct Conversion or Zero IF (ZIF), low IF or high IF. Direct quantization and signal processing at RF frequencies is not practical and thus not used. The following describes the relative advantages and disadvantages of each architecture.

(Direct Conversion, or Zero IF (ZIF))

Advantages

A. *Circuit simplicity*

Since direct conversion avoids the use of an IF, external IF filtering circuitry is not required. Thus, a baseband circuit has a relatively smaller area and a lower cost.

Disadvantages

A. *LO Leakage*

It is possible for the LO signal to leak into the input to the RF down converter (which is as simple as just a mixer. See Fig 1.1 below).

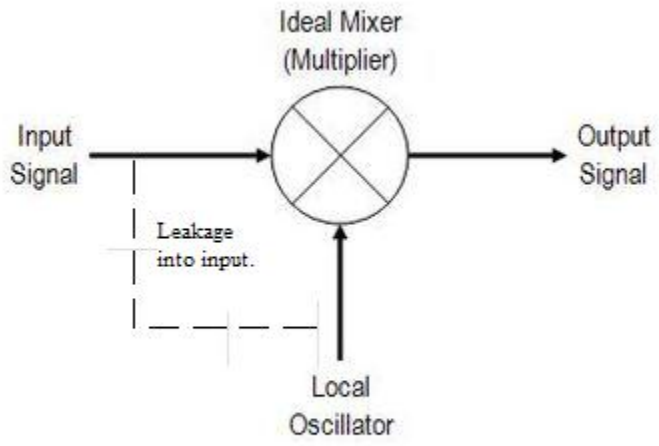

Figure 1.1 – Leakage in a mixer.

Because the LO leaks into the input of the mixer, a DC value is present at the output (two sinusoidal waves multiplied together produce a DC term). If a high gain amplifier is present after the RF down conversion stage, it will rail with the DC offset term present. This can happen for both static self mixing and dynamic self mixing. AC coupling cannot solve this problem due to

significant levels of low frequency and DC content associated with the message signal. It is possible to design filters that account for this problem, although to do so is rather difficult.

B. Noise

A type of noise that is inversely proportional to frequency (known as 1/f noise) will be present in circuits that use active devices. A receiver that processes a signal at baseband is more susceptible to this type of noise.

High IF

Advantages

A. DC offset nulling

A high IF system can get rid of the DC offset problem which the baseband scheme faces by using AC coupling.

B. Image Reject

Unless proper filtering is performed at high frequencies, an image signal will be present along with the wanted signal. The image signal is any spectral power located at a frequency deviation which is the same as the wanted signal away from the local oscillator. However, the image is located on the opposite side of the carrier relative to the wanted signal. Image rejection techniques employed at high IF (using surface acoustic wave or SAW filters) is highly effective and thus high IF is used where high image rejection is needed.

Disadvantages

A. Additional circuitry

Circuits that use a high IF require two phase locked loop (PLL) circuits to convert the signal to a low enough frequency for analog to digital conversion. There are digital signal processors (DSP) available that can handle the high frequencies of high IF (typically in the hundreds of MHz), however the amount of power and processing strength needed for this will increase the cost and current consumption. The additional circuitry needed for the high IF circuit will also make the receiver more expensive.

B. *Bulky expensive components*

As mentioned before, a SAW filter is required for filtering. A SAW filter is a bulky component that is impossible to integrate into a chip. This is the single biggest drawback to this otherwise excellent architecture.

Low IF

Advantages

A. DC offset problem doesn't exist

A low IF system can ignore DC offsets in the devices by simply AC coupling the stages.

B. Circuit simplicity

As opposed to the high IF schemes, a low IF system only requires one PLL to down convert the RF signal to a low IF. The circuit requires no additional frequency conversion since a DSP/FPGA can easily handle the frequencies associated with low IF (hundreds of KHz to ones or tens of MHz).

Disadvantages

A. *Image reject*

Image rejection at a low IF is substantially harder to realize than at a high IF. It is possible, however, to use complex bandpass filters for image rejection. Amplitude and phase matching of components ultimately places a practical limit on image rejection.

B. *Data rate constraints*

Another disadvantage is that the system is most suited for low data rate (low bandwidth) systems. High bandwidth systems such as Wi-Fi are not feasible with a low IF architecture, however, low data rate systems such as Zigbee and Bluetooth are.

Digital System Advantages

A digital signal processor (DSP) will perform the final demodulation of the signal to baseband data. Using a DSP and mixed signal processing carries several advantages over a pure analog system.

A. *Filtering*

Although the radio filters the signal before A/D conversion is employed (see Figure 3), higher order filters can be realized more easily in the digital realm as opposed to analog. The only limitations to this are processing speed and memory space. One must ensure that the signal strength is optimized for quantization since signal to noise ratio degrades for lower amplitudes due to quantization error.

B. *Customizable*

As opposed to an analog system in which individual hardware component values are changed should any parameter change happen, a digital system allows for easy customization through the use of coding.

C. Improved image rejection

If the in phase and quadrature components are maintained through the receive chain, one can use a DSP to improve the image rejection quality. Image rejection relies on matched amplitude and phases in circuitry and a DSP can account for variations in the aforementioned.

Section 3. DESIGN REQUIREMENTS AND SPECIFICATIONS

Note that the following table is derived from the EE 463 Senior Project preliminary report. The table contains both marketing requirements and engineering specifications. The following list provides a brief description of what the module must accomplish:

- Properly demodulate BPSK modulated data at 250 kbps
- Provide image rejection
- Provide a PGA
- Output data to a computer for easy visibility

Marketing Requirements	Engineering Specifications	Justification
2,3,4	Frequency of operation: 950 – 956 MHz, capable of 250Kbps data rate. Receiver must use BPSK modulation with a channel spacing of 2MHz.	Must conform to the Zigbee standard defined frequency allocation and bit rate parameters. These specifications are taken directly from the 802.15.4 standard. Note that spread spectrum techniques will NOT be used in this receiver. [5]
1	Use a low IF of half the channel spacing of 2MHz or 1 MHz.	For low data rate systems such as the Zigbee standard, a low IF is desirable. A low IF architecture typically costs less than other types. 1 MHz is used such that the adjacent channel is the only image present at IF. Any other IF used over 1 MHz will contain alternate channel images which will be substantially harder to reject.
4	Adjacent and alternate channel rejections of 20 and 50 dB respectively.	The adjacent and alternate channels are sometimes larger than the wanted signal. Therefore these have to be rejected to avoid interference (known as image rejection particularly for low IF architectures). These particular values are more than the specified values in the 802.15.4 standard .

4	ADC resolution and sampling rate: 4 bits and a sampling frequency of greater than the Nyquist frequency of twice the bandwidth (4 MHz). Therefore the sampling rate is greater than 4 MHz.	The ADC resolution is determined by the SNR required by the DSP to demodulate the signal to a BER of 1E-4. We assume 20 dB for this so that 4 bits is adequate (4x6+1.78). The preceding equation is a rule of thumb equation that relates signal to noise ratio with a given bit resolution. Excess bits will serve as headroom or margin for the AGC. The sampling rate is determined by the IF frequency and the signal BW.
4	Use an appropriate filter to achieve the channel selectivity without having the group delay distortion affect the BER of the receiver.	For a distortion less system, a circuit should have a constant or small group delay variation in the passband.
4	Use an automatic gain control system: Gain control range of 50 dB	Near/far problems and multipath will cause the signal strength to vary. The sensitivity of BPSK of 250Kbps =-110dBm at RF. Assuming 30dB gain in RF, the sensitivity at IF will be -80dBm. A maximum IF signal corresponds to -20dBm at RF or +10 dBm at IF. For this variation of the signal the level of the signal at the ADC input should be roughly constant to maximize the ADC capabilities.
2,3	User interface	The user must physically view the demodulated data stream to verify proper transmission. MATLAB or LabVIEW can interface with the receiver and display the received data. A Logic Analyzer can also export data to an excel spreadsheet for analysis.
~~2,3~~	~~Tunable IF bandwidth.~~ **NOTE: After some research, it was determined that a tunable image reject filter was beyond the scope of this project and is therefore not going to be implemented.**	~~To accommodate for different bandwidths associated with different data rates, a tunable filter is required. The circuit contains this feature for a proof of concept.~~
4	A bit error rate (BER) of 1E-4 at sensitivity. This in turn corresponds to a packet error rate of 1% for 250 Kbps for BPSK.	This is specified in IEEE 802.15.4 [5]

1	Research and development costs.	The total cost for R&D should not exceed $450.00
5	PCB on which circuit is built should not exceed 60 square inches or use more than 2 layers.	The PCB on which this circuit is built will meet these specs to conform to 4PCB.com's academic board manufacturing option. Evaluation kits for the DSP, RF and TX circuitry will be sourced from manufacturers.
Marketing Requirements 1. Affordable 2. Easy to use 3. Customizable 4. Seamless data transfer 5. Compact		

Table 3.1 – System specifications and requirements

It must be noted at this point that, although the module is meant to be a low IF receiver, a transmitter and a down converter will need to be constructed as well to provide the BPSK low IF modulated data.

Section 4. DESIGN ALTERNATIVES

IF Architecture

The only other alternatives for a receiver architecture include the aforementioned high IF and zero IF architectures. The disadvantages and advantages of each were discussed previously and it was determined that a low IF architecture would be the most feasible for this project given the allotted time and resources.

Analog vs. Digital

The alternative to using DSP/FPGA for final demodulation would obviously be using analog demodulation. However, this requires the extensive use of analog filters and other sub-system components including hard limiters and additional down converters that complicate the project and add to the overall cost. A DSP/FPGA can perform final demodulation using simple coding techniques and is highly customizable and easy to change.

DSP vs. µProcessor vs. FPGA

The alternative to using a DSP would be using a microprocessor to perform demodulation to baseband. This technique will only work if the microprocessor has sufficient memory and processing capabilities to sample and demodulate a signal (1 MHz for the project). The microprocessor needs to be able to buffer many data samples (to be determined) and possibly generate its own pseudo random binary sequence to avoid the use of two microprocessors which would require a digital PLL. The last alternative is to use an FPGA and VHDL to carry out final demodulation. The advantage of using this method is that one has direct control over any of the clocks involved in the system and can therefore alter timing constraints accordingly and with precision.

PCB vs. Bread boarding

Although bread boarding the system may offer more customizability and be less prone to error since any mistakes can easily be made, the system has enough components to the point where bread boarding the system may not be feasible. Creating a layout and designing the circuit on a PCB is much more compact and clean (especially for RF circuitry).

Section 5. PROJECT DESIGN

A high level block diagram of the system is shown below in Figure 5.1. Table 5.1 describes the function of each block.

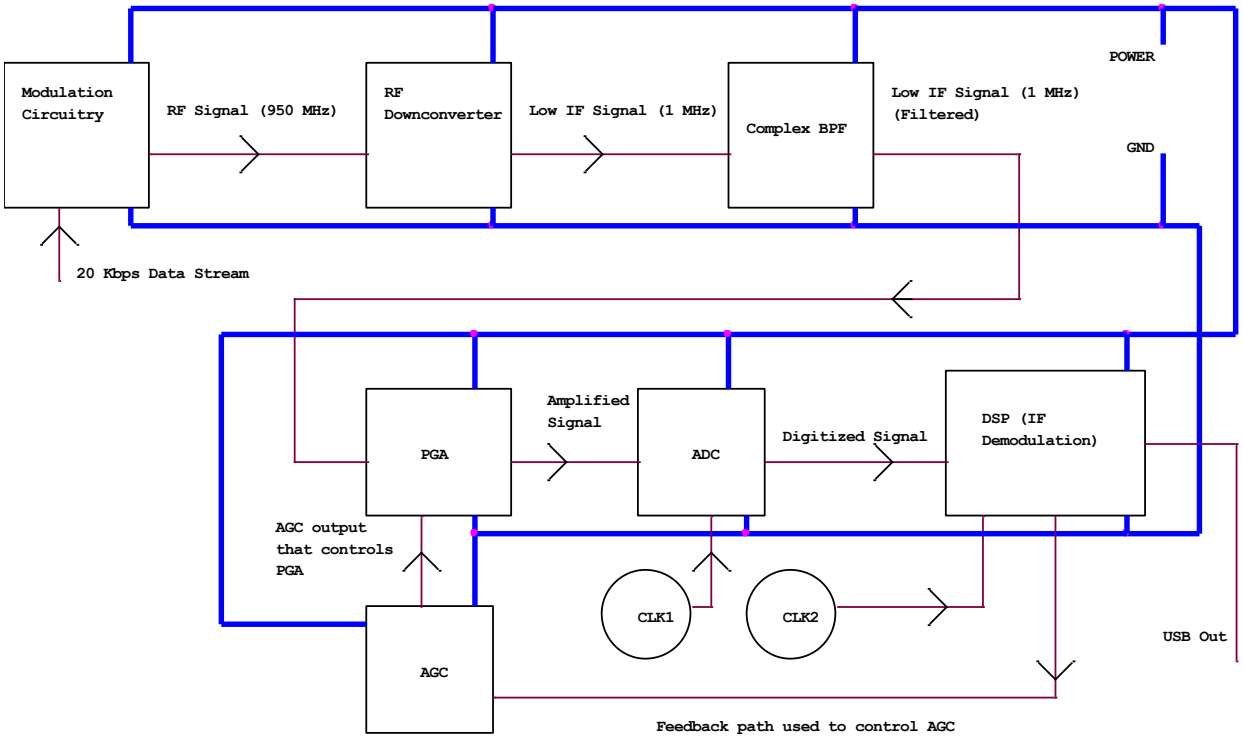

Figure 5.1. – Block diagram of the entire system.

BLOCK	*INPUT*	*OUTPUT*	*FUNCTION*
Modulation Circuitry	250 Kbps digital data stream.	Modulated 956 MHz BPSK RF signal.	The modulation circuitry is what will provide the BPSK modulated data signal to the radio receiver at 956 MHz.
RF Down conversion circuitry	RF 900 MHz signal	Low IF Signal (1 MHz)	The RF down converter will convert the modulated frequency to the low intermediate frequency for signal processing.
BPF (Band pass image reject filter)	IF Signal	Filtered IF Signal	The band pass filter will only allow the wanted signal in the pass band and attenuate all other signals. Note that this filter is complex I/Q

PGA (Programmable Gain Amplifier)	Filtered IF Signal	Filtered IF Signal with gain	The PGA is needed for the AGC.
ADC (Analog to Digital converter)	Filtered IF Signal with gain	Quantized output	The ADC will convert the analog signal to a digital value which will be processed by the DSP/FPGA. The ADC resolution will depend on the SNR requirements of the 802.15.4 system.
CLK1	No Input	5 MHz	The sampling rate of the ADC is determined by this clock.
CLK2	No Input	50 MHz	Internally generated DSP/FPGA clock.
AGC (Automatic gain control)	DSP output signal (logic bus)	Logic bus signal	The AGC will control the PGA. The AGC is controlled by the DSP/FPGA.
DSP (Digital signal processor. Can be replaced by an FPGA)	Quantized IF signal	DSP output logic bus. USB (tentative). Logic analyzer (tentative).	The DSP will perform the final demodulation to baseband. The DSP will have a user interface (a computer) such that the user can view the demodulated bit stream for test purposes. An FPGA can take the place of a DSP.

Table 5.1. Functional description of each sub system block

Sub-Section 1. Image Reject Filter Analysis and Design

One of the biggest problems associated with a low IF architecture is the presence of an image at the IF. An image is present due to the mathematical properties of frequency shifting when down converting. Although the image signal will be present in the negative frequencies, in all practical applications these frequencies are 'folded over' such that the image signal will indeed be present at the IF. Therefore, proper filtering must be performed to remove the image at the 'negative' frequencies. This can be done by positively shifting the magnitude response of a low pass filter such that no negative frequencies will be present in the pass band (the magnitude response of a LPF is symmetric about the y axis and therefore negative frequencies will be in the pass band). A shift in the frequency domain corresponds to multiplication by a complex number in the time domain. Hence, the type of filter used can be called a complex bandpass filter. Analysis of an IQ complex bandpass filter is shown below.

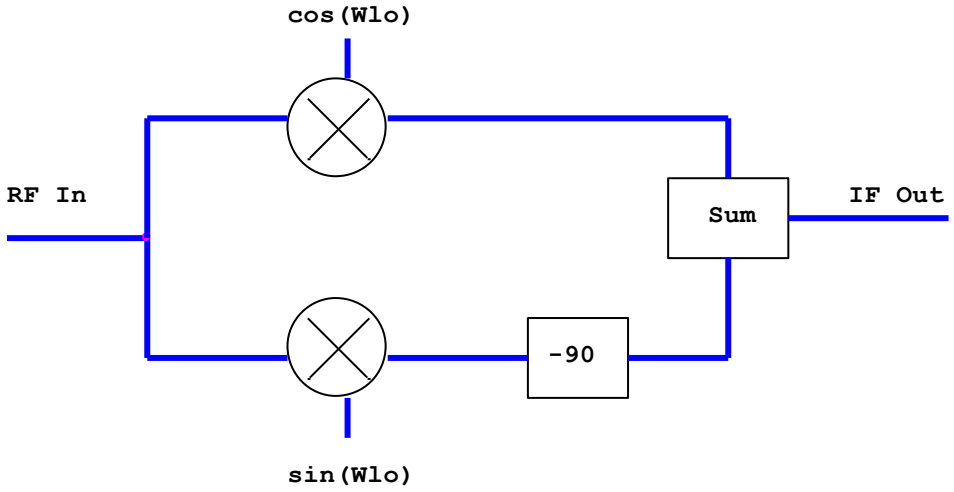

Figure 5.1.1. – Simplified block diagram of an image reject system.

First, we shall observe how an image signal is rejected using mathematical analysis. The RF Input contains two signals of interest for this analysis: the wanted signal located at ω_c and the image signal located at $\omega_{IM.}$ Note that for a ω_{LO} that is greater than the wanted signal:

$$\omega_{LO} - \omega_C = \omega_{IF}$$

and

$$\omega_{LO} + \omega_{IF} = \omega_{IM}$$

The frequencies mentioned are all at RF frequencies except for $\omega_{IF.}$ Without loss of generality, we can assume that the image and wanted signals are simple sinusoids with different amplitudes. Thus,

$$Message\ Signal = Acos(\omega_C t)$$
$$Image\ Signal = Bcos(\omega_{IM} t)$$

Using an IQ down-converter, the I channel yields (after being mixed by $\cos(\omega_{LO})$):

$$cos(\omega_{LO}t) * [Acos(\omega_C t) + Bcos(\omega_{IM}t)] =$$

$$A\left(cos\big((\omega_{LO} - \omega_C)t\big) + cos\big((\omega_{LO} + \omega_C)t\big)\right) + B\left(cos\big((\omega_{LO} - \omega_{IM})t\big) + cos\big((\omega_{LO} + \omega_{IM})t\big)\right)$$

and neglecting the higher RF frequencies due to low pass filtering, we arrive at:

$$Acos\big((\omega_{LO} - \omega_C)t\big) + Bcos\big((\omega_{LO} - \omega_{IM})t\big) =$$

$$Acos(\omega_{IF}t) + Bcos(-\omega_{IF}t)$$

The Q channel yields (after being mixed by $\sin(\omega_{LO})$):

$$sin(\omega_{LO}t)[Acos(\omega_C t) + Bcos(\omega_{IM}t)] =$$

$$A\left(sin\big((\omega_{LO} + \omega_C)t\big) + sin\big((\omega_{LO} - \omega_C)t\big)\right) + B\left(sin\big((\omega_{LO} + \omega_{IM})t\big) + sin\big((\omega_{LO} - \omega_{IM})t\big)\right)$$

and neglecting the high RF frequencies, the input to the phase shifter is:

$$Asin\big((\omega_{LO} - \omega_C)t\big) + Bsin\big((\omega_{LO} - \omega_{IM})t\big)$$

Note that the divide by 2 that occurs due to trigonometric identities is left out for simplicity. Now, since $\omega_{LO} - \omega_{IM}$ is a negative frequency, or $-\omega_{IF,}$ we can rewrite the above expression as:

$$Asin\big((\omega_{LO} - \omega_C)t\big) - Bsin\big((\omega_{IM} - \omega_{LO})t\big) =$$

$$Asin(\omega_{IF}t) - Bsin(\omega_{IF}t)$$

Due to the 90 phase shift in the Q channel, the input to the summer is:

$$Asin\left(\omega_{IF}t + \frac{\pi}{2}\right) - Bsin\left(\omega_{IF}t + \frac{\pi}{2}\right) =$$

$$Acos(\omega_{IF}t) - Bcos(\omega_{IF}t)$$

The output of the summer is:

$$Acos(\omega_{IF}t) + Bcos(\omega_{IF}t) + Acos(\omega_{IF}t) - Bcos(\omega_{IF}t) =$$

$$Acos(\omega_{IF}t)$$

Observe that the image signal, which had an amplitude of B is now completely gone. This analysis also applies to modulated signals and images at these frequencies.

This type of image reject system can be realized using two cross coupled low pass filters. Note that the 90 degrees phase shift of the I channel is simply the Q channel. In other words, the Q channel shifts the I channel by the required 90 degrees, and feeding the Q channel into a filter along with the I channel will result in the desired output. Analysis of this filter is shown below.

We first start with the simple low pass transfer function:

$$H(w) = \frac{1}{1 + \frac{j\omega}{\omega_o}}$$

The magnitude response of this TF resembles:

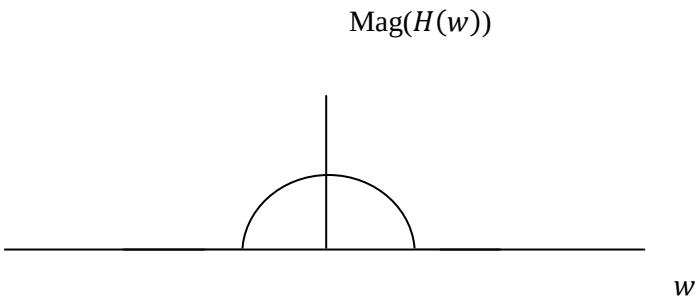

Figure 5.1.2. – Magnitude response of a simple rudimentary low pass filter

We saw how, in the previous analysis, the image signal was located in the negative frequency range. Thus, to eliminate any 'negative' frequencies, we can shift the magnitude response to the right. This will result in a new transfer function:

$$H(w) = \frac{1}{1 + \frac{j(\omega - \omega_s)}{\omega_o}}$$

where ω_s is the shifted frequency. Applying this transfer function to only the I channel (for now) and after cross multiplying, we arrive at:

$$I_{in} = I_{out} + \frac{j\omega}{\omega_o} I_{out} - \frac{j\omega_s}{\omega_o} I_{out}$$

where I_{out} is the output of the LPF. Note that the coefficient of the last I_{out} term is a 90 degree shifted scalar multiple of I_{out}. From before, we know that the Q channel is the I channel positively shifted by 90 degrees, or j*I. Therefore, any signal processing performed on the Q channel that is identical to the I channel will result in an output on the Q channel that is still j*I. Therefore it is safe to represent I_{out} as $-jQ_{out}$. Replacing this in the above equation results in:

$$I_{in} = I_{out} + \frac{j\omega}{\omega_o} I_{out} - \frac{\omega_s}{\omega_o} Q_{out}$$

Rearranging terms results in:

$$I_{out} = \frac{I_{in} + \frac{\omega_s}{\omega_o} Q_{out}}{1 + \frac{j\omega}{\omega_o}}$$

This transfer function can be realized using the following block diagram (the Q channel gain is left out for simplicity):

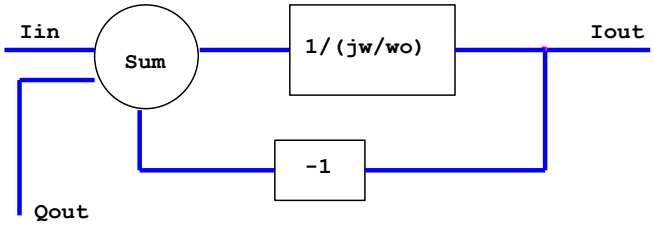

Figure 5.1.3. – Block diagram of an image reject transfer function.

This diagram is essentially an integrator with a simple negative feedback loop. One can implement this block diagram using the following circuit.

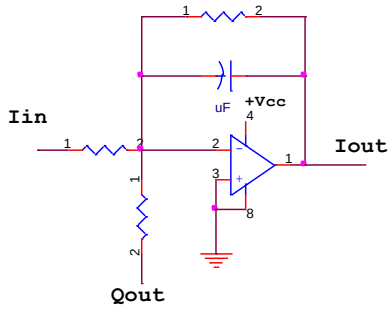

Figure 5.1.4. – Practical realization of a simple image reject filter.

Since the entire system must be symmetric to maintain properly matched I and Q outputs, the system is cross coupled as seen in Figure 5.1.5. We can now derive what the exact transfer function of the system is and show how it resembles the shifted frequency transfer function.

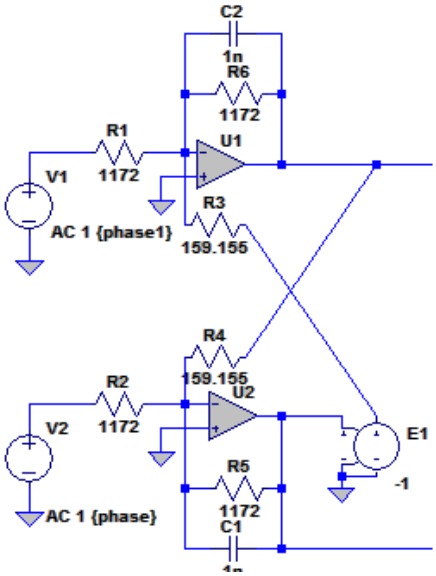

Figure 5.1.5. – A more detailed look at a simple image reject filter.

We can use nodal analysis for the I channel op-amp and arrive at the equation:

$$\frac{I_{in}}{R_1} - \frac{Q_{out}}{R_3} = -I_{out}(\frac{1}{R_6} + sC_2)$$

The negative term for Q_{out} is present due to the inverter (E1) at the output of the filter on the Q channel. This inverter is here due to mathematical reasons. If it were not present, the passband would be shifted negatively as opposed to positively. Replacing s with $j\omega$ and solving for I_{out}yields:

$$I_{out} = \frac{(-\frac{R_6 I_{in}}{R_1} + \frac{R_6 Q_{out}}{R_3})}{1 + \frac{j\omega}{\omega_o}}$$

where

$$\omega_o = \frac{1}{R_6 C_2}$$

This transfer function is almost identical to the shifted frequency transfer function we were trying to achieve with the exception of a scalar coefficient for both I_{in} and Q_{out}. The polarity does not matter since it will result in a 180 degrees phase shift for both channels due to symmetry and thus not affect the frequency response. The Q_{out} coefficient must be set to equal $\frac{\omega_s}{\omega_o}$ derived earlier. The values present in the schematic above are for a shift frequency of 1 MHz with the low pass filter configured for 150KHz. Simulating these values in LTSpice yields the following magnitude response shown in Figure 5.1.6.

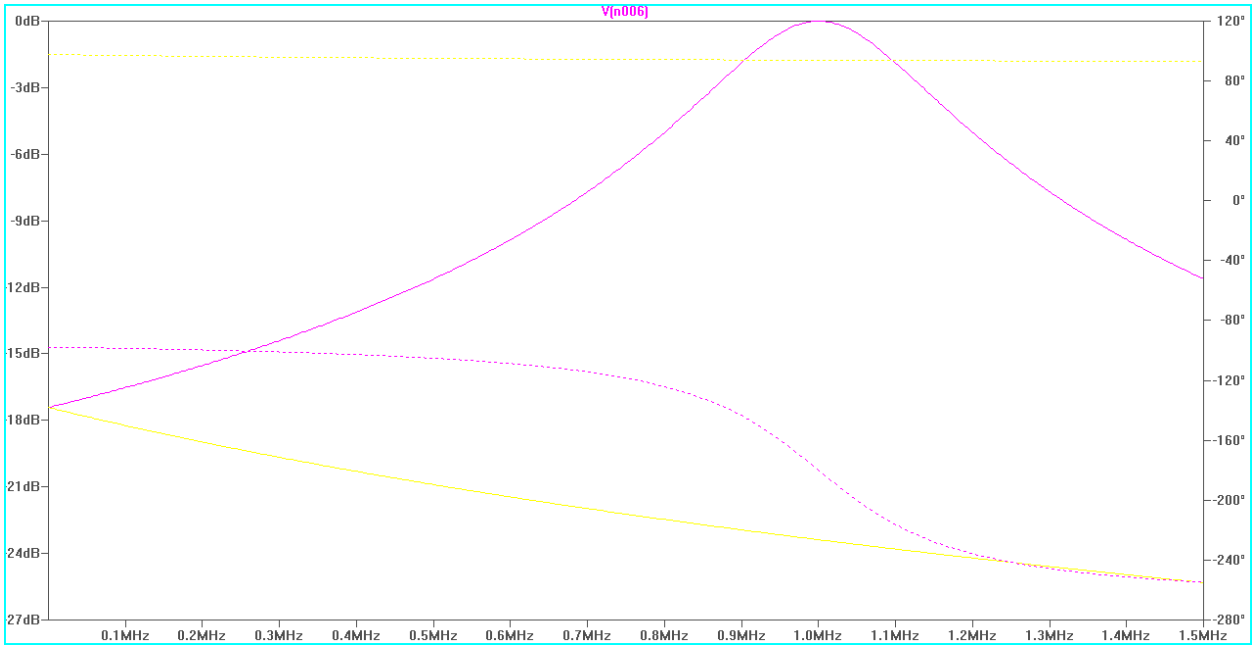

Figure 5.1.6. – Magnitude response of a simple image reject filter.

The center frequency shown in Figure 5.1.6 is indeed 1 MHz. The yellow trace on the response is the response of the frequencies at which the image would appear. Since LTSpice is incapable of plotting negative frequencies, the Q channel phase is reversed by 180 degrees such that the response is shifted left and not right. This enables us to observe the image suppression since the magnitude response has even symmetry about the magnitude axis.

This technique of applying the output of the Q channel to the input of any integrator in the I channel will always result in a shifted frequency if all resistors are chosen accordingly. Thus, one can cascade any amount of integrators in the receive chain and achieve image rejection if the integrators are all cross coupled. We can implement a higher order filter using a cross coupled Tow Thomas bi-quad filter consisting of two complex conjugate pole pairs and a real pole. Figure 5.1.7 shows this implementation.

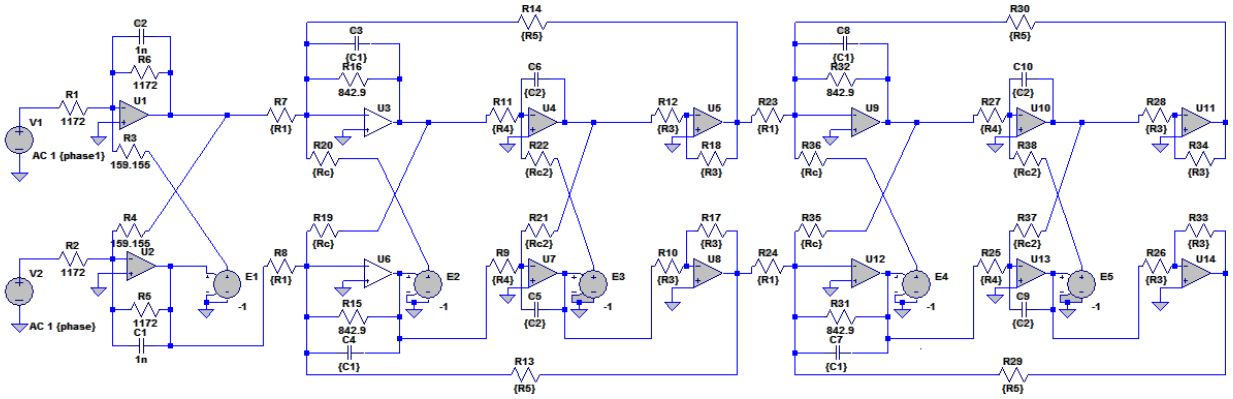

Figure 5.1.7. – Ideal image reject filter using a fifth order Butterworth low pass filter configuration.

This particular filter is configured to operate as a 5th order Butterworth low pass filter if no coupling is present. The values for the components were generated using a program called Filter Wiz Pro. The magnitude response for this filter is shown in Figure 5.1.8.

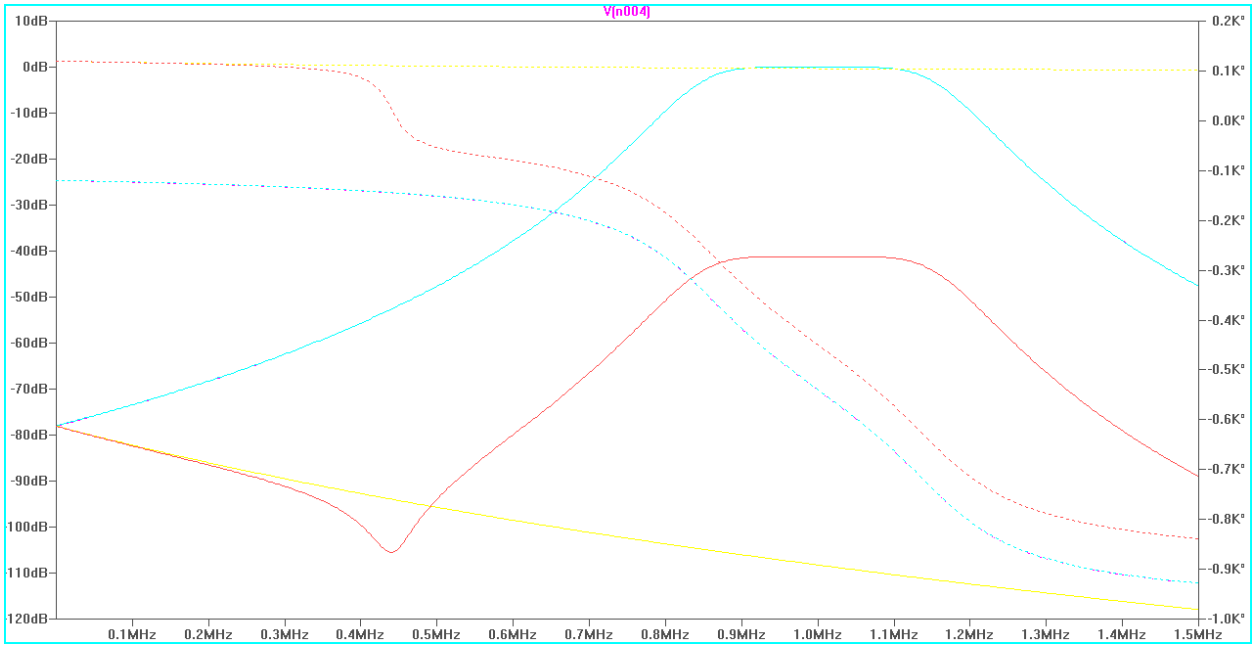

Figure 5.1.8. – Magnitude response of a 5th order BPF (ideal).

The turquoise response occurs when the filter is configured with a perfect 90 degree difference between the I and Q channels. The red trace indicates what happens if there is a 1 degree offset present between the I and Q channels. The yellow trace once again shows us the suppression of the image. The image rejection is well above the 20 and 50 dB attenuation given by the specifications. Both the I and Q channel outputs yield similar transfer characteristics. To observe the group delay in the passband, we take the derivate of the phase with respect to frequency and divide by 2*pi.

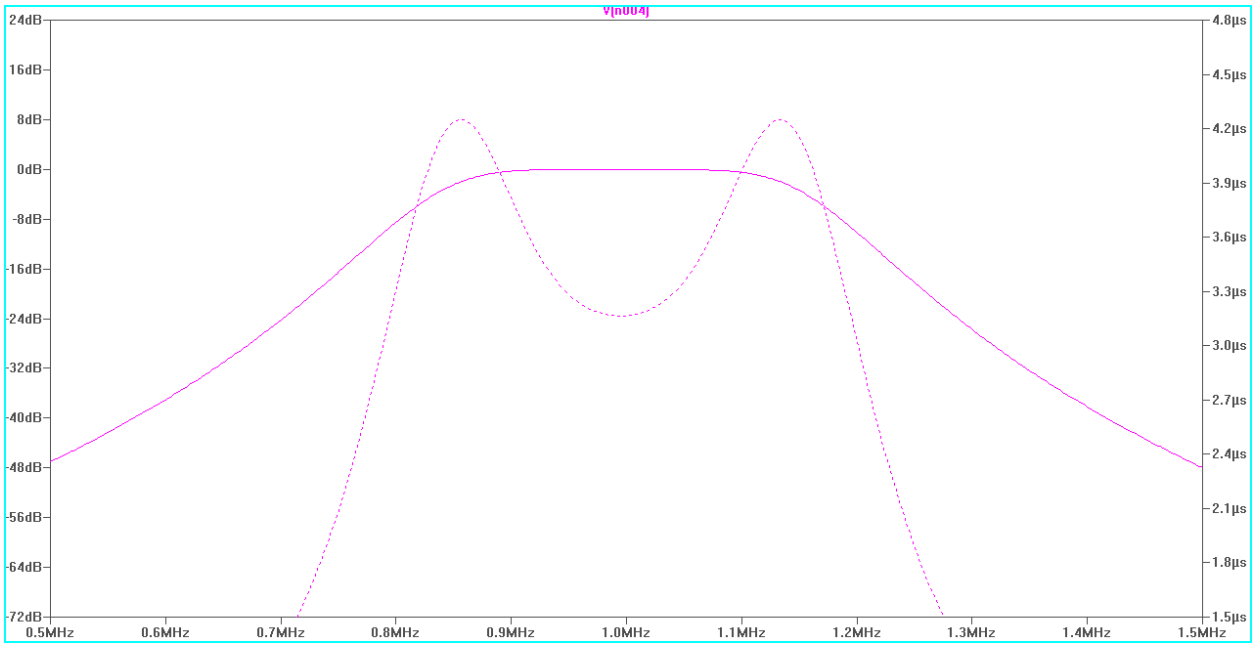

Figure 5.1.9. – Magnitude response of a 5th order BPF with group delay (ideal).

The dotted line in Figure 5.1.9 indicates the group delay in the passband. The maximum group delay is 4.26 µs. If we use standard resistor values and non-ideal op-amps with an open loop gain of 5.2K and a GBWP of 100MHz, the magnitude response is as follows:

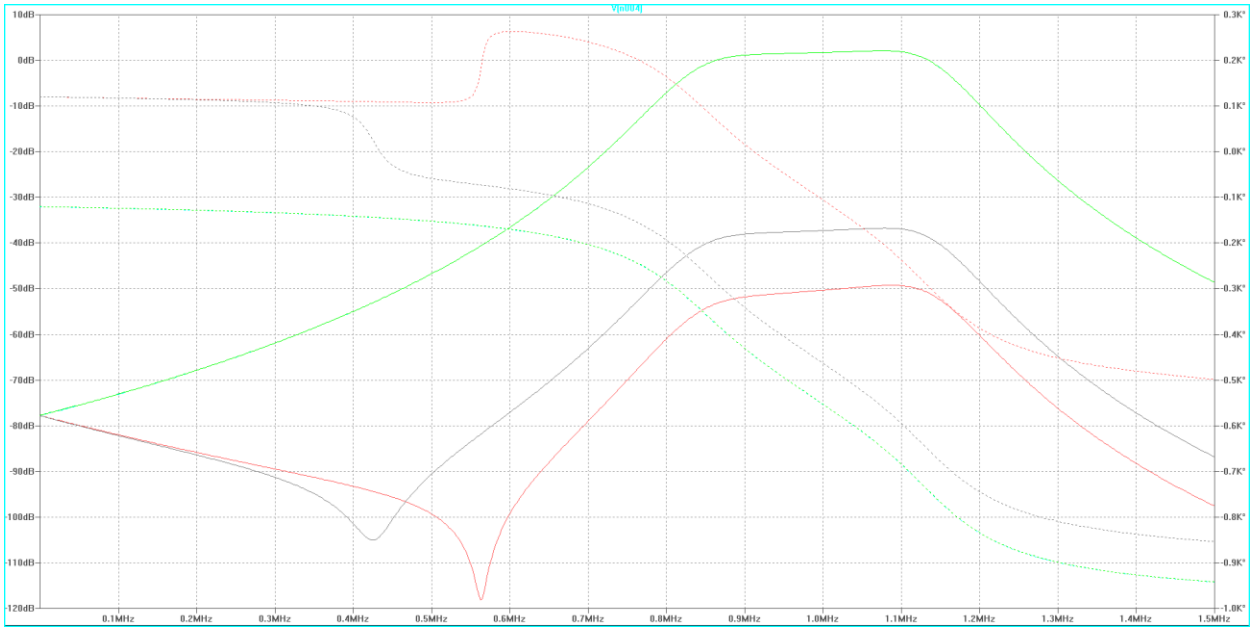

Figure 5.1.10. – Magnitude response of a 5th order BPF (non-ideal).

The image (maroon) in Figure 5.1.10 is not as suppressed using this configuration.

For the receiver, the LT6230 op amp from Linear Technologies was chosen due to its high gain bandwidth product and fast slew rate. As this design was intended for a single supply design at 5V, the circuit was redrawn with the LT6230 and proper referencing to a common mode voltage (CMV) of 2.5V. Figure 5.1.11 shows the complete schematic with proper value components. The inverters have been replaced with inverting op amps. In a fully differential design, the inverters would not be needed due to the inverting output terminals present in a differential op amp.

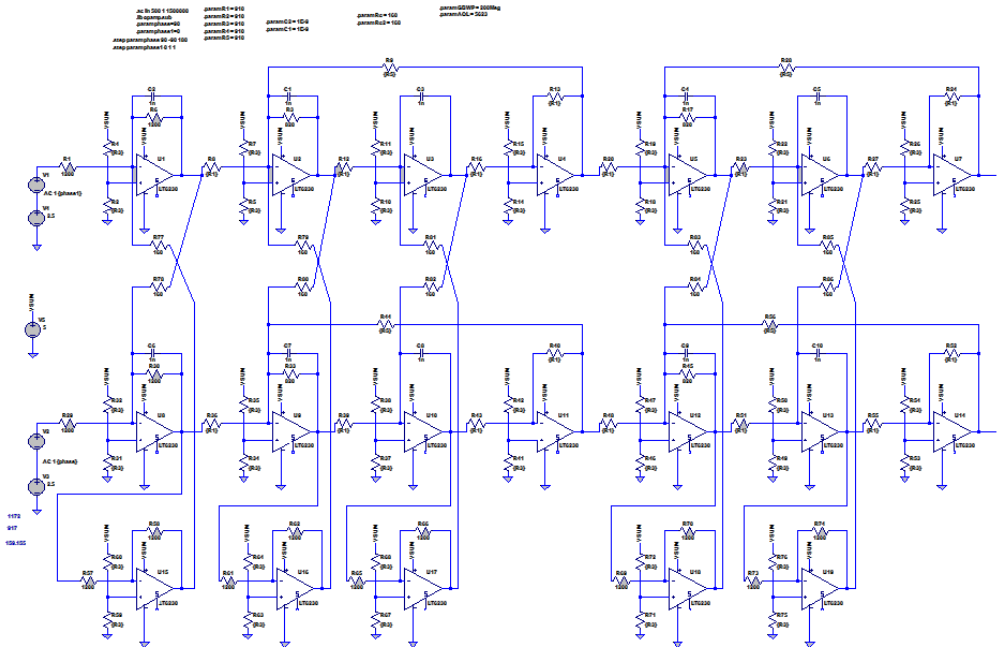

Figure 5.1.11. – Completed Filter Schematic. This is the schematic view in LTSpice.

The schematic created in OrCAD Capture can be viewed in Appendix B, Figure B.5 and Figure B.6 more clearly than the schematic in Figure 5.1.11.

Sub-Section 2. BPSK Theory, Design, and Simulation

A BPSK system only has two possible states with a phase deviation of pi radians. A BPSK system can be modeled using a DSB-SC architecture with a binary message signal as shown below:

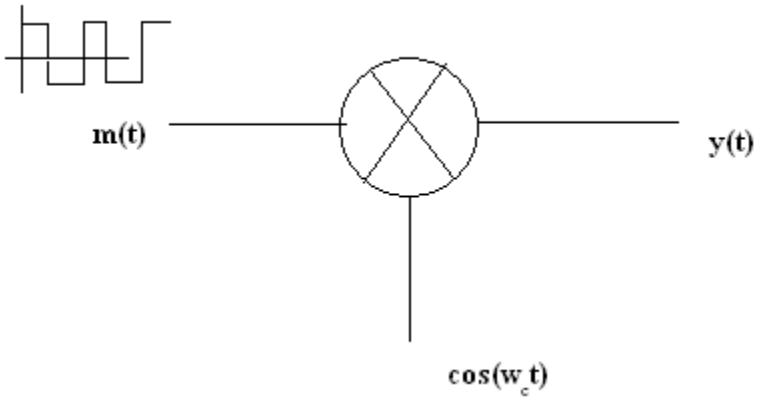

Figure 5.2.1. – A BPSK system representation using a DSB-SC architecture.

The output of the mixer is then $y(t) = m(t)\cos(\omega_c t)$ or $y(t) = \cos(\omega_c t + k(t))$ where k is either pi or 0 depending on the polarity of the message signal. Shown below is the waveform of the carrier signal with a modulated message signal. The simulation was performed in Simulink. The input data is not bi-polar and the carrier is not smooth due to sampling limitations.

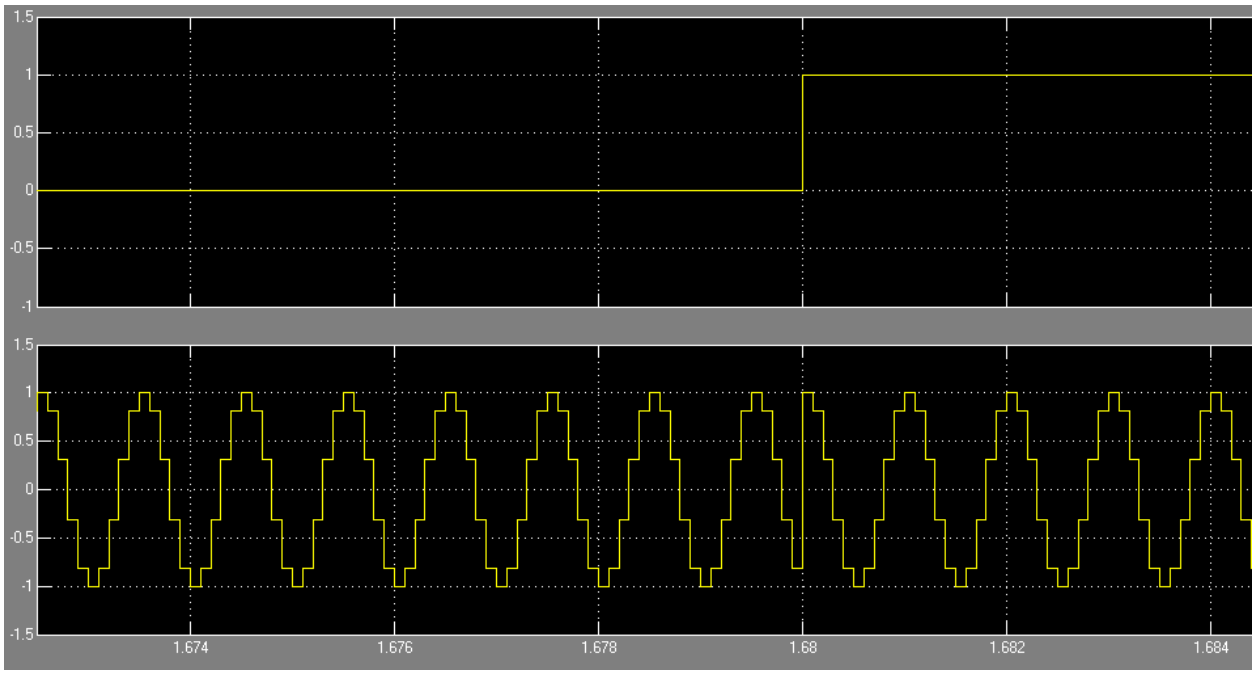

Figure 5.2.2. – Simulation showing the modulated signal. The phase changes when the message signal changes state.

To demodulate a BPSK signal down to a low IF, we can multiply the modulated signal by I and Q phased local oscillators. To down convert the signal to 1 MHz, we multiply by an LO which is 1MHz higher than the carrier frequency (high side LO injection):

$$\cos(\omega_{LO}t) * \ cos\big(\omega_c t + k(t)\big) = cos\big(\omega_{c+LO}t + k(t)\big) + cos\big(\omega_{IF}t - k(t)\big)$$

After image rejection and channel selection processing is performed at the IF, the signal is then down converted by mixing a sinusoid with the same frequency as the IF:

$$\cos(\omega_{IF}t) * cos\big(\omega_{IF}t - k(t)\big) = \cos\big(k(t)\big)$$

The signal is now demodulated. Higher frequencies are filtered out and are therefore neglected. A suitable filter for this system can be an integrate and dump filter which acts as a low pass filter. The down converted low IF signal is shown in Figure 5.2.3 below. High frequencies exist in the signal due to no filtering. Both the I and Q demodulation channels are shown.

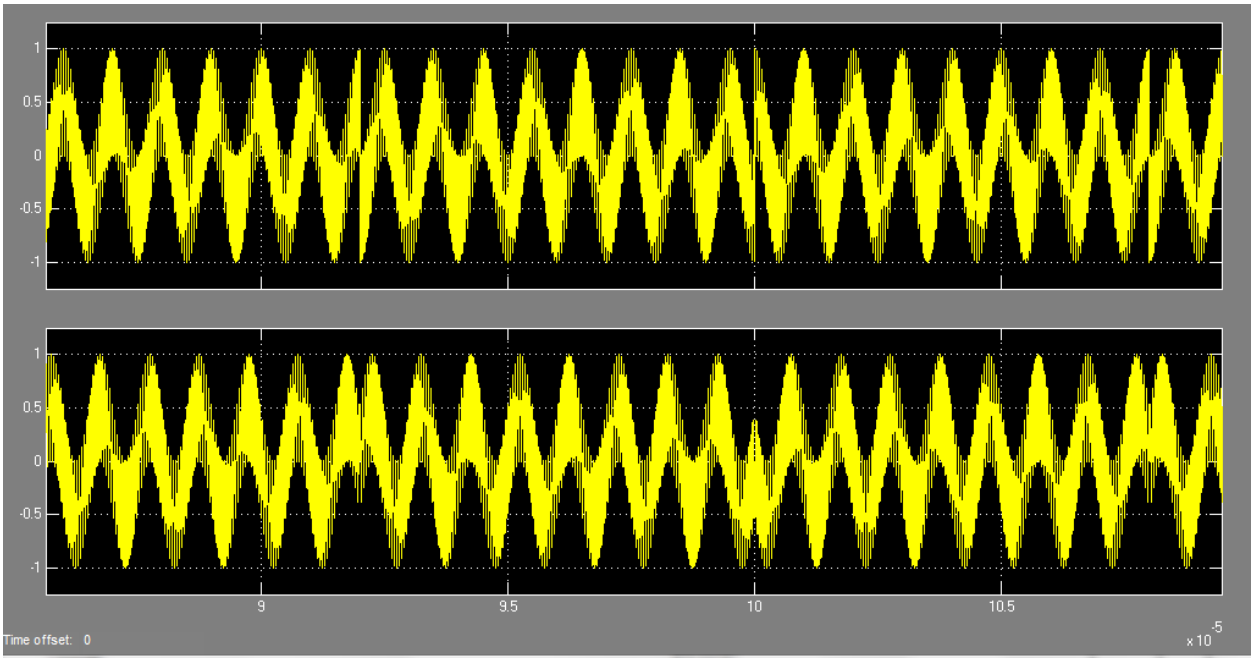

Figure 5.2.3. – Down converted low IF signal without filtering (both I and Q channels).

Figure 5.2.4 shows the final demodulated signal without filtering on the next page.

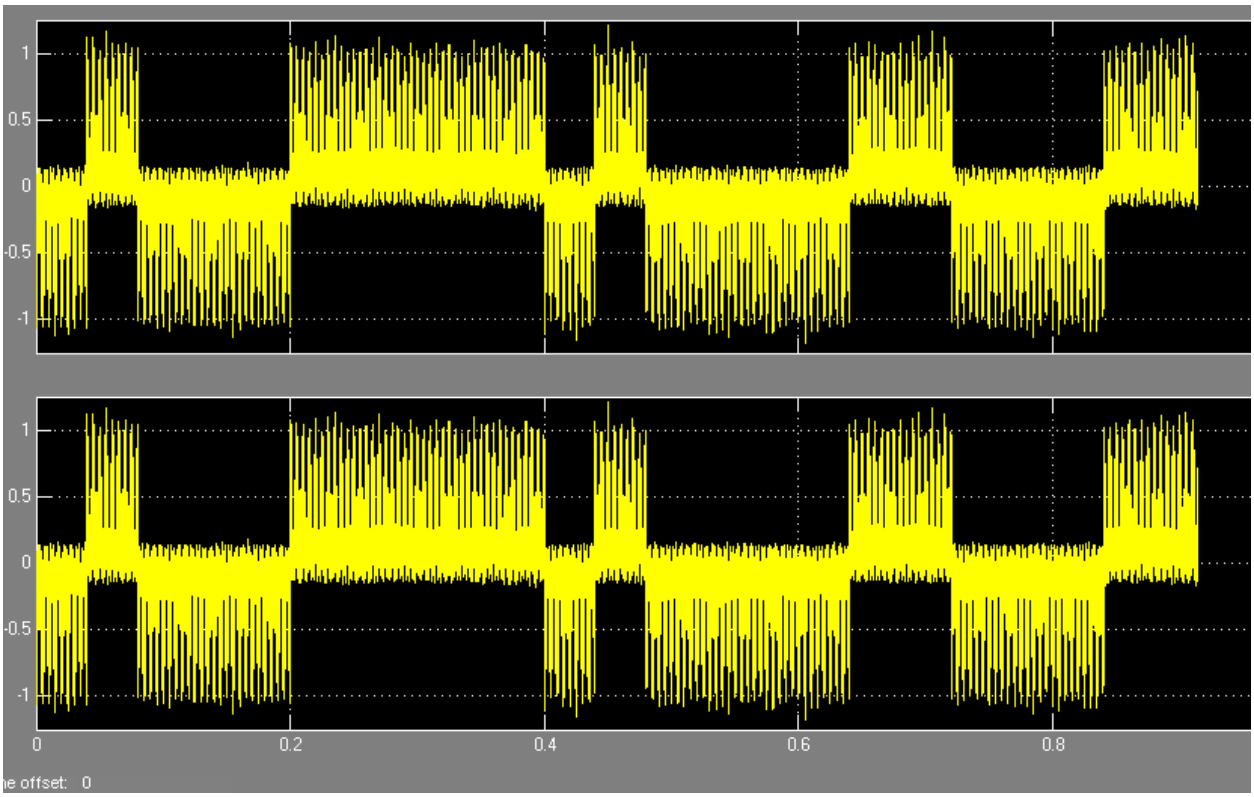

Figure 5.2.4. – Demodulated output signal (both I and Q channels).

The aforementioned signal capture is for a perfect channel. Figure 5.2.5 shows the demodulated wave with additive Gaussian noise.

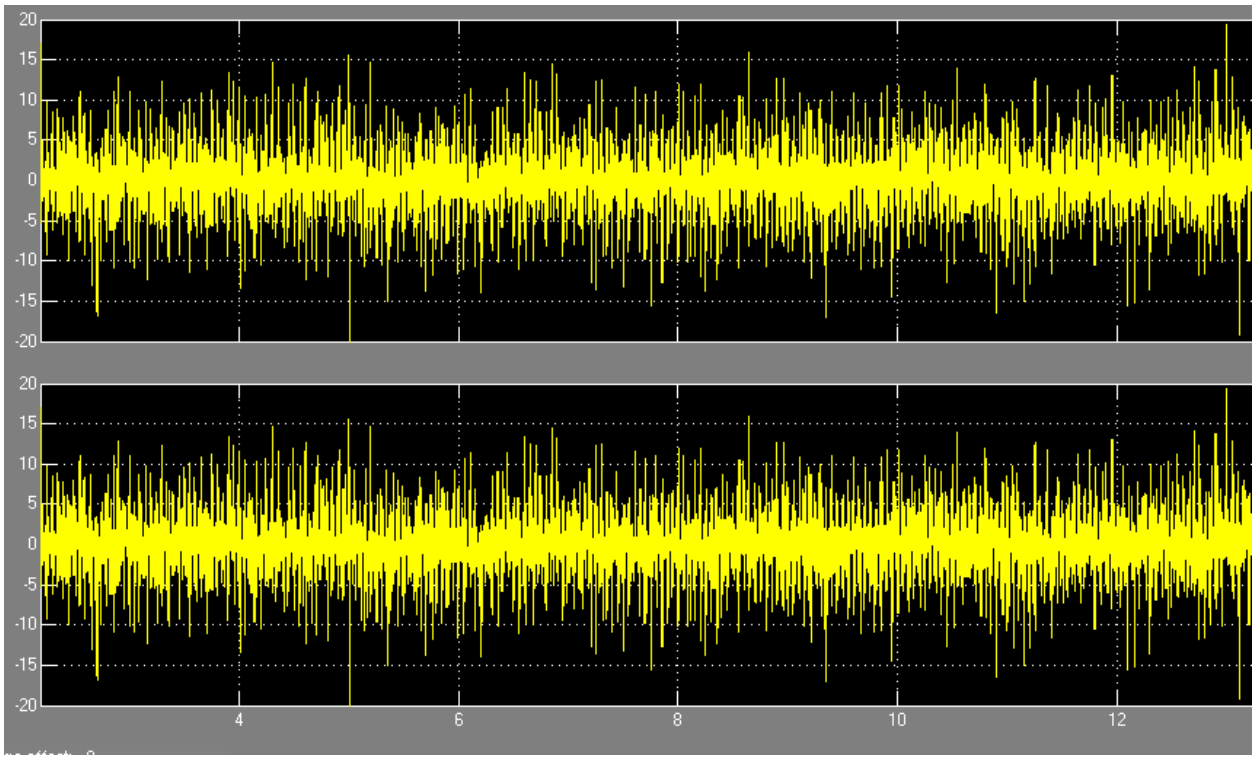

Figure 5.2.5. – Unfiltered demodulated signal with AWGN.

The previous figures were obtained using Simulink. The entire BPSK communication system was simulated in Simulink with the digital communications toolbox. The RF frequency, however, had to be reduced to 10 MHz due to computer processing limitations. The system sampling frequency had to be at least twice the Nyquist frequency, but to obtain more accurate results, a sampling frequency of 100 MHz was used. The first stage mixers, which represent the RF down converters, were set to 11 MHz. The second stage mixer, which will ultimately be performed using a DSP, was set to 1 MHz to down convert the final signal to baseband.

The system is shown in the next page with the error rate calculators present on the right side of the page. For the given test in the figure, Display2 represents the Q channel and Display1 represents the I channel. A total of 15890 bits were sent and the average bit errors was 77 which yields an average BER of .004971 or .49% for a given Eb/No value of 8.2 dB (Note that this value does not meet specifications. This is merely for an example case). An additive Gaussian noise channel was used to perform the simulation as can be seen in the following page. To extract data from the output of the integrate and dump filters, a sign block was used. A look up table had to be used only due to the fact that the sign block output values based upon three conditions and not two. All of the aforementioned functions can be readily implemented in a DPS/FPGA.

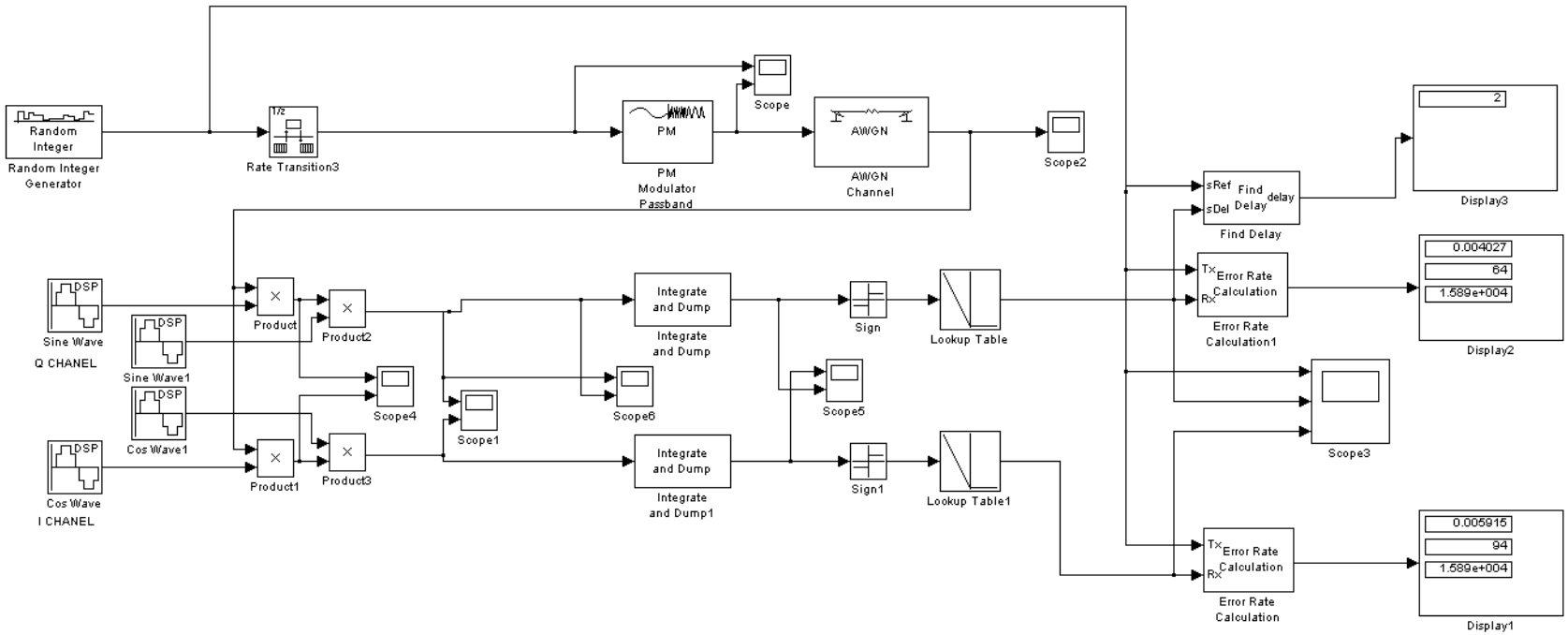

Figure 5.2.6. – Entire simulated BPSK system in Simulink.

Sub-Section 3. Modulator/Demodulator

The modulation circuitry for the receiver consisted of a basic attenuation network and the LT5568 IQ modulator. Since the system is a BPSK system, the quadrature component is not used on the modulation side and thus had its pins tied together. The inputs to the modulator are differential inputs, and therefore any data that is sent to the modulator must be in differential form. Failure to provide differential signaling can cause severe harmonic distortion. The CMV of the differential inputs to the modulator has to be .54V with a total voltage of less than 2.5V. Thus, an attenuation network was added to the input of the modulator to accommodate this specification. The modulator schematic can be seen in Appendix B, Figure B.1.

The output and LO are both connected to SMA cables for ease of testing and connectivity. The demodulation circuitry utilized the LT5575 IQ direct down conversion demodulator. This demodulator had two differential outputs for the I and Q channel. Since the filter is single ended, an instrumentation amplifier (IA) had to be utilized to convert from differential to single ended signaling. Also, all the LT6230 op amps in this design could not accept a voltage below 1.5V or above 4V. Therefore, the signals from the demodulator had to be AC coupled and centered around a reference voltage of 3V before being fed into the instrumentation amplifier. The IA was then referenced to the CMV of 2.5V which is the same CMV as the filter. Refer to Appendix B, Figure B.2 for the demodulator, Figure B.3 for the DC offset circuit, and Figure B.4 for the instrumentation amplifier.

Sub-Section 4. Programmable Gain Amplifier

The specifications require that a PGA be used in the receive chain to correct for signal level changes caused by near-far problems. This is done by using two resistor banks controlled by multiplexer switches and an op-amp in the inverting configuration as can be seen in Appendix B, Figure B.7. The FPGA will control a switch array that adjusts the gain of the amplifier based upon which resistor is grounded. The FPGA will have an algorithm that checks the amplitude of the signal and will adjust the switches accordingly. An explanation of the PGA software can be found in Sub-Section 5, Software Component 7.

Sub-Section 5. Software/VHDL

At this point, it must be noted that the Nexys-2 FPGA was selected as the software demodulation device. All VHDL code is listed in Appendix D.

The main blocks that need to be implemented in software include the following:

1) Interface to ADC – the FPGA will have to interface with an ADC to input data.
2) A digital local oscillator/mixer – this will perform the final demodulation to baseband. The local oscillator should have enough samples to offer reliable data.
3) Integrate and dump – this block essentially acts as a low pass filter. Its main purpose is to accumulate enough data to provide the sign block information about the polarity of the signal.
4) Sign determiner (hard limiter) – this block will output a logic '1' if the input is positive and '0' if negative. The output of this block will be a completely demodulated signal
5) Pseudo-Random Binary Sequence Generator – random data will need to be generated by the DSP itself to avoid clock conflicts. If a different device were to be used, a digital PLL will have to be built which is beyond the scope of this project.
6) Sync Clock – This is used to sync with the LO's.
7) PGA Feedback – the FPGA will have to have a control system to determine the switch values for the PGA.

Software Component 1. – ADC

The schematic for the ADC is shown in Appendix B, Figure B.8. The output of the PGA was AC coupled to the ADC for the easiest configuration. Although an ADC is a hardware component, it is the interface between the hardware and software realm of the project and therefore was included in the software section. The 10-bit output of the ADC was fed into the Nexys-2 board for processing. An output of the Nexys-2 board was used for the clock that was fed into the ADC. The ADC outputs parallel data for each falling edge of the clock (up to 20 MHz). The sample rate used for this project was 5 MHz. Data is delayed by 5 clock cycles and therefore has to be taken into consideration in software.

VHDL code lines 399 to 478 in Appendix D contain the ADC sampling process.

Software Component 2. – Digital Oscillator Multiplication and Demodulation

After the data has been input into Nexys board, it needs to be multiplied by a local oscillator to shift the spectrum down to baseband. The sample rate of the ADC was set to 5 MHz. Because of this unique sampling frequency, no multiplication was necessary. This is due to the fact that separating a signal into 5 separate parts and multiplying those parts by their corresponding cosine terms results in only one of three states. Either the sample is passed through unaffected, inverted, or not passed through at all. After all the samples are complete, the data is pushed out of the process onto another set of signals for further processing.

VHDL code lines 399 to 478 in Appendix D contain the multiplication process.

Software Component 3. – Integration

Since integration is essentially a Riemann sum of areas which can be composed of rectangle approximations, the integration step can be simplified to a basic addition operation with a constant scale factor. This can be further simplified by removing the scale factor because only the polarity is needed in the next step and not an actual quantitative value. Thus, no multiplication is present anywhere in the system which is highly efficient.

VHDL code lines 500 to 549 in Appendix D contain the integration process.

Software Component 4. – Sign Determiner

This process is fairly straightforward. Since the data from the ADC and all subsequent data is in signed two's complement, only the most significant bit of the output of the integrator needs to be read to determine the polarity.

VHDL code lines 559 to 570 in Appendix D contain the sign determine process.

Software Component 5. PSBR Signal Generator

A simple array and clock generator was used to create a PRBS sequence. Each time the allocated clock line pulled low, a signal was indexed and output differentially from the board.

VHDL code lines 321 to 382 in Appendix D contain the signal generation process.

Software Component 6: Sync Clock

A 10 MHz clock had to be generated by the Nexys-2 board to synchronize it with the LO's since no clock extraction circuitry was used in this circuit. The LO's external sync inputs were 50 Ohm inputs and required an AC coupled signal. Therefore the output of the Nexys-2 board was fed to a 10 uF capacitor before the LO's. It must also be noted that it is impossible to generate a 10 MHz clock using falling edge triggered clock generation as done for the 5 MHz signal and the 250 KHz signal due to the fact that a total of 2.5 clock cycles is needed for every 1 half period of the sync clock. Therefore, Xilinx's IP Core generator had to be used to instantiate a clock divider 'black box' design. The VHDL code for this is proprietary and unknown. Only a component with a port map was provided. The clock signal was fed to the component and the 10 MHz clock was taken from the output.

VHDL code lines 296 to 302 in Appendix D contain the sync clock process

Software Component 7: PGA Feedback

A software flow diagram of the PGA feedback algorithm can be found below in Figure 5.5.7.1.

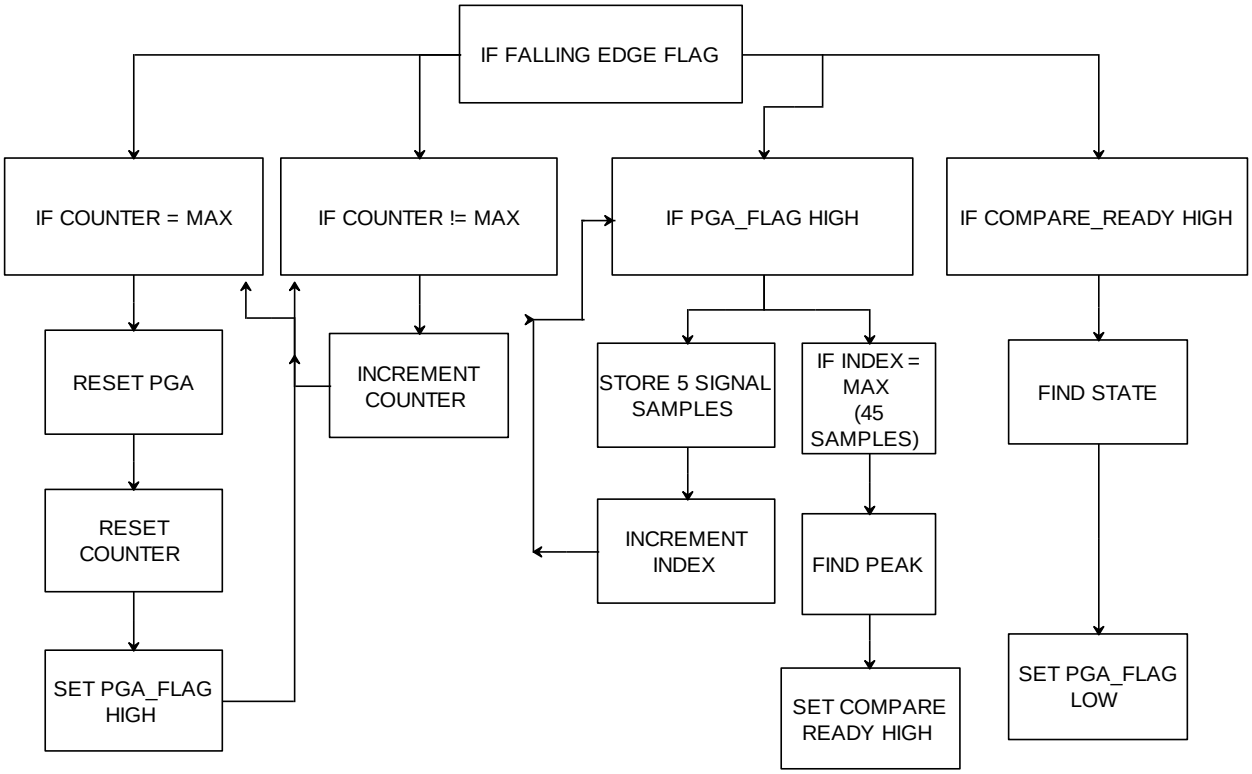

Figure 5.5.7.1 - Software flow diagram for the PGA

A master *if* statement will reset the PGA and initialize a sample store and compare sequence if a counter hits a certain value. This process exists to avoid positive feedback in the system. The following scenario describes how positive feedback can occur. Suppose the signal is very weak, and the maximum gain of the PGA needs to be applied. Assuming the PGA has a gain of 1 initially, the ADC will read the signal, detect the peak, and apply the appropriate gain. Because the PGA is fed into the ADC, the ADC will now read the signal with the gain applied and assume that the signal is ‘normal’ in which case it will set the gain back down to 1. This will cause the signal to become small again, and the process will continue to oscillate. By resetting the PGA in the master if statement, this oscillation is avoided.

A state variable diagram is shown in Figure 5.5.7.2 for the different states a signal can exist in depending on its amplitude. Note that the bi directional paths exist because the PGA is reset upon each master *if* statement reset.

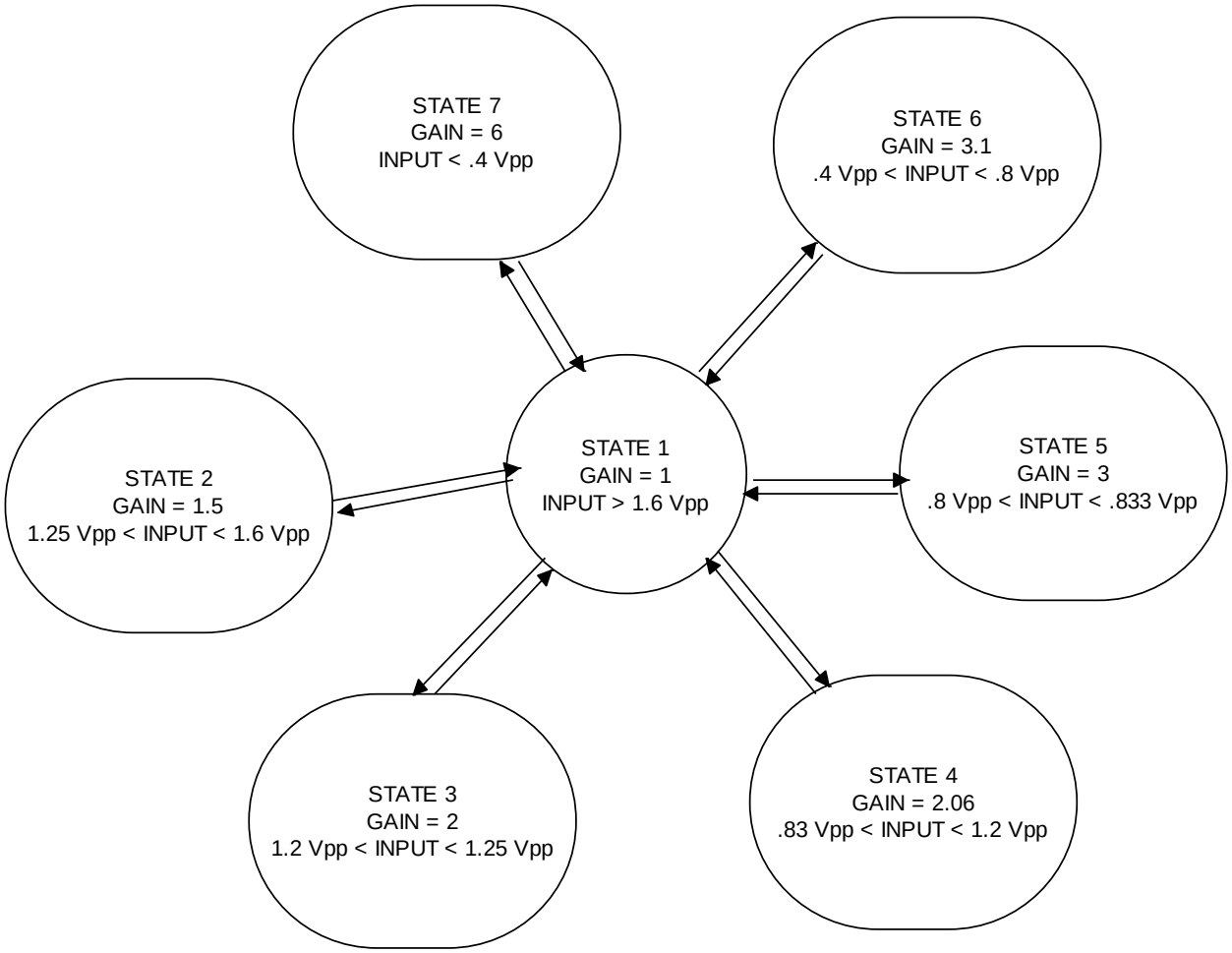

Figure 5.5.7.2. – State Variable Diagram

The gain steps of the PGA are clearly non-linear and not very ideal. However, the PGA in this project is more proof of concept than practical. Moreover, the gains can be adjusted by setting the resistor values accordingly and calibrating the code for the different states.

Section 6. CONSTRUCTION AND INTEGRATION

Construction consisted of soldering all components onto the PCB's. No further soldering beyond this was necessary. Because all of the system was localized onto a board, there was no need for interconnects beyond the interface with the Nexys-2. Due to unforeseen circumstances, the modulator and demodulator QFN footprints used for the first revision were footprints taken from another part that had an identical footprint name. This footprint was of the incorrect size and therefore the components could not be soldered on due to the fact that RF components are near impossible to wire mod. A second revision had to be created with the correct footprint. However, since all of the complex filter components were still on the initial revision board, the two boards were jumped together to avoid the removal and replacement of all the components which would have been time consuming.

Wires were soldered onto the pads on the PCB to interface to the PMOD™ connectors on the Nexys-2 board directly. The sync clock output was sent to a separate prototype PCB with the 10uF capacitor to the LO's. The data output was a simple wire connection.

Section 7. SUB SYSTEM TESTING

Sub-System 1. Filter/Demodulator Testing

To properly characterize the filter, each individual chain had to be analyzed individually, and then analyzed when fully completed. At first, each filter was decoupled (I/Q cross coupling removed) and the input test signal was inserted directly at the inputs of the filters *with* the instrumentation amplifier and CMV shifting circuitry in place. This proved to be an erroneous test set up since the output data was clearly wrong (see Table 7.1.1 below). Thus, the input test signals had to be inserted at the inputs of the IA which yielded fairly accurate low pass transfer characteristics for a Butterworth response as can be seen in Table 7.1.2.

F (Hz)	Vpp input (V)	Vpp2 output (V)	Gain	Gain (dB)
1.00E+03	1E-45	0	0	#NUM!
1.00E+04	0.5	0.5	1	0
5.00E+04	0.5	0.5	1	0
1.00E+05	0.62	0.58	0.935484	-0.57927
1.10E+05	0.68	0.6	0.882353	-1.08715
1.20E+05	0.72	0.62	0.861111	-1.29882
1.30E+05	0.76	0.64	0.842105	-1.49267
1.50E+05	0.84	0.6	0.714286	-2.92256
1.60E+05	0.88	0.52	0.590909	-4.56959
1.70E+05	0.9	0.5	0.555556	-5.10545
1.80E+05	0.92	0.46	0.5	-6.0206
1.90E+05	0.96	0.42	0.4375	-7.18044
2.00E+05	0.96	0.38	0.395833	-8.04975
2.20E+05	1	0.3	0.3	-10.4576
2.40E+05	1.06	0.24	0.226415	-12.9019
2.80E+05	1.12	0.112	0.1	-20
3.00E+05	1.16	0.1	0.086207	-21.2892
5.00E+05	1.26	0.08	0.063492	-23.9456

Table 7.1.1 – Recorded values for the initial set up. The input voltage incorrectly varied with frequency. The voltage output to input *ratio* however, did yield a correct Butterworth response.

F (Hz)	Vpp input (V)	Vpp2 output (V)	Gain(I-Channel)	Gain (dB)	Vpp input (V)	Vpp2 output (V)	Gain(Q-Chanel)	Gain (dB)
1.00E+03	1.48	4.48	3.027027027	9.620325972	1.48	4.8	3.243243243	10.21959044
5.00E+03	1.48	4.32	2.918918919	9.304440628	1.48	4.72	3.189189189	10.07360566
1.00E+04	1.48	4.32	2.918918919	9.304440628	1.48	4.72	3.189189189	10.07360566
2.00E+04	1.48	4.32	2.918918919	9.304440628	1.48	4.72	3.189189189	10.07360566
5.00E+04	1.48	4.32	2.918918919	9.304440628	1.48	4.64	3.135135135	9.925125303
1.00E+05	1.48	3.92	2.648648649	8.460487033	1.48	4.48	3.027027027	9.620325972
1.10E+05	1.48	3.84	2.594594595	8.281390179	1.48	4.32	2.918918919	9.304440628
1.20E+05	1.48	3.68	2.486486486	7.911722066	1.48	4.16	2.810810811	8.976632305
1.30E+05	1.48	3.36	2.27027027	7.12155124	1.48	3.92	2.648648649	8.460487033
1.40E+05	1.48	3.2	2.162162162	6.697765258	1.48	3.6	2.432432432	7.720815707
1.50E+05	1.48	2.88	1.945945946	5.782615447	1.48	3.28	2.216216216	6.912242566
1.60E+05	1.48	2.56	1.72972973	4.759564998	1.48	3.04	2.054054054	6.252237364
1.80E+05	1.48	2	1.351351351	2.615365605	1.48	2.48	1.675675676	4.483799309
2.00E+05	1.48	1.36	0.918918919	-0.73445614	1.48	1.62	1.094594595	0.785065983
2.50E+05	1.48	0.6	0.405405405	-7.8422093	1.48	0.76	0.513513514	-5.78896246
3.00E+05	1.48	0.27	0.182432432	-14.7779590	1.48	0.36	0.243243243	-12.2791842
5.00E+05	1.48	0.06	0.040540541	-27.8422093	1.48	0.06	0.040540541	-27.8422093

Table 7.1.2 – Recorded values for the individual filter test. The I channel filter results are on the left and the Q channel results are on the right.

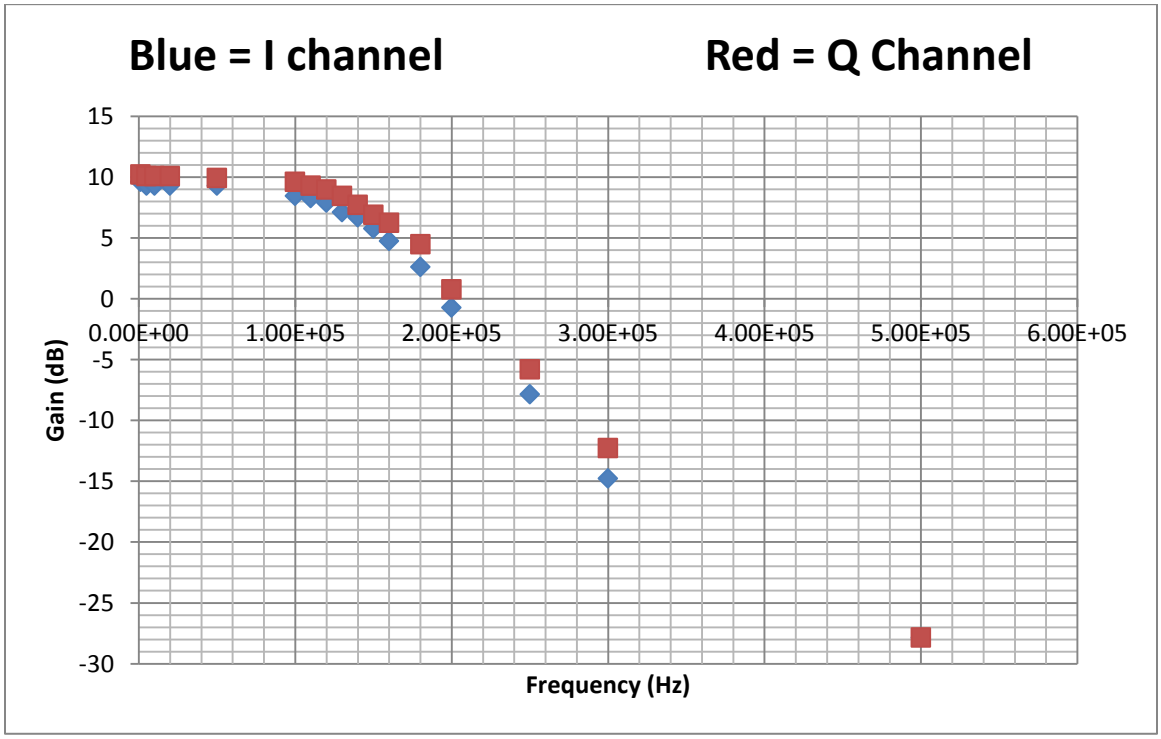

Figure 7.1.1 – Plotted responses for the filters.

The filters, tested individually, yielded excellent test results and each channel had a 3 dB cut off frequency of 150Khz, which is precisely what the filters were designed for. However, there existed a slight gain difference between the I and Q channel suggesting the presence of I and Q channel mismatch. This mismatch may lead to degradation in image rejection (as can be seen later). To test the entire complex filter, the demodulator was needed. The demodulator was soldered on, and the RF and LO inputs were connected to two synchronized signal generators as can be seen in Figure 7.1.2 below.

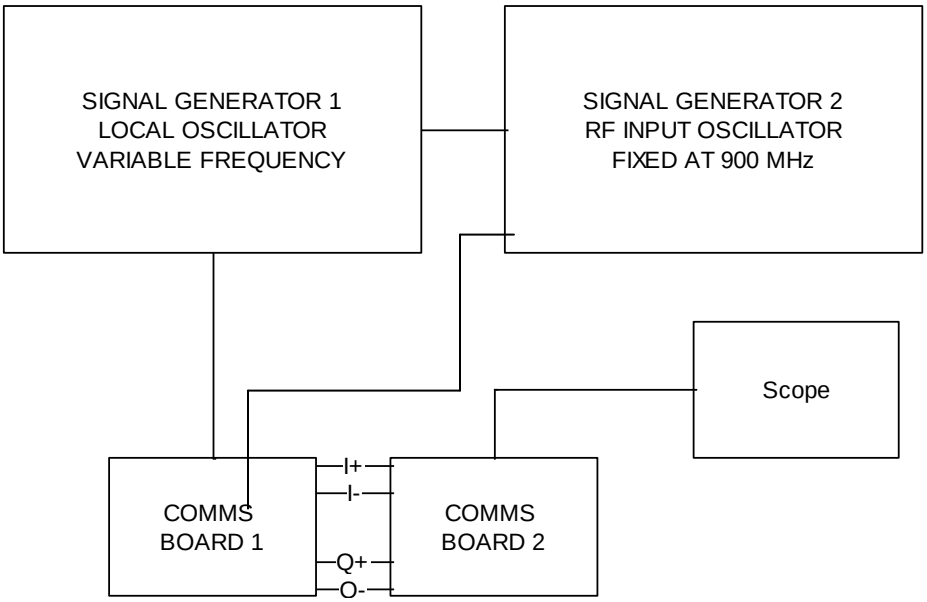

Figure 7.1.2 – Test setup diagram.

There exist two communications boards due to a manufacturing footprint error. The I and Q channels were simply jumped from one board to the other. Due to the low IF of 1 MHz, neither the line lengths nor the type of wire used was important. With the RF input set to 900 MHz and the LO set to 901 MHz, the output of the demodulator yielded a clean 1 MHz sinusoid on the I positive channel as can be seen in Figure 7.1.3 below.

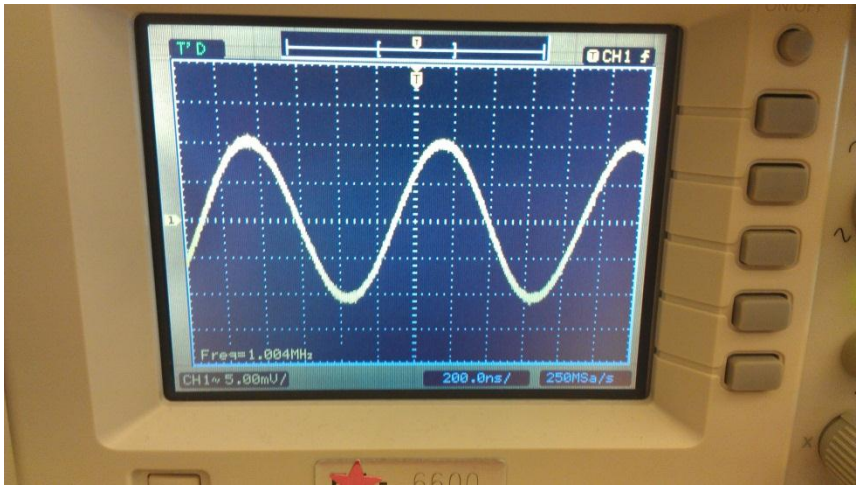

Figure 7.1.3 – I channel demodulator output.

Figure 7.1.4 shows the I channel with both differential lines probed.

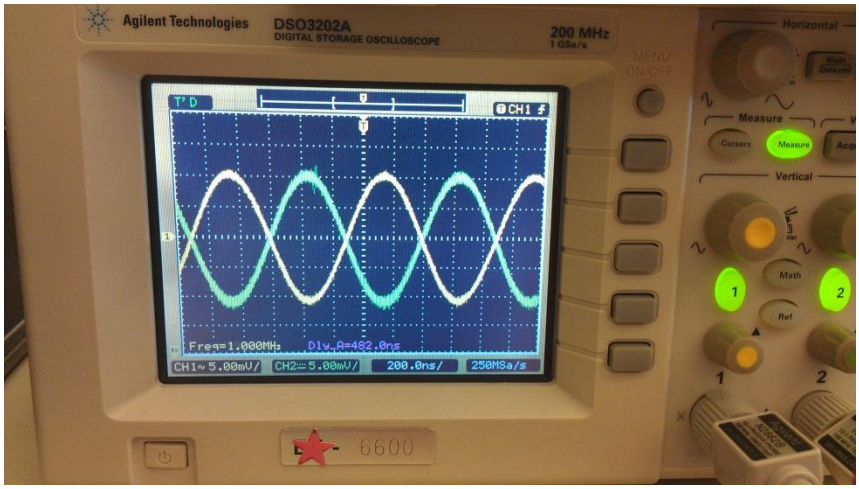

Figure 7.1.4 – I channel differential demodulator output.

Figure 7.1.5 shows the I positive and Q positive channel. The delay between the two signals is 750 ns (or 250 ns) which is indeed a quarter of a wavelength (1us) indicating that the I and Q channels are 90 degrees apart.

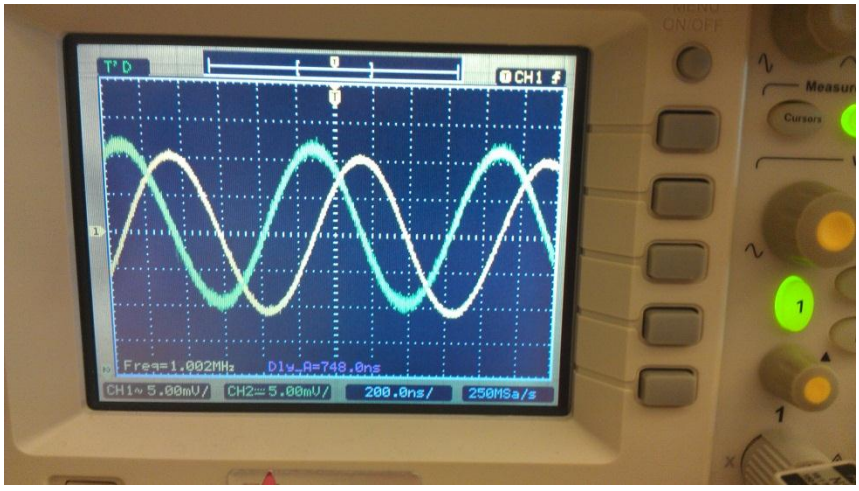

Figure 7.1.5 – I and Q channel positive demodulator outputs. Notice how the signals are 90 degrees apart.

Finally, Figure 7.1.6 shows the Q channel output.

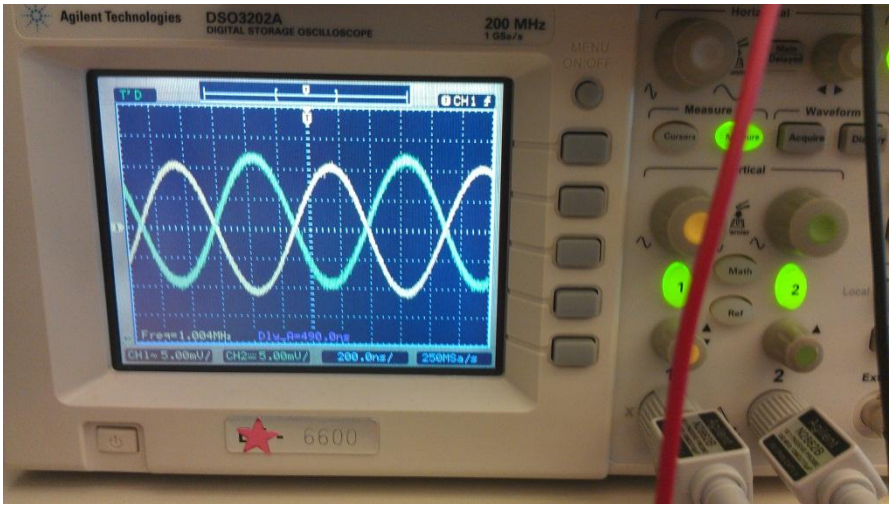

Figure 7.1.6 – Q channel differential demodulator output.

Because the inputs to the filter were completely complex, the full bandpass filter can be tested. The LO was varied from the positive sequence to the negative sequence to observe both the pass band response and the image response. Over 100 data points were taken, so Figure 7.1.3 only shows several samples.

LO (MHz)	IF (MHz)	RF (MHz)	IF FILTER OUT (Vpp)	IF FILTER OUT (dB)
901	1	900	1.841	5.30107577
901.08	1.08	900	1.881	5.487775911
901.42	1.42	900	0.084	-21.51441428
900.77	0.77	900	0.24	-12.39577517
899.5	-0.5	900	0.001	-60
898.8	-1.2	900	0.108	-19.33152489

Table 7.1.3 – Table of 5 random samples chosen from the data collected from the complex filter.

Figure 7.1.7 shows the filter response over the positive and negative sequence in addition to the simulated values.

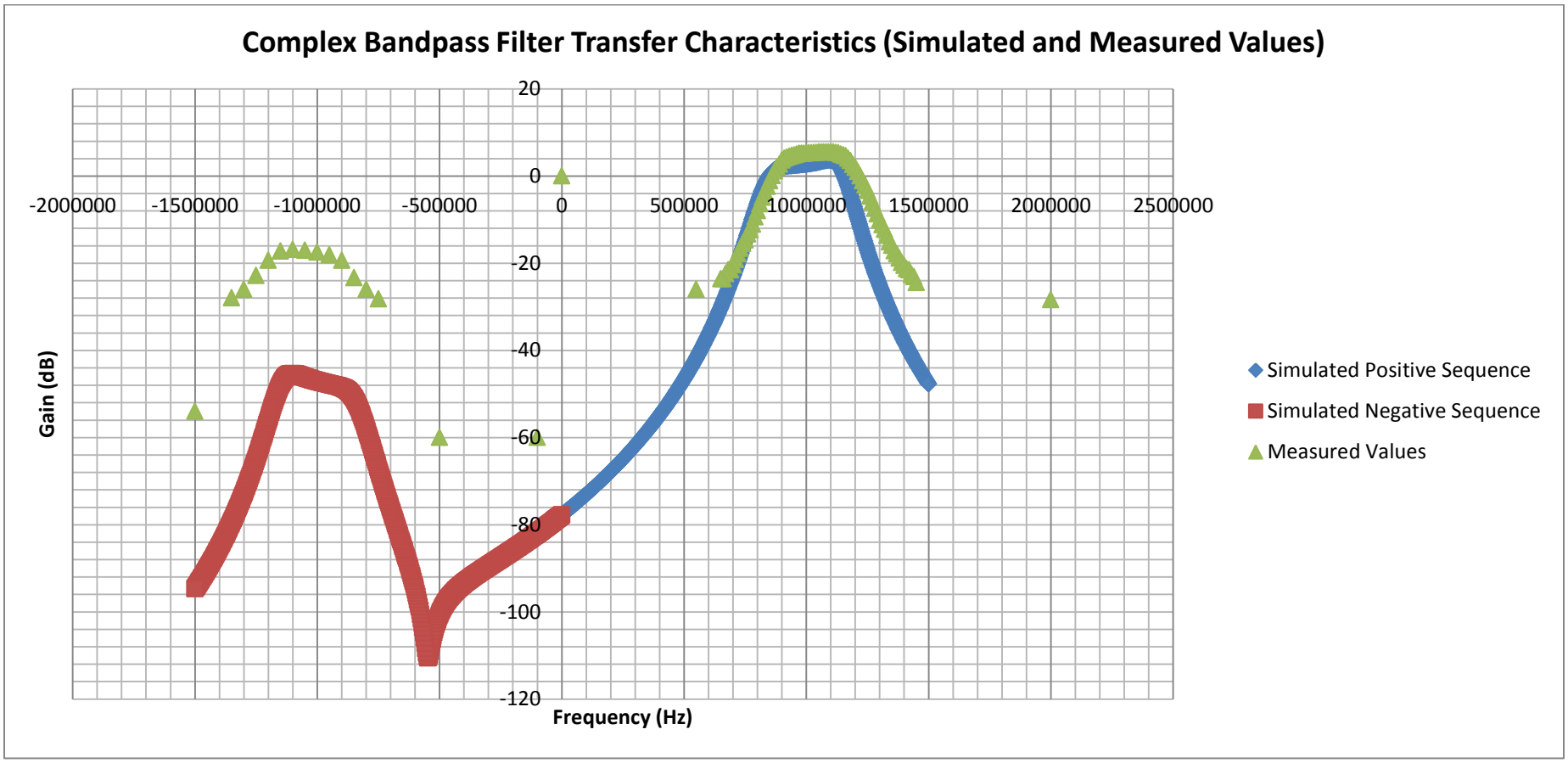

Figure 7.1.7 – Complex filter response with positive and negative sequences shown.

The negative sequence response is rejected with a minimum attenuation of -22 dB. The negative sequence, when viewed with an oscilloscope, was indeed a 1 MHz signal. Therefore, it can be concluded that the image reject filter is working and successfully rejects an image located to the right of the local oscillator for high side LO injection. It must be noted that the image attenuation was not as simulated; this is due to I and Q channel phase offset and amplitude characteristics mismatch. However, this filter is still within specifications.

Sub-System 2. - ADC Testing

The ADC was first tested by simply feeding a 1 MHz signal into the ADC. A 5 MHz clock was fed to the ADC and the outputs were polled by a logic analyzer. The waveforms are shown in Figure 7.2.1.

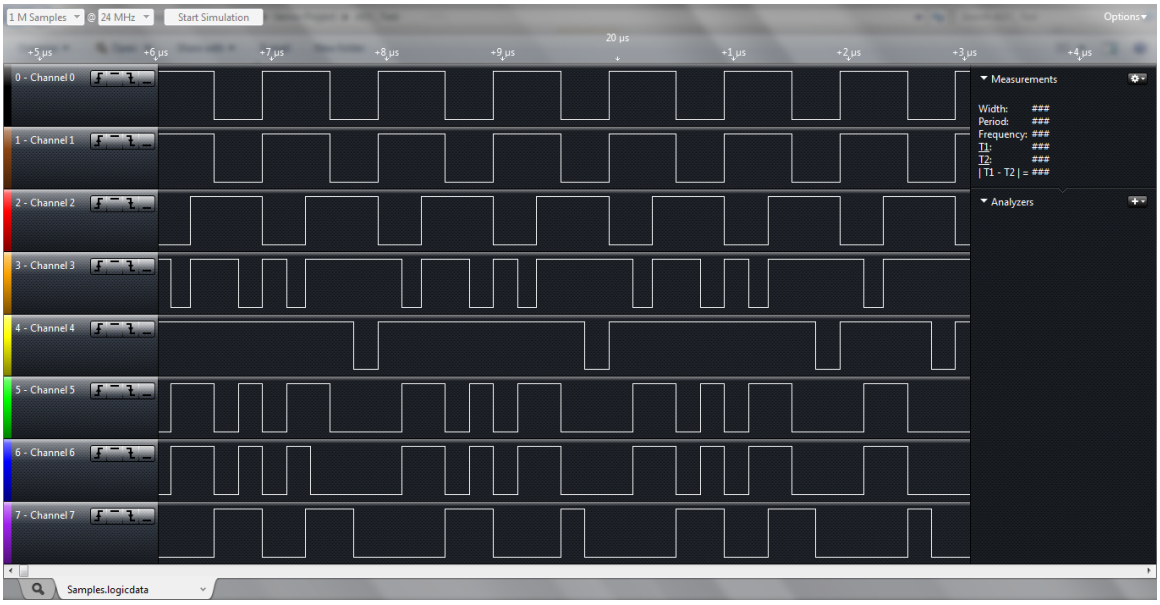

Figure 7.2.1 – Logic Analyzer output with Channel 0 as the MSB. The analyzer only had 8 bits so the 2 least significant bits were left out of the test.

The MSB clearly indicates a 1 MHz polarity switch indicating correct operation. After decoding the output of the ADC, the following normalized graph was obtained for the first 16 samples:

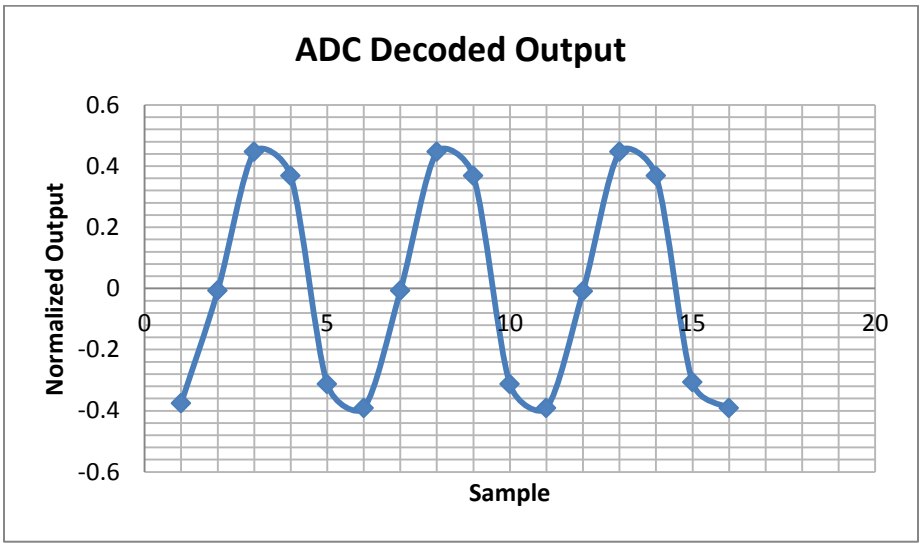

Figure 7.2.2. – Observe that sampling at 5 MHz is more than enough to reconstruct the original waveform.

Following these tests, the ADC outputs were then fed through the Nexys board which sampled the ADC and, upon every clock falling edge in a process statement, were then fed back to the outputs of the Nexys board and sampled as can be seen in the following figure.

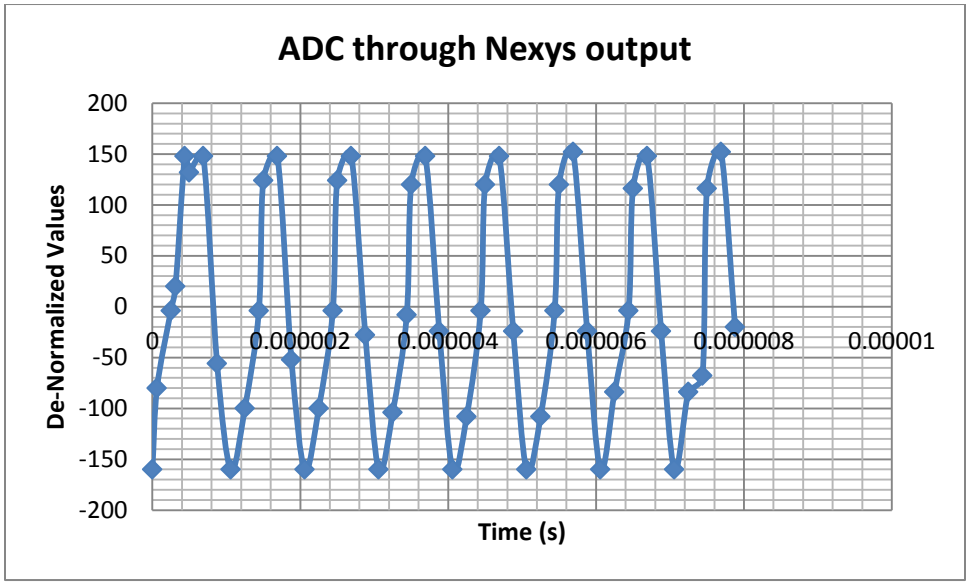

Figure 7.2.3 – ADC values after being fed through the Nexys board.

Some of the points in the figure above appear to deviate from their proper position. This is due to noise generated by the clock line which causes occasional bit errors. However, these errors happened relatively rarely and were ignored for the time being. The errors were most likely caused by improper routing because the clock line was routed underneath the ADC; it should have been isolated from the ADC and separate from the data and any other analog lines.

Sub-System 3. Instrumentation Amplifier through ADC Test

To test the IA through the ADC, only either the I or Q channel was used. Since the input is differential, one input was just tied to a CMV voltage of 1.5V and the other had a 2Vpp 100 KHz sine wave inserted into it. The output of the ADC is shown in Figure 7.3.1 on the next page.

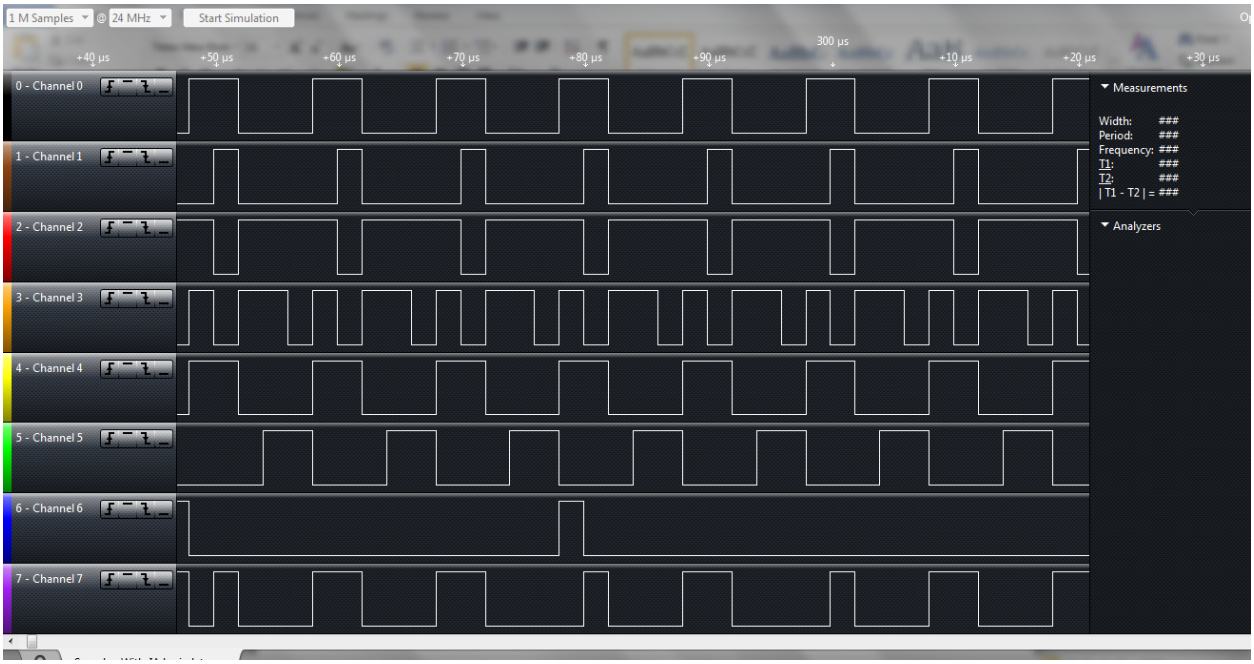

Figure 7.3.1 – Logic Analyzer output with the input at the IA.

The measured sign bit change is 100 KHz indicating that the ADC is reading data properly. Both channels behaved identically so the Q channel result picture has been left out.

Sub-System 4. – Modulator Testing

To test the modulator, a simple 150 KHz differential square wave was fed to the input and the RF was fed to a spectrum analyzer. The LO was set to 900 MHz at 0 dBm. Figure 7.4.1 shows the resulting spectrum.

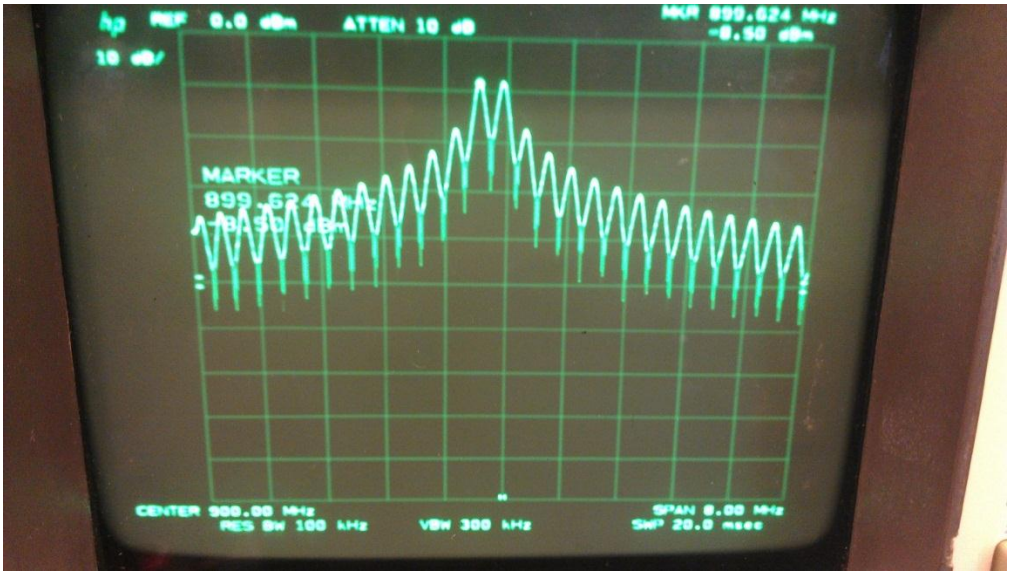

Figure 7.4.1 – Spectrum of the modulated 150 KHz square wave.

Without pulse shaping applied to the system, the signal will occupy a large bandwidth as can be seen in Figure 7.4.1. However, Figure 7.4.1 does indeed demonstrate that the signal is being modulated to 900 MHz and that the LO is suppressed at 900 MHz. The sidebands follow a sinc function which is indeed the Fourier Transform of a 150 KHz square wave.

Sub-System 5. – Programmable Gain Amplifier Testing

To test the PGA, a sine wave signal with a 2.5 CMV was inputto the PGA and its amplitude was varied. The output of the PGA was observed in an oscilloscope. Refer to Figures 7.5.1, 7.5.2, and 7.5.3 for selected input and output waveforms. Because of the reset process described in Section 5, Sub-System 4, Software Component 7, the signal did indeed get passed through with a gain of 1 for a brief instant in time. This time depends on the time it takes for the comparator process to run.

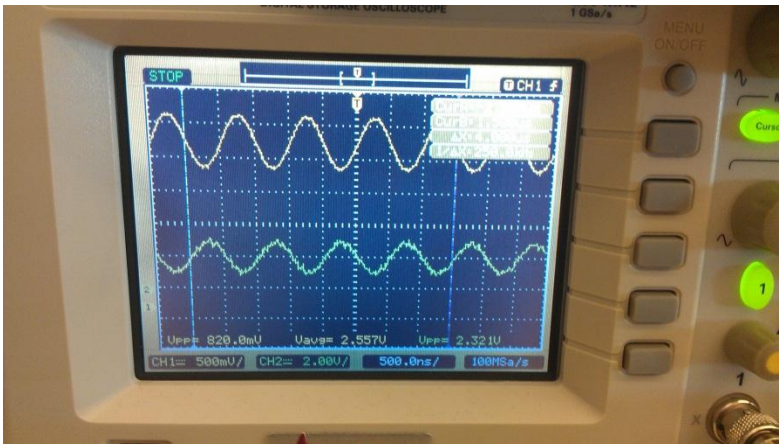

Figure 7.5.1. – Output waveform (green) for a .82 Vpp 2.5V CMV sin wave input (yellow).

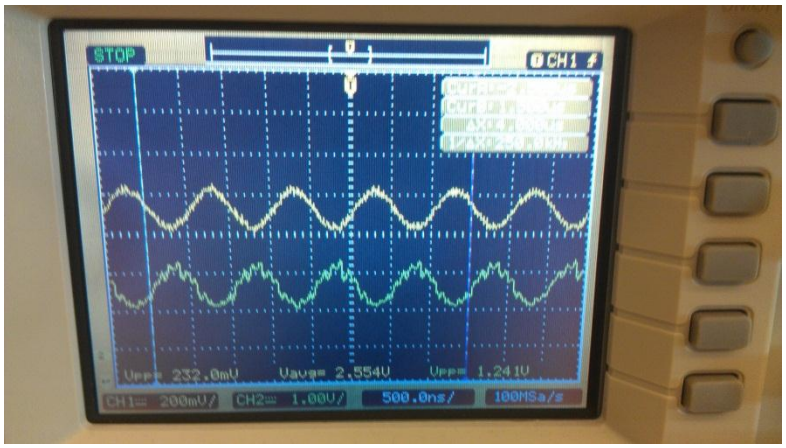

Figure 7.5.2. – Output waveform (green) for a .23 Vpp 2.5V CMV sin wave input (yellow).

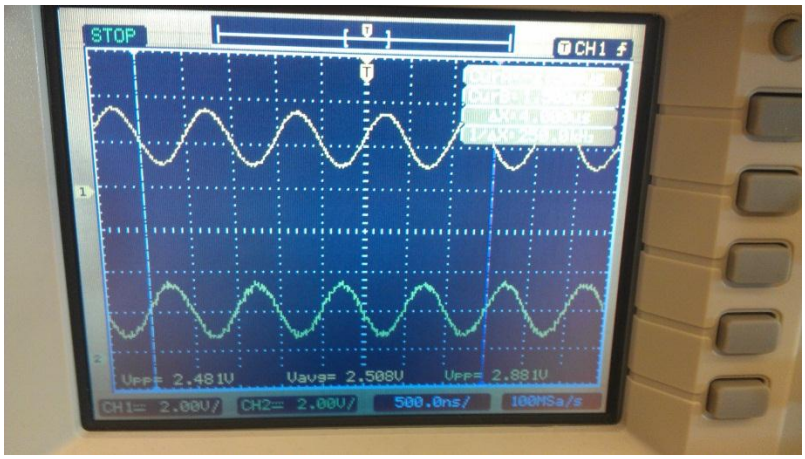

Figure 7.5.3. - Output waveform (green) for a 2.48 Vpp 2.5V CMV sin wave input (yellow).

Table 7.5.1 shows the gains of each of the previous waveforms and compares them to the nominal gains that correspond to the respective states given in Figure 5.5.7.2.

Vin (Vpp)	Vout (Vpp)	Gain	State	Nominal Gain	% Error
0.82	2.32	2.829268293	5	3	-5.69106
0.232	1.24	5.344827586	7	6	-10.9195
2.48	2.881	1.161693548	1	1	16.16935

Table 7.5.1. – Selected waveforms and their respective gains and states compared with nominal values.

The percent errors shown for these states are relatively high. However, they are within reason and we can assume that the PGA works. It must be noted that the oscilloscope somewhat struggled to obtain a proper waveform picture because of the resetting nature of the algorithm, therefore, some measurements may be innacurate.

Section 8. FULL SYSTEM TEST

With all the sub-systems tested and working, the system was ready to be integrated and tested together. For preliminary tests, the PGA was excluded. The first test was to establish a direct demodulation communication system by simply connecting the RF_IN and RF_OUT ports of the modulator and demodulator and setting both LO's to 900 MHz. The 900 MHz RF carrier was used as opposed to 950 MHz for arbitrary reasons. Absolutely no discernable difference was noticed between these two frequencies. With a 150 KHz square wave differential input, the I positive output of the demodulator is shown in Figure 8.1 below.

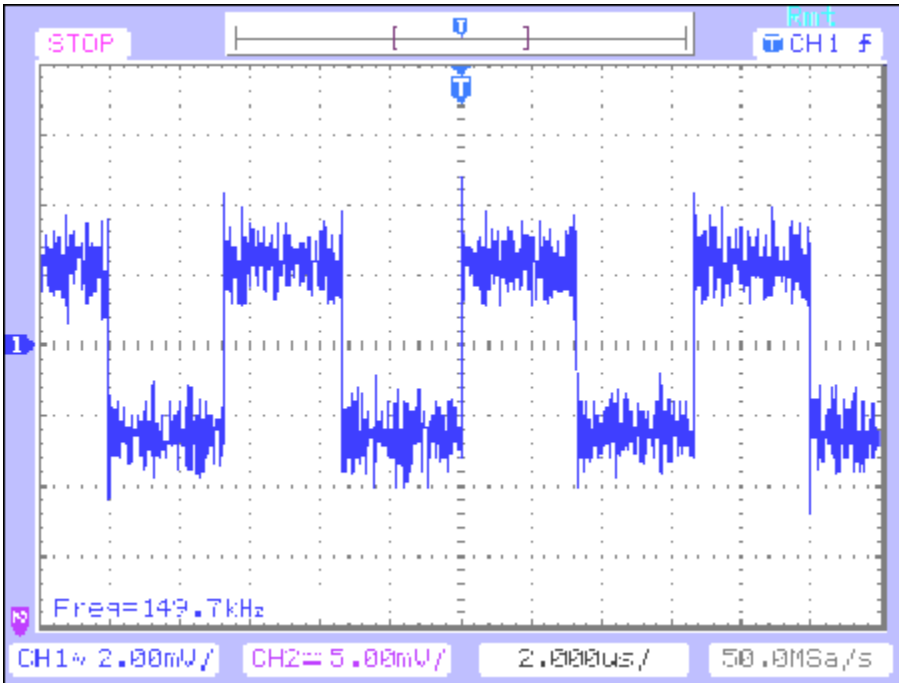

Figure 8.1. – Demodulated square wave using direct conversion.

The noise present in the wave is the unfiltered high frequency content associated with down conversion. This test confirmed that the demodulator does indeed down convert a signal properly.

By simply changing the LO frequency to 1 MHz higher than the RF frequency on the demodulation side, the system becomes a low IF architecture. The resulting demodulator output is shown on the next page in Figure 8.2.

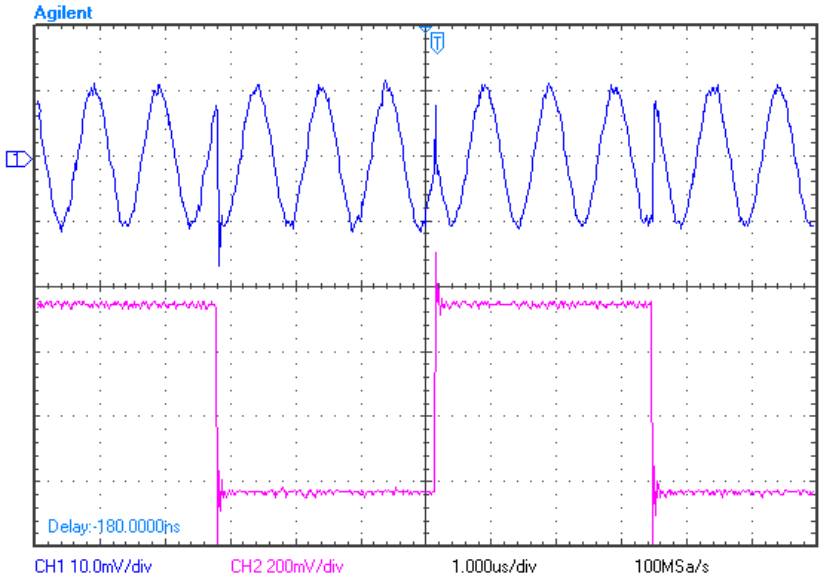

Figure 8.2. – The I channel output of the demodulator. The blue signal is the output and the pink signal is the square wave generated by the Nexys board.

It can be seen from Figure 8.2. that the low IF signal clearly is a BPSK signal; the phase changes for every bit change. This signal can be directly compared with the simulated waveform in Simulink previously shown in Figure 5.2.4. This abrupt change in phase contains a lot of spectral content, however, bandwidth is not of concern in this project. The signal after the filter can be seen in Figure 8.3 below.

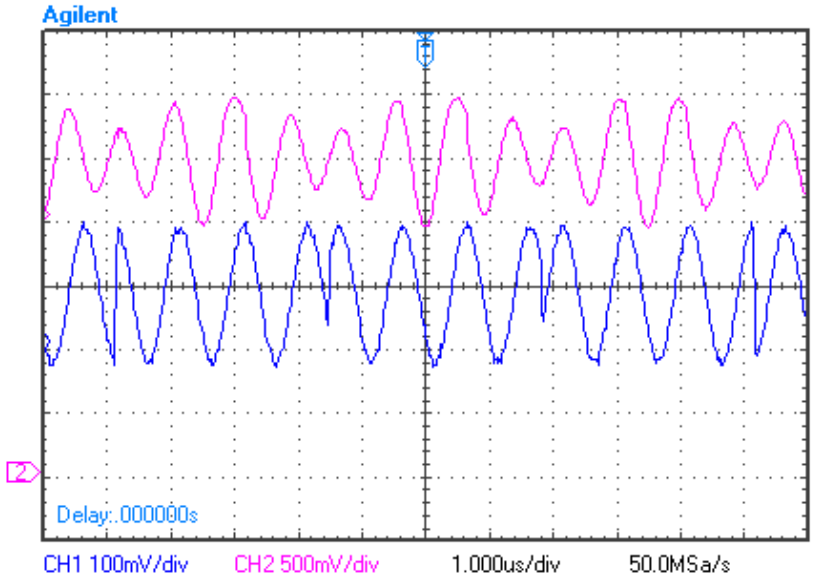

Figure 8.3 – The input (blue) to the filter and the output (pink)

The abrupt changes in phase are clearly filtered out. Without the PGA, the output of the filter was fed directly to the ADC and FPGA for final demodulation to baseband. For a square wave as the input data, the FPGA succesfully demodulated the data as can be seen in Figure 8.4 below.

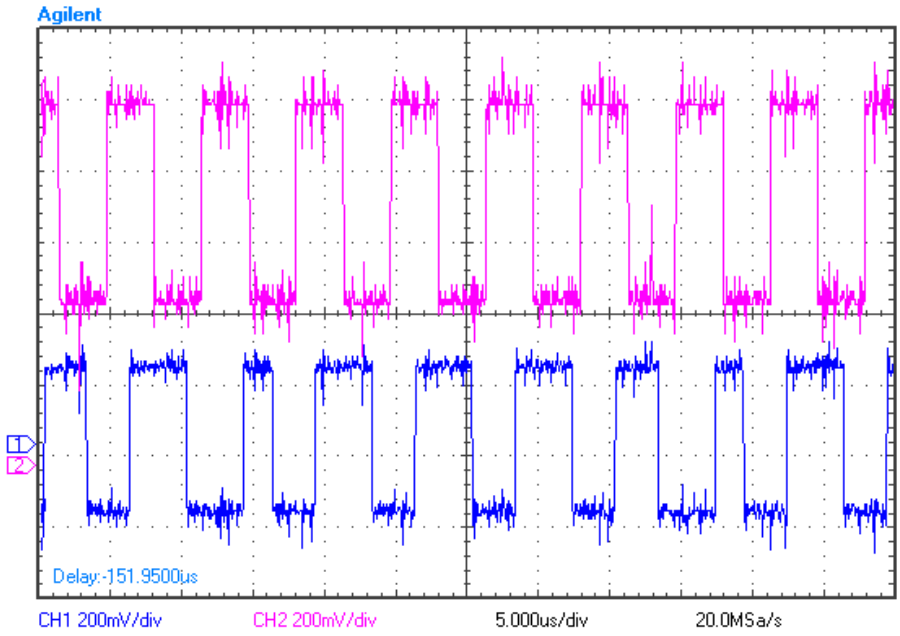

Figure 8.4. – Input data (pink) and demodulated data (blue)

It can be seen from Figure 8.4 that the data has indeed been demodulated, although at certain points the duty cycle appears to change. This can be attributed to the noise the ADC was experiencing as explained earlier. With a PRBS signal at 250 Kbps generated by the Nexys board, the resulting demodulated output is shown in Figure 8.5.

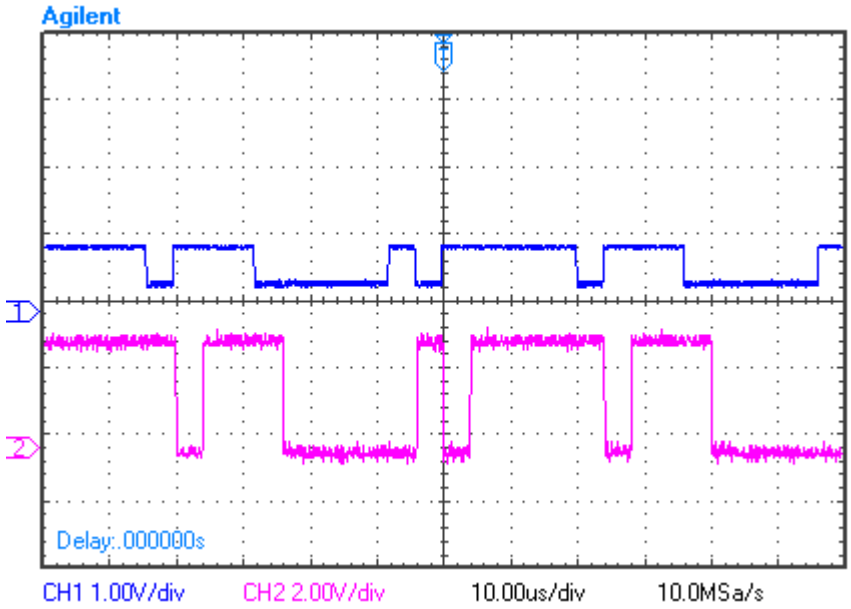

Figure 8.5. – Input PRBS data (blue) and the demodulated output data (pink)

More demodulated bits can be seen more clearly through a logic analyzer. Figure 8.6. shows the output waveforms from the logic analyzer.

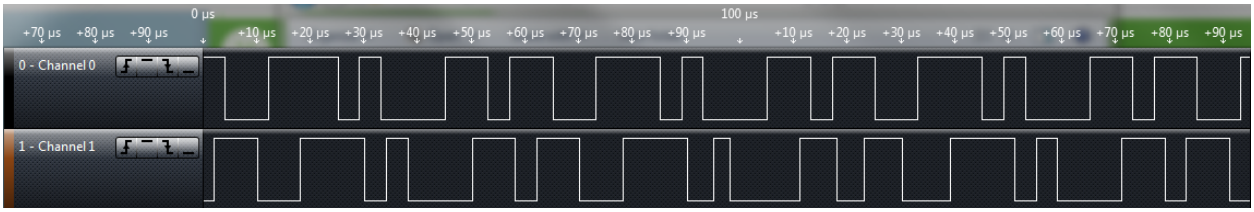

Figure 8.6. – Input data (Channel 0) and demodulated output data (Channel 1).

The delay between the two waveforms is roughly 5 us which corresponds to the 4.8 us of group delay associated with the image reject filter.

It is more evident in the PRBS data sequence in Figure 8.6 that the system indeed demodulates the signal properly. Further evidence is provided in Figure 8.7 when the LO from the input is turned off which results in no RF data being transmitted. This test shows that the Nexys-2 board is not just somehow picking up the signal generated in another part of the board and outputting that sequence.

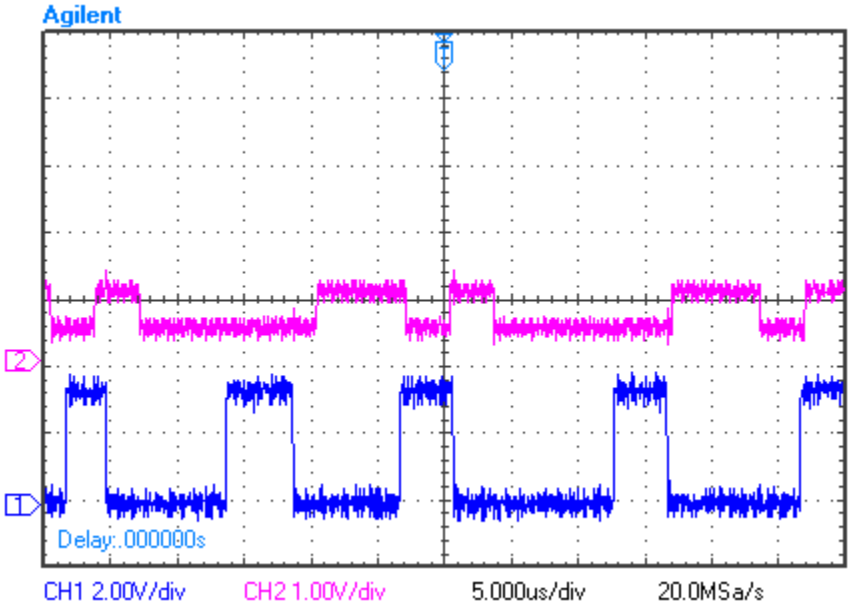

Figure 8.7. – Input data (pink) and output data (blue) with the modulator LO turned off.

The data the Nexys-2 outputs is a result of random noise associated with what the ADC is giving the borad.

With data being demodulated correctly and all other systems funcitonal, the PGA was ready to be integrated into the system in order to conduct BER sensitivity tests. Figure 8.8 shows the final test set up and Table 8.1 shows the results of the tests. A variable attenuator was used to conduct the tests.

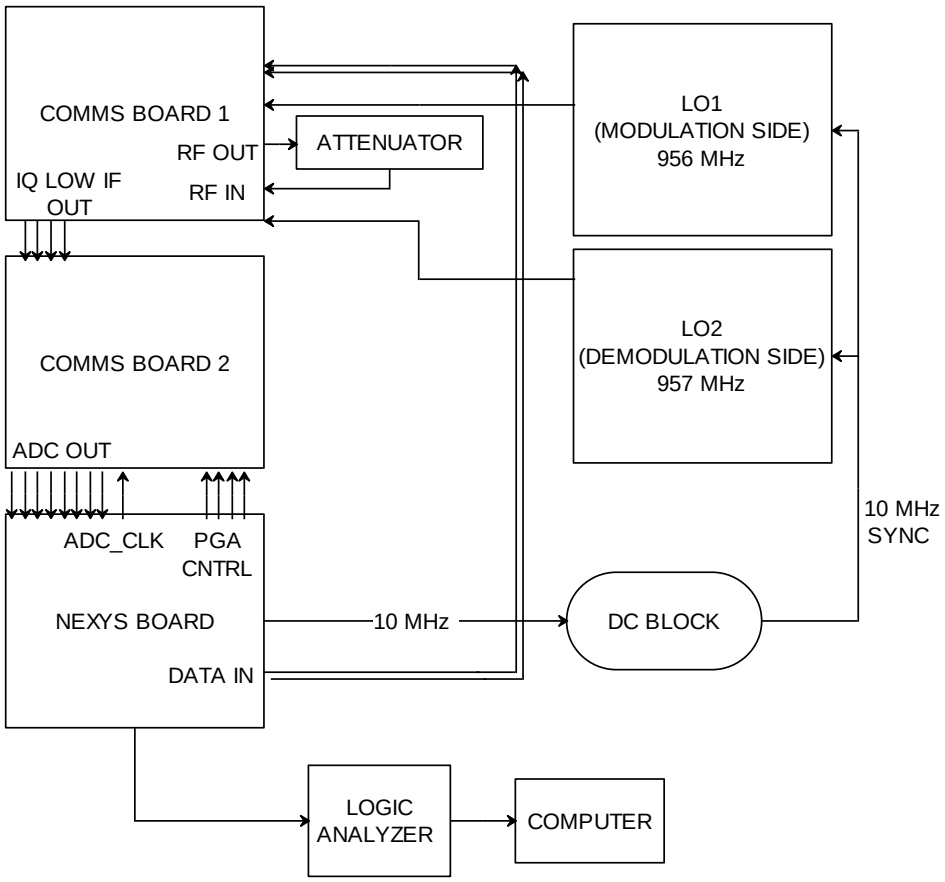

Figure 8.8. – Final test setup diagram.

	NO PGA		PGA	
Attenuation (dB)	BER (%)	Error Bits	BER (%)	Error Bits
0	0	0	0	0
2	0	0	0	0
4	0	0	0	0
6	0	0	0	0
8	0	0	0	0
10	0	0	0	0
12	0.01	1	0.01	1
14	0.02	2	0.07	7
16	0.07	7	0.15	15
18	0.11	11	0.23	23
20	0.23	23	0.32	32

Table 8.1. – Sensitivity test results.

Section 9. – RESULTS ANALYSIS/CONCLUSION

From Table 8.1 we can clearly see that the BER increases as the attenuation increases. However, the BER increase even more with the addition of the PGA. It must be noted that when tests were performed the previous day, this was not the case. The PGA only started seeing bit errors at 14 dB the previous day as opposed to 12 dB when the usable data was collected. The data collected the previous day was stored in excel files, however, due to the delay between the input and output signals, the logic analyzer exported data that had a mismatch of 0's and 1's for a given bit period between the channels. Thus, all the data collected was rendered uesless. It must also be noted that only 100 bits were analyzed and therefore the required BER of 1E-4 could not possibly be measured using this method. Thus, the data shown is for demonstrative purposes only.

Despite the fact that BER could not be measured properly and that the PGA failed to work for the last test, the receiver did indeed successfully demodulate a BPSK 956 MHz modulated 250 Kbps data stream and reject the alternate channel by more than 20 dB. The final top level specifications of the project are found in Figure 9.1 below.

Supply Voltage	5V
Supply Current	400 mA
Sensitivity for 1% BER	-20 dBm

Figure 9.1. – Basic top level specifications of the receiver.

Significant additions and improvements can be made to this project in the future. These improvements are as follows:

1) The addition of a PLL and a clock extraction system.

   Because there was no PLL to lock the phase of the LO to the incoming RF signal and because there was no clock extraction algorithm implemented in the FPGA, the receiver was severely limited in its uses. It relied on external local oscillators and syncing signals to properly demodulate data. By implementing the aforementioned additions, the receiver can act as a stand alone device and not rely on external machines.

2) Improved image reject filter.

   Although the image reject filter performed to spec, improvements can be made to the cost of the filter, the realestate it uses, the image reject quality, and the power it consumes. The filter itself costed 130 dollars mainly due to the expensive op amps used. Also because the filter used completely off the shelf components, it drew a lot of current (130 mA) and took up a lot of space. All of these problems can be solved by integrating the filter into a circuit using advanced CMOS technology. In addition, by implementing the design in an IC, the phase offset and I/Q channel mismatch can be more properly controlled.

3) Improved PGA.

   The PGA used in the present design did not significantly improve the BER. It also did not have a wide dynamic range nor did it have a very linear gain sweep. Therefore, for future designs a PGA with a larger multiplexer array should be used with more appropriately chosen resistors.The calibration of states in the PGA algorithm should be more linear and evenly spread out as well. In addition to this, further calibration of what the ADC outputs for a given input signal strength should be performed to make the system more robust.

4) Improved signal routing.

   The PCB traces for the ADC were not optimally routed. The data sheet specifically mentioned that the clock route be routed away from all other signals. However, the clock route was run directly under the IC itself and was near all the data signals. In addition to the ADC, the filter I and Q channel routing can be more exact to avoid I and Q mismatch (if the filter will be built on a PCB again).

Section 10. BIBLIOGRAPHY

[1] J.Crols and M. Steyaert, "An Analog Integrated Polyphase Filter for a High Performance Low-IF Receiver," Katholieke Universiteit Leuven, Heverlee, Belgium, 1995

[2] A. Ambardar, *Analog and Digital Signal Processing*, Second Edition. Pacific Grove, CA: BROOKS/COLE PUBLISHING COMPANY , 1999.

[3] IEEE Std. 802.15.4 2009 http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf

[4] B. P. Lathi, *Modern Digital and Analog Communication Systems*, Third Edition. New York: Oxford University Press, 1998.

APPENDIX A.

ABET SENIOR PROJECT ANALYSIS

Project Title: Low IF Digital Radio Receiver

Student Name: Sanjay Avasarala **Student Signature:**

Advisor Name: Professor Wayne Pilkington **Advisor Signature:**

1) **Summary of Functional Requirements:**

The radio shall accept a 956 MHz signal with modulated BPSK 250 Kbps data as an input and successfully demodulate and output the data.

2) **Primary Constraints:**

Because of various constraints such as the lack of a direct down conversion (zero IF) circuit that requires a specific crystal oscillator or because of 1/f noise inherent in MOSFETS, the use of low IF is highly desirable for low noise receivers. Therefore the demodulation should be performed at low IF which warrants a more complicated circuit that is capable of demodulating data at a higher frequency.

Another constraint for the project is the cost associated with research and development. Radios that use DSP's as opposed to traditional analog radios are relatively expensive (some of the cheapest can be as expensive as $40) and thus the component choices used in the circuit require careful attention.

For this particular project, the image reject filter used op-amps that cost 3 dollars each, which ideally is not practical. However, if the entire circuit was fabricated in an IC which it is meant to be, then the cost will decrease significantly.

3) **Economic:**

Due to the many electronic silicon components involved in this circuit, several economic factors have to be taken into consideration. Ordering the parts from electronics distributors such as DigiKey will have a direct impact on the company's net profit. Also, the workers at the company will be directly involved since the selection of parts and the quantity of each part is handled manually. The actual shipping of parts also has an impact on the shipping industry's financial situation. This in turn also affects oil prices and brings into question environmental impacts that will be discussed later. The parts ordered for this project also have an indirect affect on the actual manufacturers of the part itself. A significant rise in demand will warrant an increase in supply. However, all the aforementioned economic factors are rather miniscule with regards to the scope of this project. They are worth mentioning though, should large scale manufacturing occur.

The majority of the costs of this project will lie in the purchasing of components. Since test equipment is provided by the school, this is of negligible cost.

The project should not cost more than $450.00 to design build and test. This does not include the research material needed to design the project. The funding for this project will come from personal funds.

Each receiver unit should have a net profit of $10.00. The project costs are not what the actual end product will cost to build since the project costs include development boards and kits which cost a significant amount.

The product shall be designed built and tested by the end of Spring Quarter 2012. A second revision may be implemented should sufficient demand warrant it.

Please refer to Table A.1 (below) for an estimate of costs. Note that optimistic costs should not cost more than $450.00 for physical costs as specified in Table 3.1.

Cost Estimates

	Optimistic	Pessimistic	Realistic
Labor (15hr/week)($10/hour)25weeks	$7,500	$7,500	$7,500
DSP (includes dev kit)	$100.00	$350.00	$200
Analog Parts (including shipping from Digikey)	$50.00	$200.00	$100
Board Layout	$33.00	$50.00	$50
SUM	$7,683	$8,100	$7,850
Variable Costs			
Board Revision	$0.00	$100.00	$200
Parts Revision	$0.00	$50	$100
DSP switch out	$0	$350	$400
Computer Interface Peripheral	$50	$200	$400
SUM	$50	$700.00	$1,100
TOTAL	**$7,733**	**$8,800.00**	**$8,950**

Table A.1. – Estimated costs.

Using Equation (6) in Ford and Coulston Chapter 10, the total projected cost is therefore **$8647.00.**

The actual costs of the project are shown in Table A.2.

ITEM	COST
Components (1)	$55.56
Components (2)	$120.78
Components (3)	$15.02
FPGA	$90
Boards	$166
TOTAL	**$448.52**

Table A.2. – Actual costs.

As can be seen from Table A.2, the actual costs of the project just barely met the costs constraint established in the requirements and specifications. Please refer to Table A.3 for the specific bill of materials. Note that the bill of materials does not contain the board or FPGA costs which are specifically listed in Table A.2.

Index	Quantity	Description	Unit Price	Total Price
10	8	CONN SOCKET SMA R/A DIE CAST PCB	3.33	$26.64
11	10	RES 100 OHM 1/8W 5% 0805 SMD	0.04	$0.40
12	50	RES 10K OHM 1/8W 5% 0805 SMD	0.0162	$0.81
13	50	RES 100K OHM 1/8W 5% 0805 SMD	0.0162	$0.81
14	10	RES 1.2K OHM 1/8W 5% 0805 SMD	0.04	$0.40
15	50	RES 910 OHM 1/8W 1% 0805 SMD	0.0276	$1.38
16	10	RES 820 OHM 1/8W 5% 0805 SMD	0.04	$0.40
17	50	RES 1.0K OHM 1/8W 5% 0805 SMD	0.0162	$0.81
18	50	RES 160 OHM 1/8W 5% 0805 SMD	0.0162	$0.81
19	1	IC DIRECT QUADRATURE MOD 16-QFN	10.7	$10.70
20	10	CAP CER 0.1UF 25V 10% X7R 0603	0.026	$0.26
21	10	CAP CER 5.6PF 50V NP0 0603	0.029	$0.29
22	10	CAP CER 4.7PF 50V NP0 0603	0.029	$0.29

23	50	CAP CER 1000PF 50V 10% X7R 0603	0.0204	$1.02
24	3	CAP CER 2.2UF 16V Y5V 0603	0.28	$0.84
25	10	CAP CER 10PF 50V NP0 0603	0.036	$0.36
26	10	CAP CER 10000PF 50V 10% X7R 0603	0.023	$0.23
27	10	CAP CER 1UF 10V 10% X5R 0603	0.052	$0.52
28	4	IC MULTIPLEXER 4X1 10MSOP	2.84	$11.36
1	40	IC OP AMP 3.4MA 215MHZ SOT23-6	2.7388	$109.55
2	10	CAP CER 1UF 10V 10% X5R 0603	0.052	$0.52
3	4	CAP CER 10UF 6.3V 20% X5R 0603	0.43	$1.72

Table A.3. – Bill of Materials from Digi-Key

Please refer to Figure A.1. for a Gantt chart timing diagram of the project life cycle. Since the project timeline followed the Gantt chart quite closely, only one chart is needed to describe the life cycle. If manufactured, a receiver should be able to last indefinitely until the entire system it is installed in is recycled. At the end of the project life for this senior project specifically, more improvements will be added for a summer project.

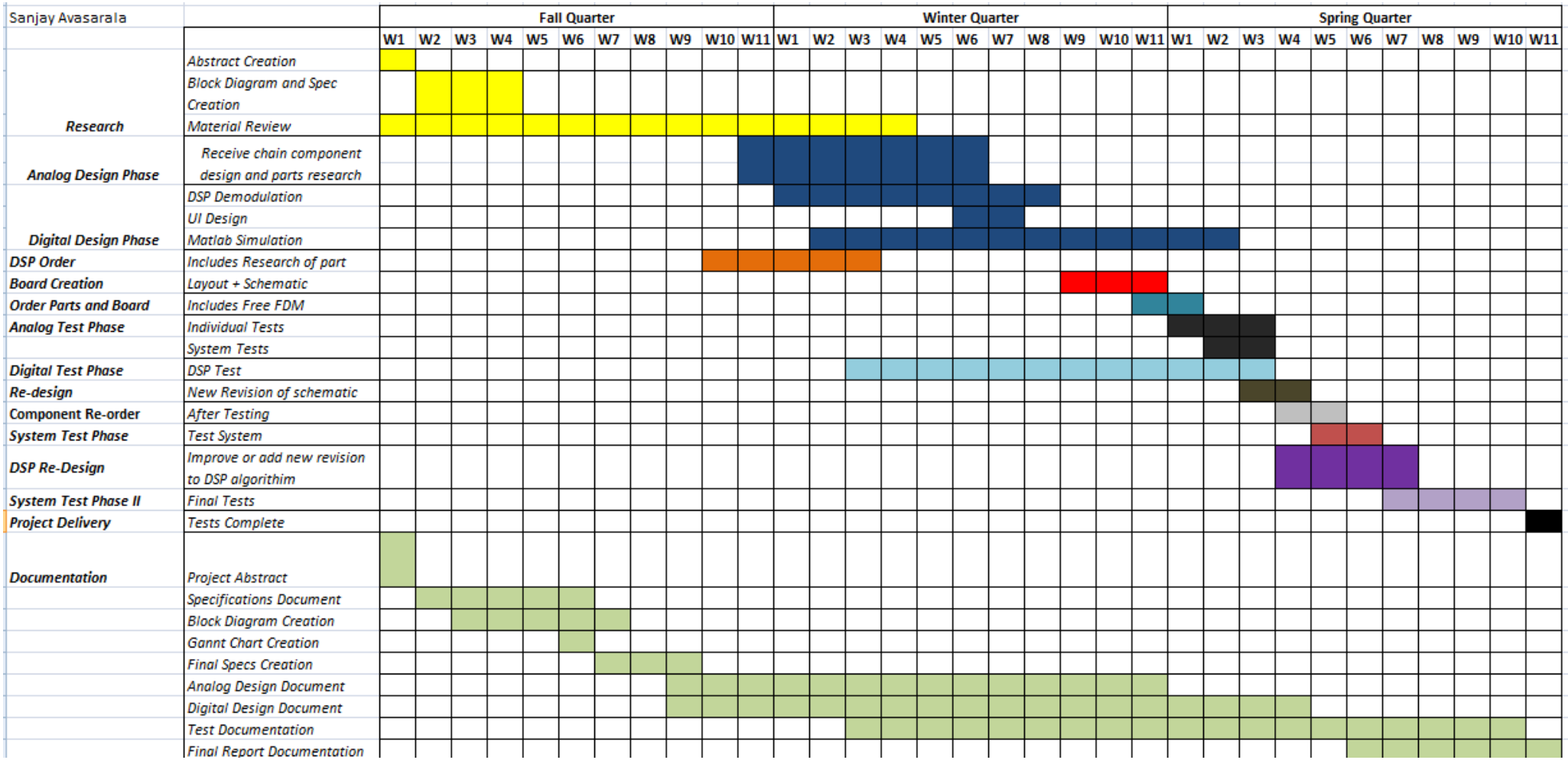

Figure A.1. – Gantt Chart Timeline

4) Commercial Manufacturing:

If manufactured, as many as 500 products may be sold per year. There exists an increasing demand for software defined radios since they are much more customizable than traditional analog radios. Should this radio enter mass production, manufacturing costs should not exceed $7.00 per unit. Purchasing components in bulk can ensure that this requirement is met. Each device should cost no less than $17.00 per unit. Estimated profits would therefore be $5000 per year. The cheapest software defined radios that are readily available for consumer use are as cheap as $40.00. Building a radio that costs $17.00 is a significant step towards low cost software defined radios. Estimated cost to operate device per unit time depends strictly on how the device is being powered e.g. mains power or battery operated. There are, of course, environmental issues associated with using batteries.

5) Environmental:

Environmental impacts include the manufacturing of the components and their packaging, the shipping of the components, and the storage of the components. The use of silicon, germanium, and other semiconductor materials are directly involved in this project along with other materials such as FR-4, copper, and solder (Lead and Tin). The mining and harvesting of these materials will have a definite environmental impact.

Since this receiver is more power efficient than earlier receiver designs, long run environmental impacts made by this radio receiver over an older radio are less.

Should the receiver utilize batteries, consideration should be given to the type of battery used and the environmental impacts that battery creation and disposal has. The disposal factors heavily outweigh the creation factors and remain as one of the biggest issues with batteries today. However, there are companies that are solely devoted to battery disposal treatment in a safe, eco-friendly way. Mains power is, of course, not used if batteries are used. Therefore there are some positive factors to batteries in the environmental sense.

6) Manufacturability:

Some issues involved with manufacturing, in the event that this product enters mass production, include pick and place machining, packaging, electricity, gas, and water costs, worker labor costs, and facility and storage costs. Like all manufacturing, a process has to check the product for defects and other issues before shipping. This incurs additional costs and machining. In addition, management and other bureaucratic affairs are all necessary to manage a successful product development process.

7) Sustainability:

The completed device will have to run solely on battery power or whatever power source the entire system provides. Therefore, a renewable energy source would be the most viable source for long term operation. The project has no long term effect on the sustainable use of resources with the exception that, should the project be mass produced, consideration will have to be placed on manufacturing materials. The circuit could be improved by using state of the art technology to alleviate power dissipation problems and other limiting factors associated with older technology. Ideally, fabrication of this project in the form of an integrated circuit is the most viable solution to an environmentally friendly yet optimized product. However, the fabrication of an IC for small scale manufacturing processes is not a cost effective solution.

8) Ethical Implications:

As long as the device conforms to the IEEE Zigbee standard and 802.15.4 standards set by IEEE, there are no physical (in the sense of physical harm) ethical implications. Since the device is *purely* a receiver, transmitting power is completely ignored. Transmit power is strictly regulated by the FCC. Should the project contain a transmitter as well, the transmit power will need special consideration.

However, issues arise should one use the device for illegal activities such as hacking or stealing. One may choose to use the device to intercept private transmissions. There is absolutely no way to regulate this misuse of the device since directional radio finding will not find a device that is used only for intercepting a signal. However, the chance that one might use the device for data interception is extremely slim since devices that conform to the Zigbee standard are short range (mostly domestic) devices.

9) Health and Safety:

This topic will avoid indirect health and safety issues. The use of lead in solder and the dangers inherent in soldering are all considered a safety issue. Skin burns and lead inhalation are just some of the issues associated with soldering. Lead that enters the bloodstream has the potential for physical damage. Once again, since the project is only a receiver, the health effects of transmit power are ignored. A receiver broadcasts absolutely nothing.

10) Social and Political:

The device must conform to international laws that deal with ISM-Band transmissions and the Zigbee standard. Strict export laws to other countries may prevent this issue. However, for satellite communications, for example, one has to pay special consideration to ISM-Band regulations maintained by other countries.

Stakeholders in this project are non-existent at this moment. This project is being funded personally and no one stands to benefit monetarily from it.

11) Development:

This project extensively requires the use of SIMULINK during the project to simulate the demodulation of data and also the operation of various receive chain systems. MATLAB is also used to observe frequency and phase responses of receive chain systems and ensure that proper stability is reached. This project also requires the review of DSP.

APPENDIX B: SCHEMATICS

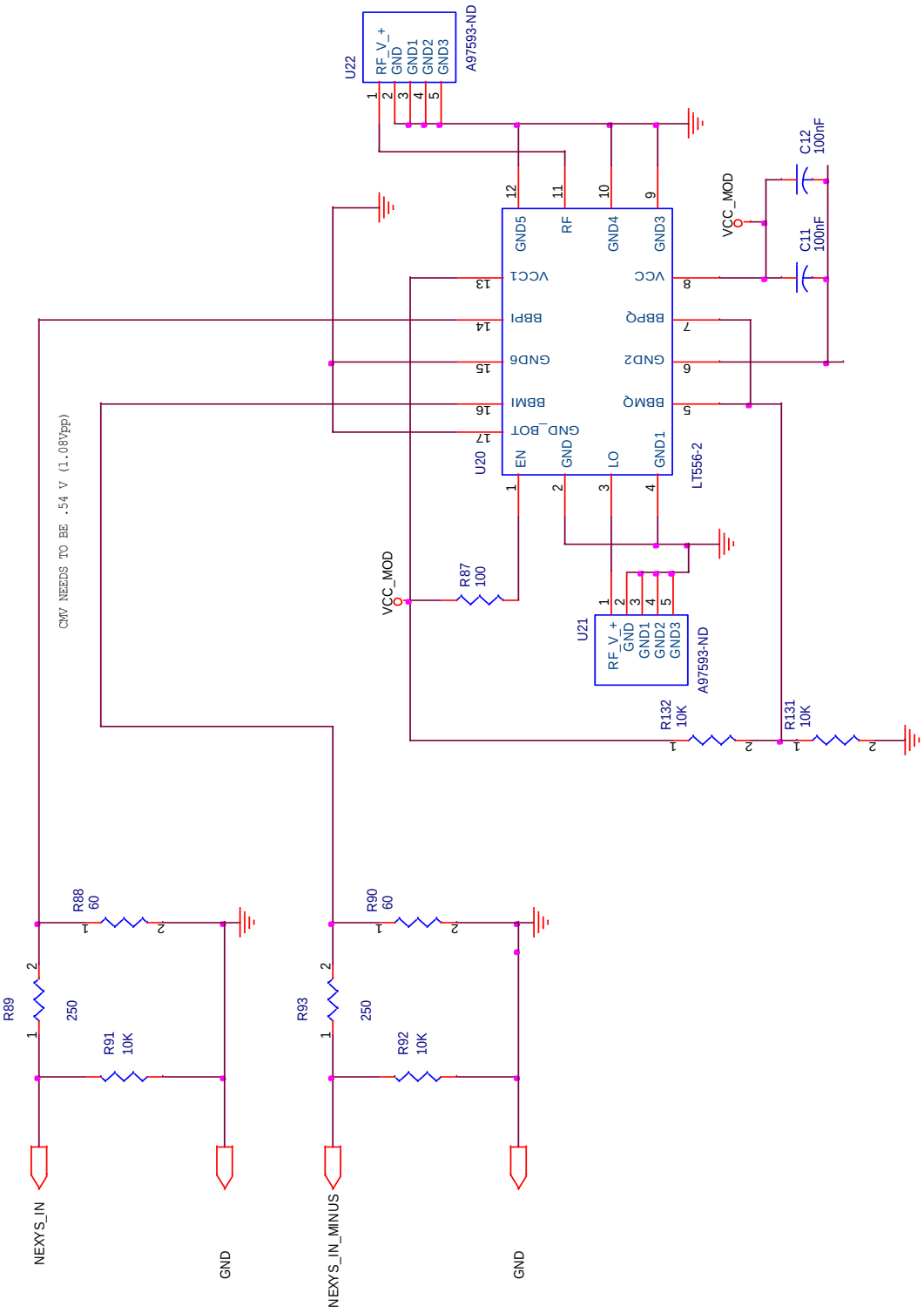

Figure B.1. – Modulator Schematic

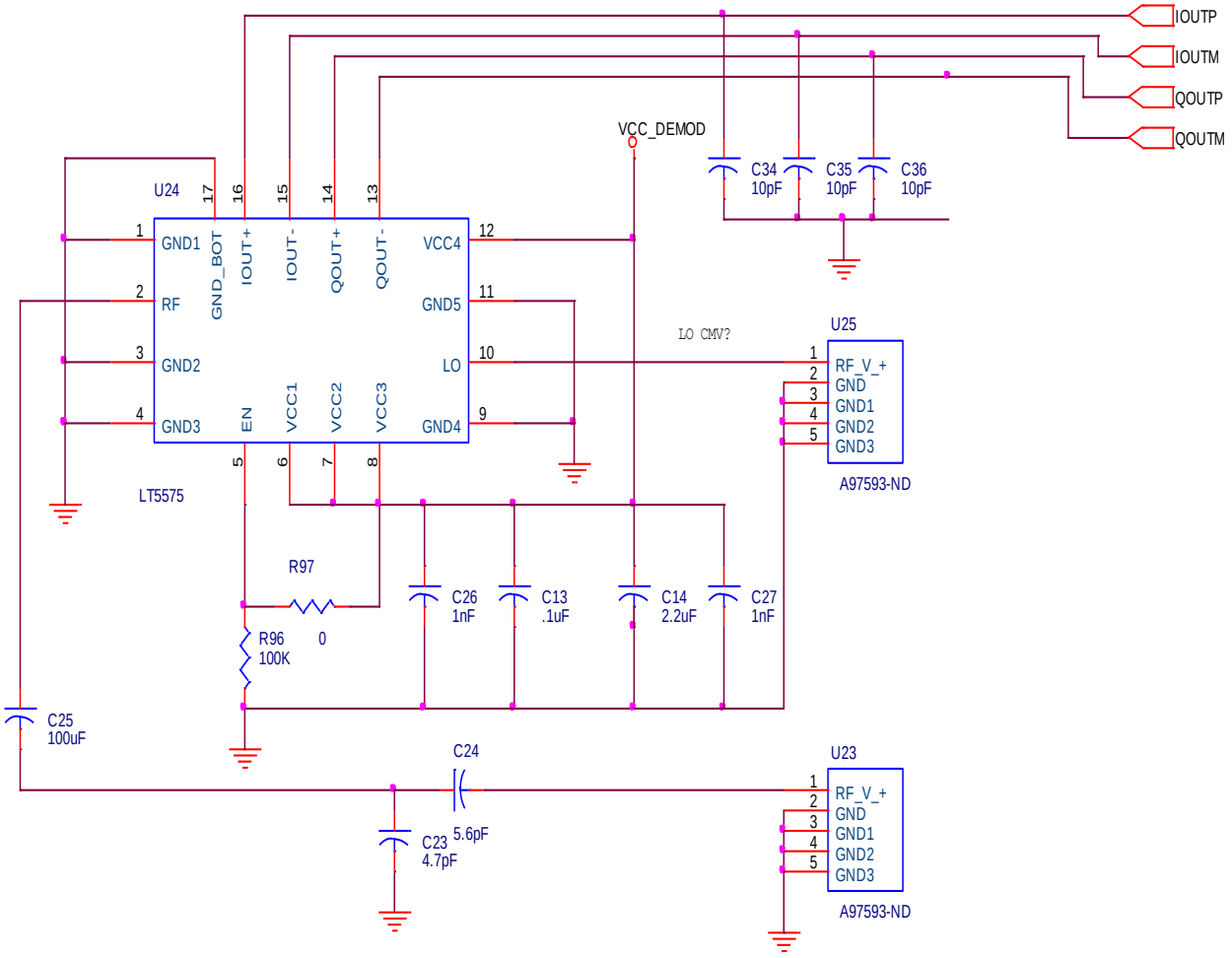

Figure B.2. - Demodulator

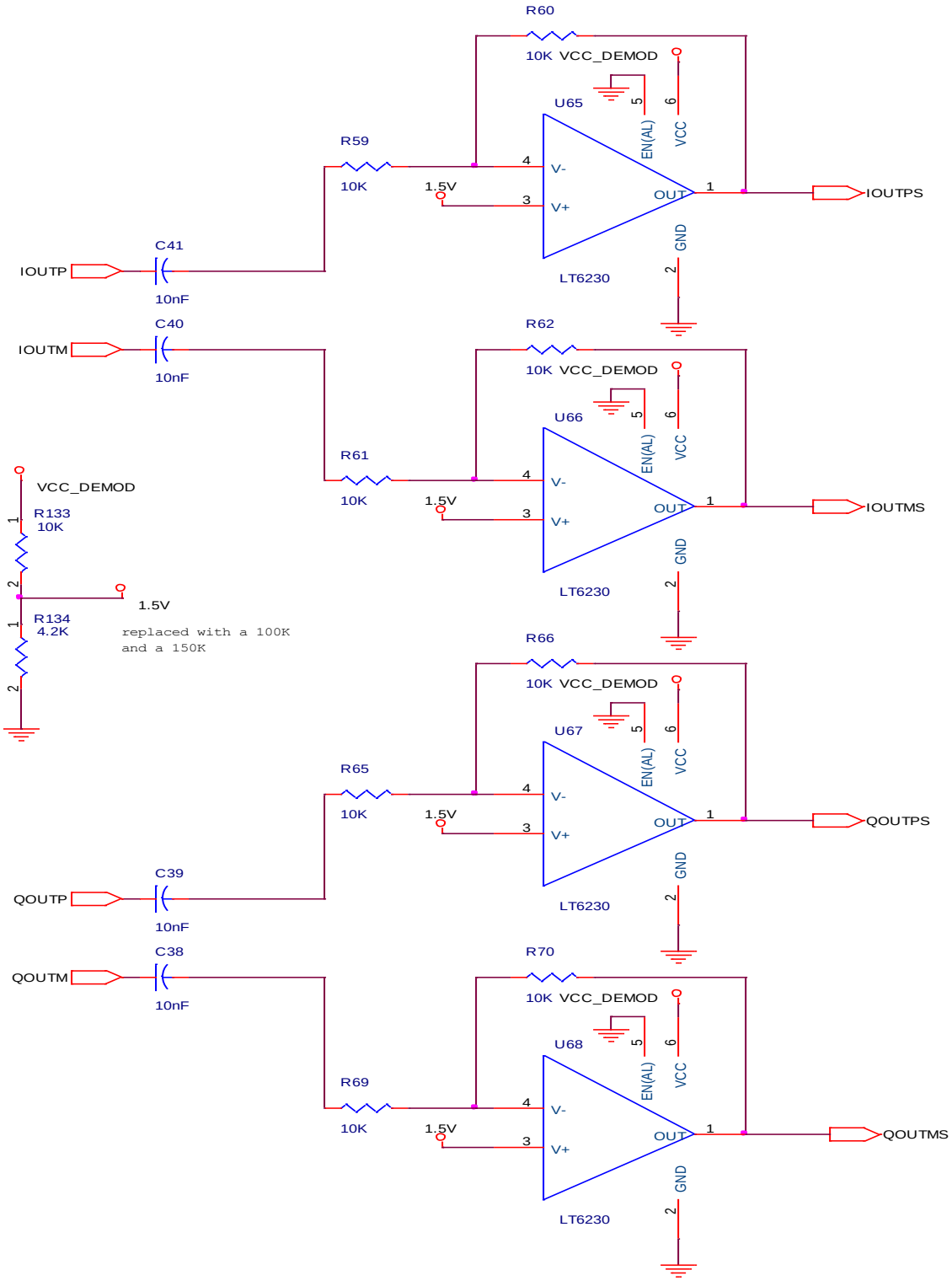

Figure B.3. – CMV DC offset shift circuits

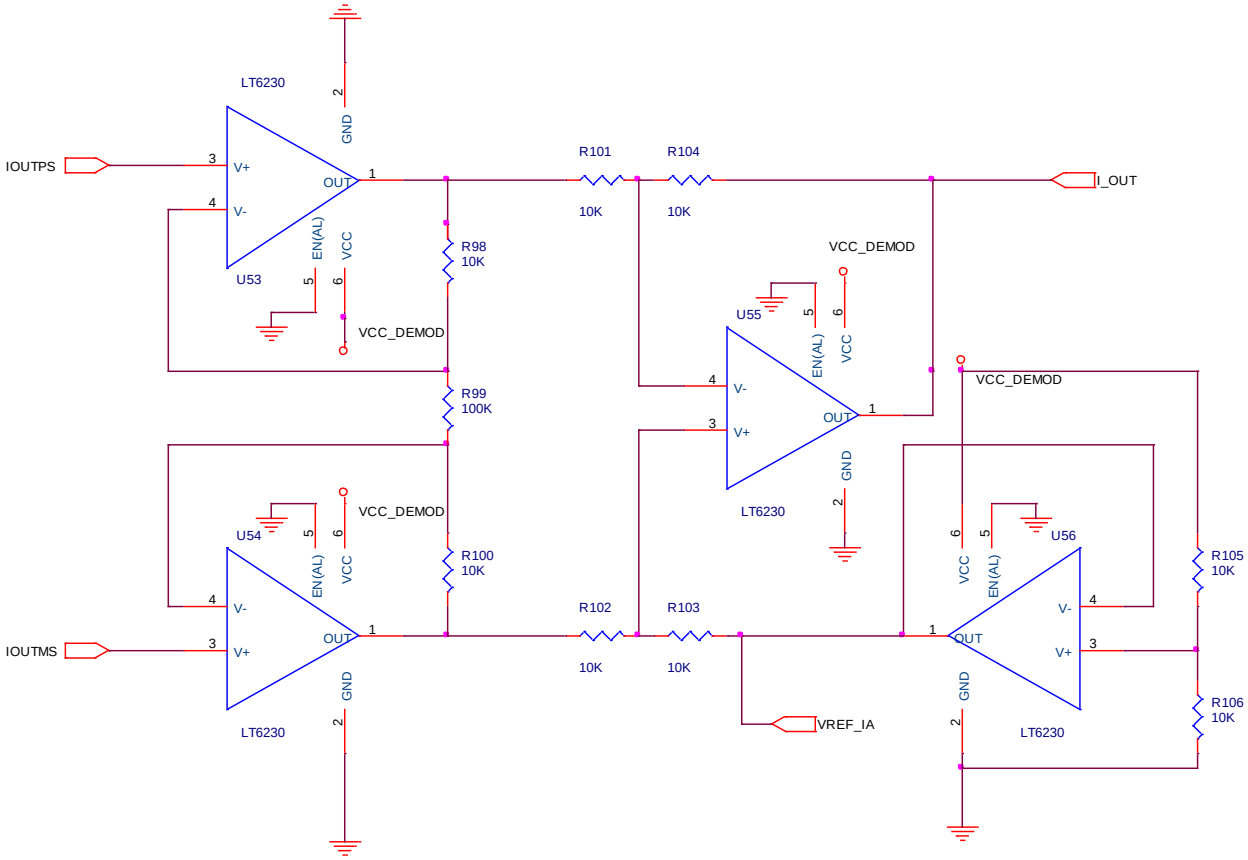

Figure B.4. – Instrumentation Amplifier for converting a differential signal to a single ended signal.

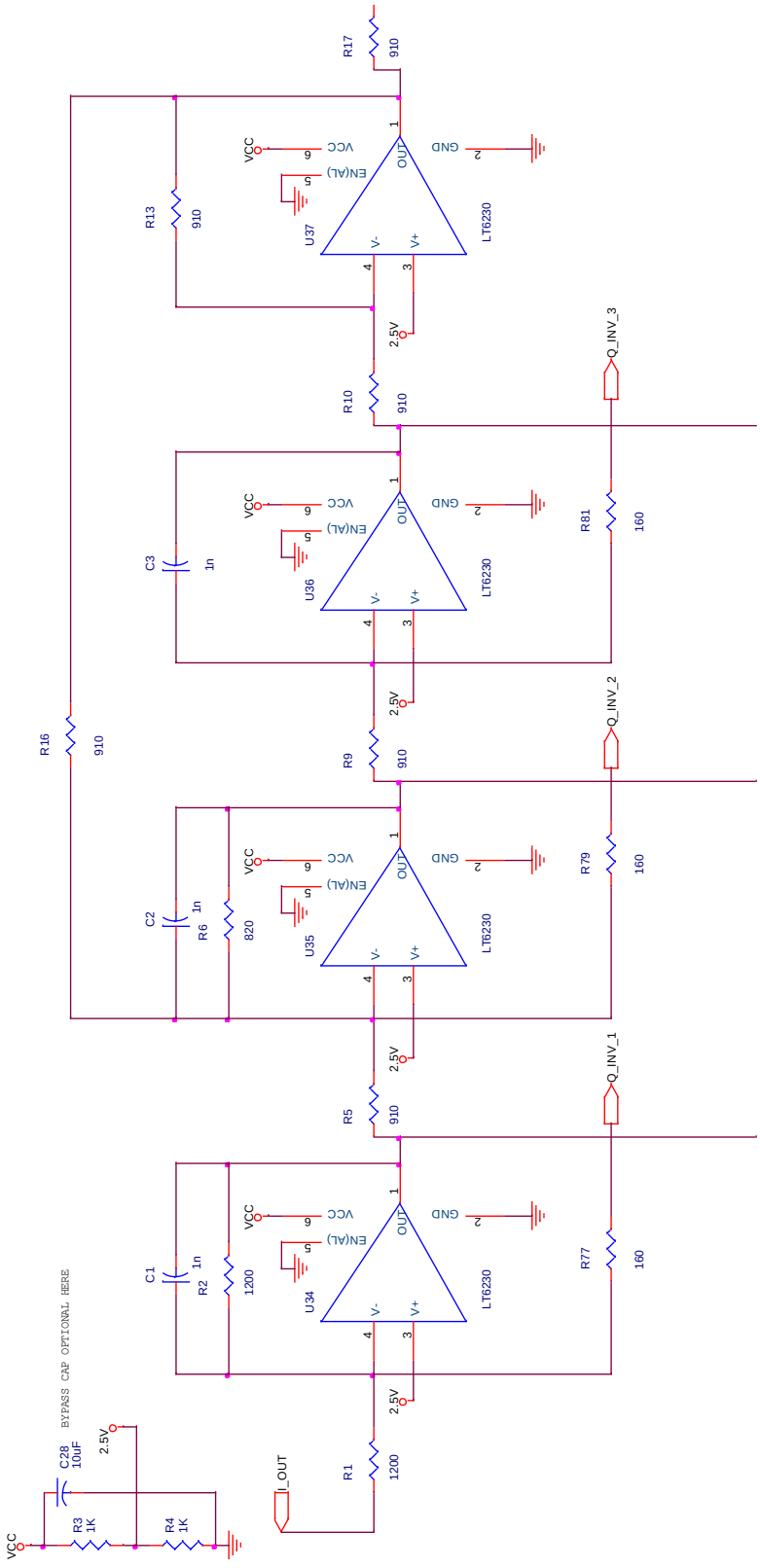

Figure B.5. – Stage 1 and 2 of the I channel for the complex bandpass filter. Only the inputs of the Q channel can be seen with regards to cross coupling. Stage 3 is identical to stage 2.

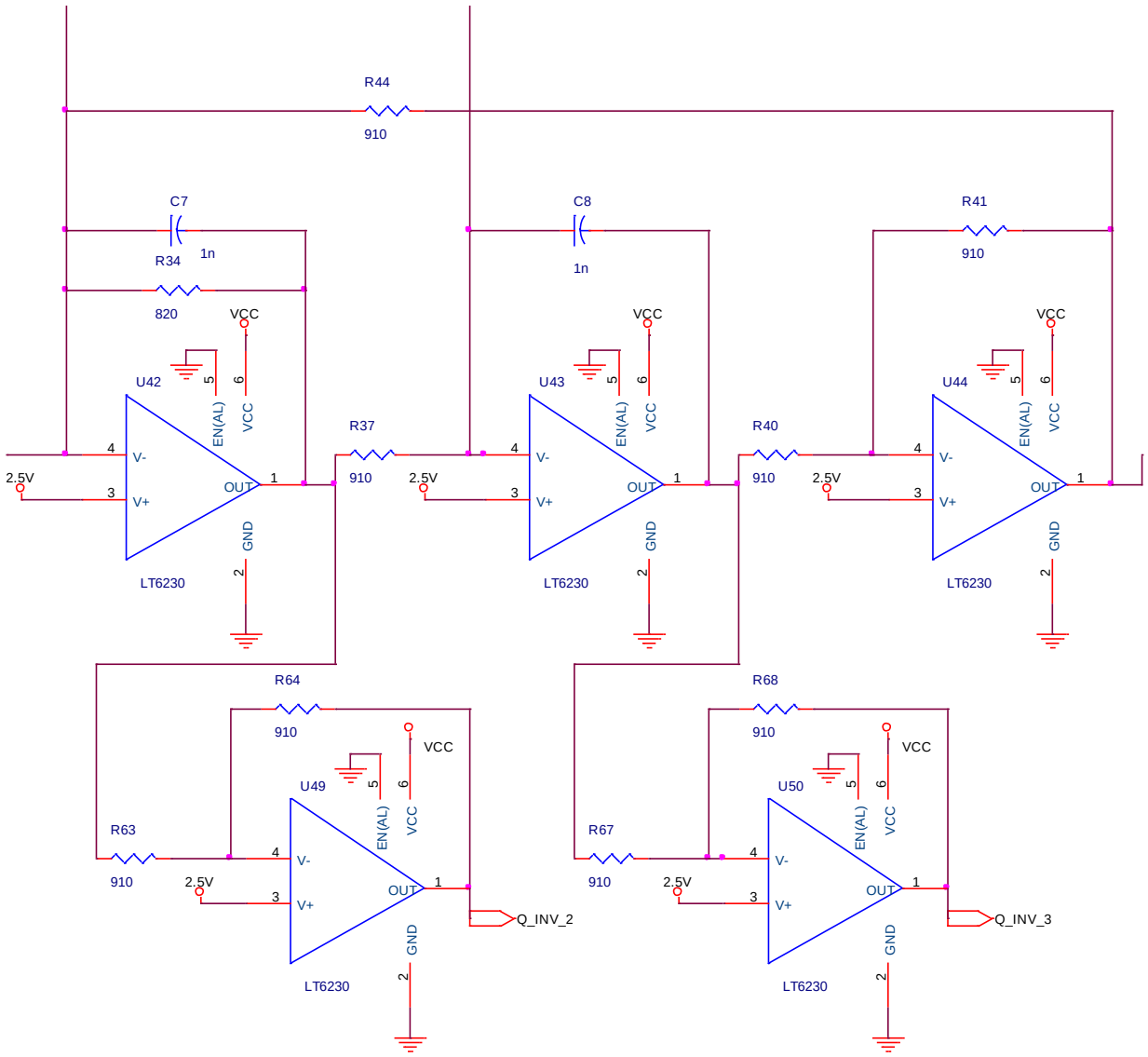

Figure B.6. – Stage 2 of the Q channel of the complex bandpass filter. The inverters for cross coupling the Q output to the I input integrators are shown. The I channel inputs are the sourceless connections at the top of the schematic.

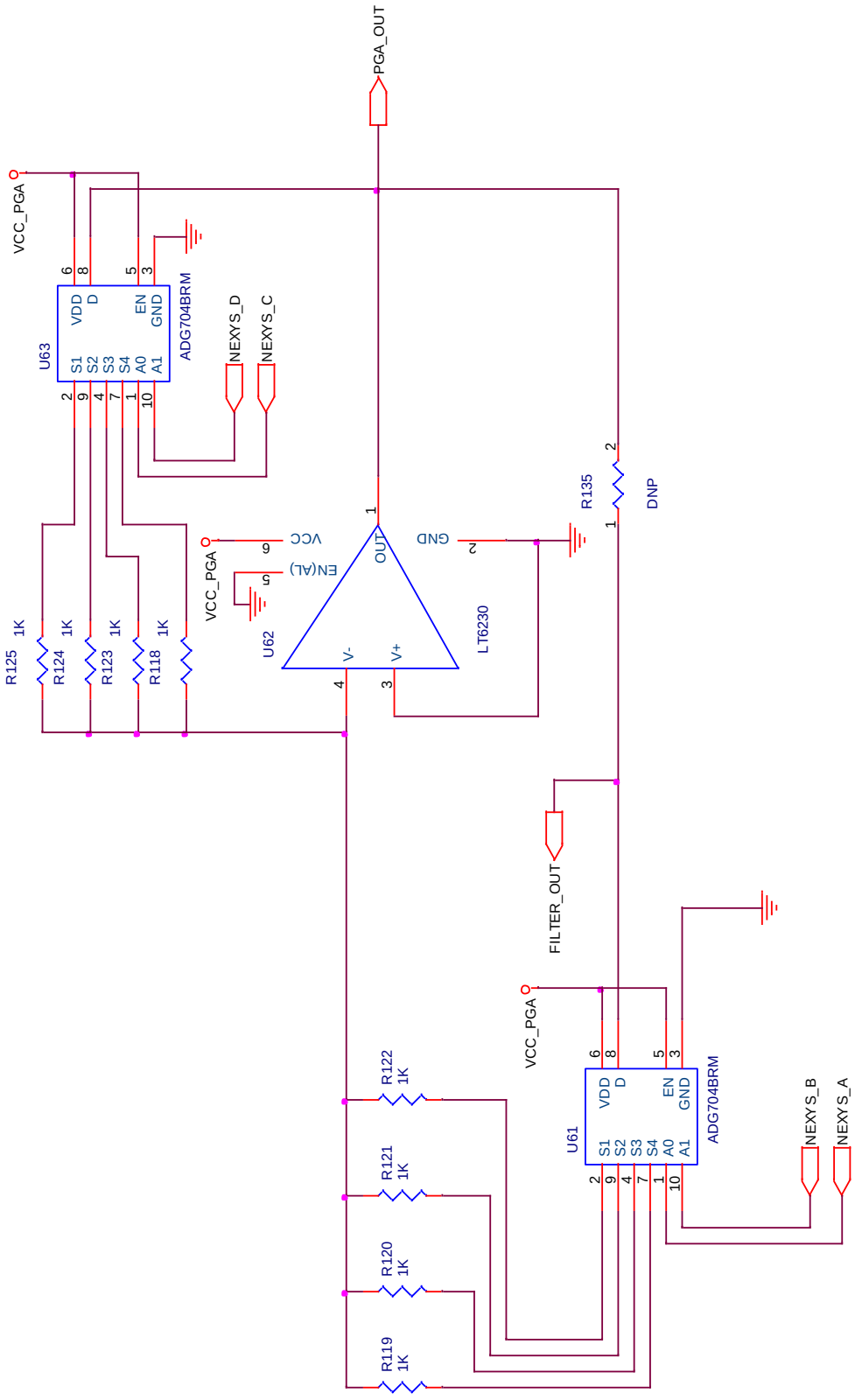

Figure B.7. – PGA schematic.

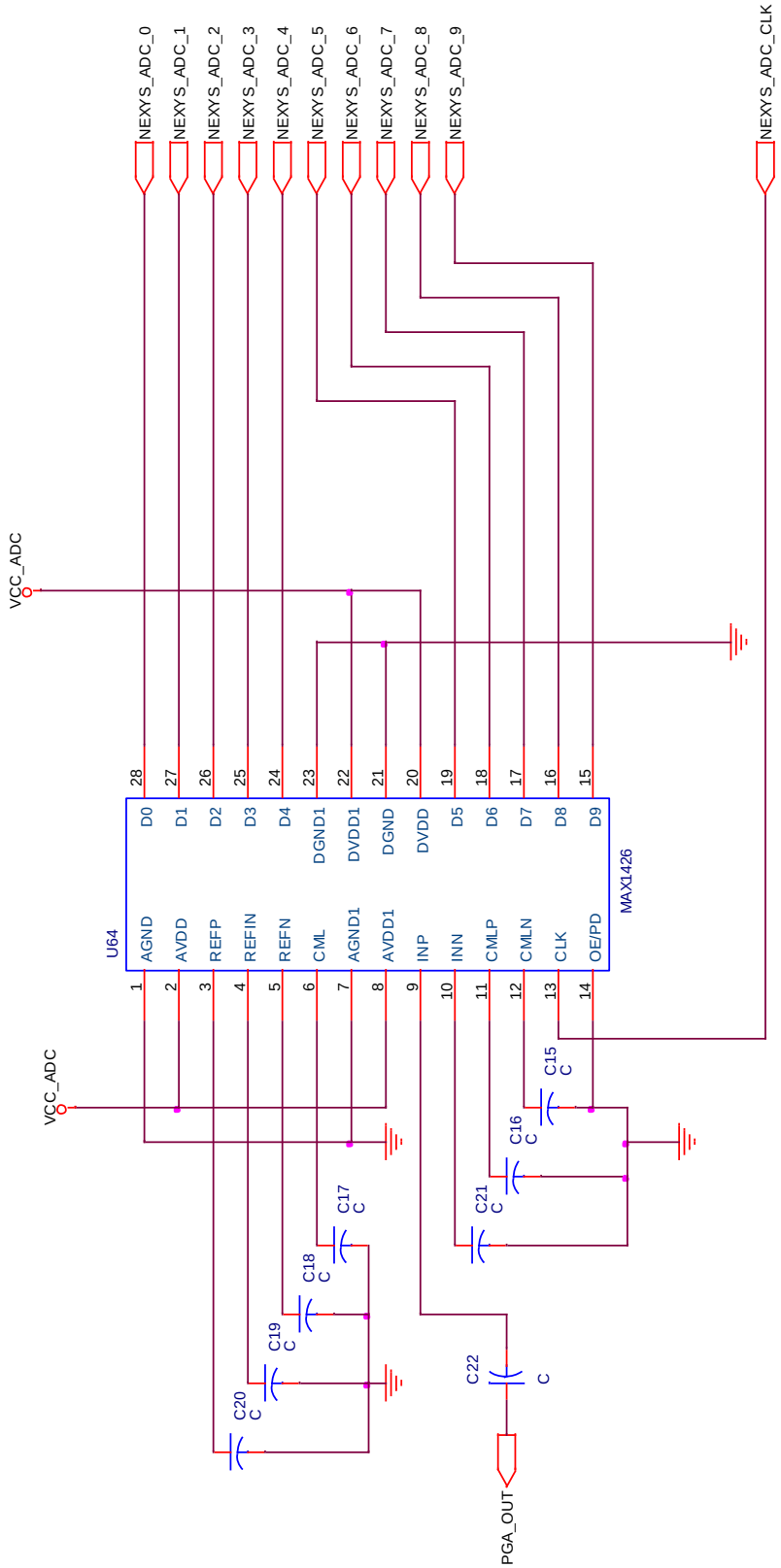

Figure B.8. – ADC interface schematic.

APPENDIX C: LAYOUT ARTWORK

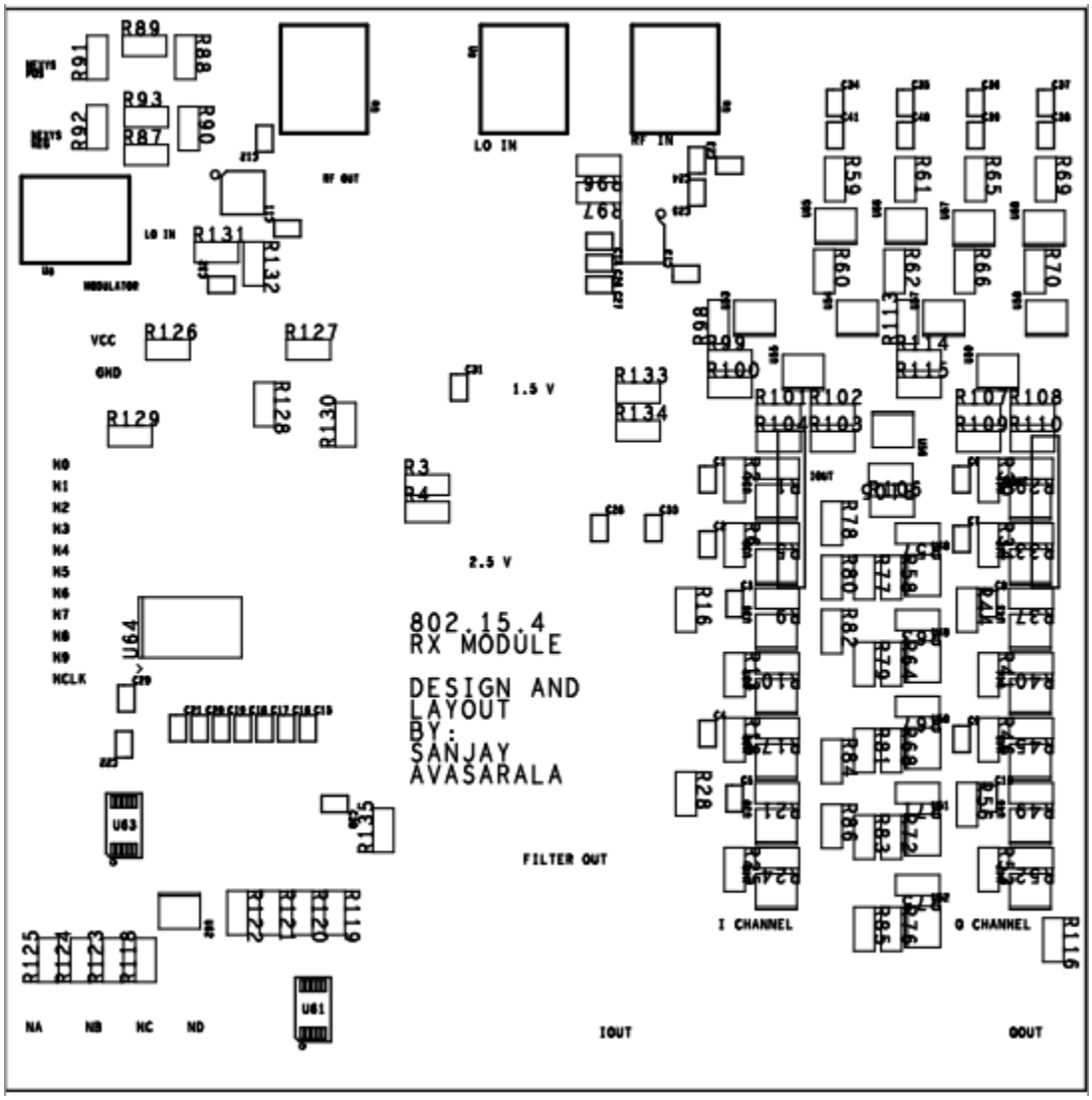

Figure C.1. – Top Silkscreen. The components can easily be seen above.

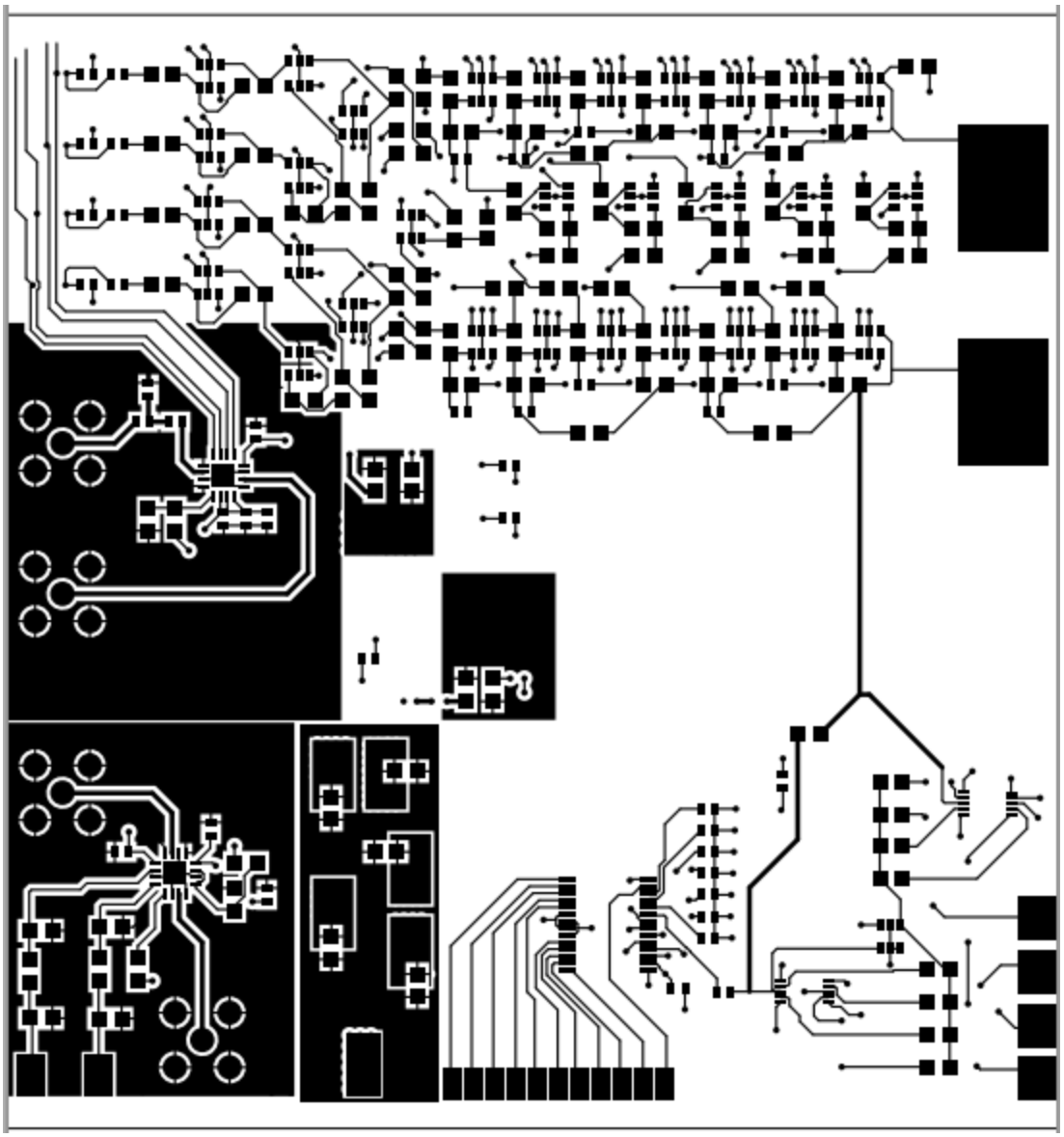

Figure C.2. – Top copper layer.

All subsequent layers are not shown since they consist of mainly ground planes and a small amount of traces.

APPENDIX D: PROGRAM LISTING (VHDL)

```
----------------------------------------------------------------------------------
-- Engineer: Sanjay Avasarala
--
-- Create Date:    18:05:32 04/16/2012
-- Design Name:      Final Pseudo-DSP Demodulator
-- Module Name:    Demodulation - Behavioral
-- Project Name:   Senior Project 802.15.4 Baseband Demodulator
-- Target Devices: Nexys 2 Board
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;


entity Demodulation is
    Port ( ADC_DATA     : in  STD_LOGIC_VECTOR (9 downto 0); -- Inputs data
from the ADC
           ADC_CLK  : out STD_LOGIC; -- Outputs a 5 MHz clock signal to the
ADC for sampling
              SIG_CLK_P : out STD_LOGIC; -- Outputs a 175 KHz data test
signalto the board.
              SIG_CLK_N : out STD_LOGIC; -- Outputs a 175 KHz data test
signalto the board.
              CLK          : in  STD_LOGIC; -- System CLK
              SYNC_CLK  : out  STD_LOGIC; -- External SYNC CLK
              TEST      : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST1         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST2         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST3         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST4         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST6         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST7         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST8         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
              TEST9         : out STD_LOGIC_VECTOR (14 downto 0); -- Prevents
removal of components
```

```
54                TEST5         : out STD_LOGIC; -- Prevents removal of
55  components
56                TEST0      : out STD_LOGIC; -- Prevents removal of components
57                DATA       : out STD_LOGIC; -- Data output of signal
58             PGA_CNTRL : out  STD_LOGIC_VECTOR (3 downto 0); -- PGA Controller
59  output
60                PGA_RSSI   : out STD_LOGIC_VECTOR (3 downto 0);
61                LOG_OUT_P : out STD_LOGIC;
62                LOG_OUT_N : out STD_LOGIC);
63          --   CLK_OUT    : out  STD_LOGIC);
64  end Demodulation;
65
66  architecture Behavioral of Demodulation is
67
68  ------------------SIGNAL DECLARATION--------------------------------------
69      signal temp_clk             : STD_LOGIC := '0'; -- Used for clock
70  division
71      signal temp_clk_2           : STD_LOGIC := '0';
72      signal temp_clk_3           : STD_LOGIC := '0'; -- used for pga flag
73
74      signal sampled_signal0  : STD_LOGIC_VECTOR (9 downto 0); -- These signal
75  lines are the sampled ADC values.
76      signal sampled_signal1  : STD_LOGIC_VECTOR (9 downto 0);
77      signal sampled_signal2  : STD_LOGIC_VECTOR (9 downto 0);
78      signal sampled_signal3  : STD_LOGIC_VECTOR (9 downto 0);
79      signal sampled_signal4  : STD_LOGIC_VECTOR (9 downto 0);
80
81      signal push_signal0     : STD_LOGIC_VECTOR (9 downto 0); -- These signals
82  are the receive the pushed signals
83      signal push_signal1     : STD_LOGIC_VECTOR (9 downto 0); -- from the
84  sampled values each time the counter trips.
85      signal push_signal2     : STD_LOGIC_VECTOR (9 downto 0);
86      signal push_signal3     : STD_LOGIC_VECTOR (9 downto 0);
87      signal push_signal4     : STD_LOGIC_VECTOR (9 downto 0);
88
89      signal add_signal0      : STD_LOGIC_VECTOR (14 downto 0); -- These
90  signals concatenate the pushed signals such
91      signal add_signal1      : STD_LOGIC_VECTOR (14 downto 0); -- all of them
92  can be added without overflow.
93      signal add_signal2      : STD_LOGIC_VECTOR (14 downto 0);
94      signal add_signal3      : STD_LOGIC_VECTOR (14 downto 0);
95      signal add_signal4      : STD_LOGIC_VECTOR (14 downto 0);
96
97      signal int_signal0      : STD_LOGIC_VECTOR (14 downto 0); -- These
98  signals receive the add_signals such that no
99      signal int_signal1      : STD_LOGIC_VECTOR (14 downto 0); -- driver
100 contention issues arise. These signals are the
101     signal int_signal2      : STD_LOGIC_VECTOR (14 downto 0); -- the signals
102 that are actually added.
103     signal int_signal3      : STD_LOGIC_VECTOR (14 downto 0);
104     signal int_signal4      : STD_LOGIC_VECTOR (14 downto 0);
105
106     signal pga_signal0      : STD_LOGIC_VECTOR (9 downto 0); -- These signals
107 are used for peak detection in the
108     signal pga_signal1      : STD_LOGIC_VECTOR (9 downto 0); -- PGA.
109     signal pga_signal2      : STD_LOGIC_VECTOR (9 downto 0);
110     signal pga_signal3      : STD_LOGIC_VECTOR (9 downto 0);
```

```
111       signal pga_signal4      : STD_LOGIC_VECTOR (9 downto 0);
112
113       signal comp_signal0     : STD_LOGIC_VECTOR (9 downto 0);
114       signal comp_signal1     : STD_LOGIC_VECTOR (9 downto 0);
115       signal comp_signal2     : STD_LOGIC_VECTOR (9 downto 0);
116       signal comp_signal3     : STD_LOGIC_VECTOR (9 downto 0);
117       signal comp_signal4     : STD_LOGIC_VECTOR (9 downto 0);
118
119       signal comp_signal5     : STD_LOGIC_VECTOR (9 downto 0);
120       signal comp_signal6     : STD_LOGIC_VECTOR (9 downto 0);
121       signal comp_signal7     : STD_LOGIC_VECTOR (9 downto 0);
122       signal comp_signal8     : STD_LOGIC_VECTOR (9 downto 0);
123       signal comp_signal9     : STD_LOGIC_VECTOR (9 downto 0);
124
125       signal comp_signal10        : STD_LOGIC_VECTOR (9 downto 0);
126       signal comp_signal11        : STD_LOGIC_VECTOR (9 downto 0);
127       signal comp_signal12        : STD_LOGIC_VECTOR (9 downto 0);
128       signal comp_signal13        : STD_LOGIC_VECTOR (9 downto 0);
129       signal comp_signal14        : STD_LOGIC_VECTOR (9 downto 0);
130
131       signal comp_signal15        : STD_LOGIC_VECTOR (9 downto 0);
132       signal comp_signal16        : STD_LOGIC_VECTOR (9 downto 0);
133       signal comp_signal17        : STD_LOGIC_VECTOR (9 downto 0);
134       signal comp_signal18        : STD_LOGIC_VECTOR (9 downto 0);
135       signal comp_signal19        : STD_LOGIC_VECTOR (9 downto 0);
136
137       signal comp_signal20        : STD_LOGIC_VECTOR (9 downto 0);
138       signal comp_signal21        : STD_LOGIC_VECTOR (9 downto 0);
139       signal comp_signal22        : STD_LOGIC_VECTOR (9 downto 0);
140       signal comp_signal23        : STD_LOGIC_VECTOR (9 downto 0);
141       signal comp_signal24        : STD_LOGIC_VECTOR (9 downto 0);
142
143       signal comp_signal25        : STD_LOGIC_VECTOR (9 downto 0);
144       signal comp_signal26        : STD_LOGIC_VECTOR (9 downto 0);
145       signal comp_signal27        : STD_LOGIC_VECTOR (9 downto 0);
146       signal comp_signal28        : STD_LOGIC_VECTOR (9 downto 0);
147       signal comp_signal29        : STD_LOGIC_VECTOR (9 downto 0);
148
149       signal comp_signal30        : STD_LOGIC_VECTOR (9 downto 0);
150       signal comp_signal31        : STD_LOGIC_VECTOR (9 downto 0);
151       signal comp_signal32        : STD_LOGIC_VECTOR (9 downto 0);
152       signal comp_signal33        : STD_LOGIC_VECTOR (9 downto 0);
153       signal comp_signal34        : STD_LOGIC_VECTOR (9 downto 0);
154
155       signal comp_signal35        : STD_LOGIC_VECTOR (9 downto 0);
156       signal comp_signal36        : STD_LOGIC_VECTOR (9 downto 0);
157       signal comp_signal37        : STD_LOGIC_VECTOR (9 downto 0);
158       signal comp_signal38        : STD_LOGIC_VECTOR (9 downto 0);
159       signal comp_signal39        : STD_LOGIC_VECTOR (9 downto 0);
160
161       signal comp_signal40        : STD_LOGIC_VECTOR (9 downto 0);
162       signal comp_signal41        : STD_LOGIC_VECTOR (9 downto 0);
163       signal comp_signal42        : STD_LOGIC_VECTOR (9 downto 0);
164       signal comp_signal43        : STD_LOGIC_VECTOR (9 downto 0);
165       signal comp_signal44        : STD_LOGIC_VECTOR (9 downto 0);
166
167
```

```
168     signal flag                 : STD_LOGIC := '0'; -- The flag variable
169 changes state per bit period. This state change
170                                                     -- will trip
171 the process that determines the sign
172     signal pga_flag             : STD_LOGIC := '0';
173
174
175     -- max_count ---> for generating the 5 MHz clock
176     -- max_count_s ----> for generating the 150 Khz signal
177     -- max_count_l ----> used for the sequential process
178
179     -- NOTE:
180     -- To set the bit rate for a square wave, the max_count_s variable has to
181 be changed.
182     -- The formula is 50M/(2*Bitrate)
183     -- For 150 Kbaud ----> 165
184     -- For 250 Kbaud ----> 100
185     --
186     -- To set the bit rate for a signal, the max_count_s variable has to be
187 changed.
188     -- The formula is 50M/(Bitrate)
189     -- For 150 Kbaud ----> 330
190     -- For 250 Kbaud ----> 200
191     constant max_count      : integer := (4); -- sets sampling clk to 5 MHz.
192 Note that the real number
193     constant max_count_l    : integer := (5); -- should be 4 for a 5MHz clock
194 but the value of 5 is
195     constant max_count_s        : integer := (100); -- used for sequential
196 process operation reasons.
197     constant max_count_pga  : integer := (9); --arbitrary wait value for pga
198 update process
199     constant max_count_pga_process : integer := (5000); -- sets PGA update
200 frequency
201     constant index_count    : integer := (15); -- used for data sequence
202 indexing
203     type my_arr is array (integer range 0 to 15) of std_logic; -- declaring
204 the data array
205     signal data_prbs: my_arr :=
206 ('1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0'); --
207 intialize the array
208     signal PRBS_SEQ         : STD_LOGIC_VECTOR(15 downto 0);
209
210
211     signal out_s0           : STD_LOGIC_VECTOR (9 downto 0);
212     signal out_s1           : STD_LOGIC_VECTOR (9 downto 0);
213     signal out_s2           : STD_LOGIC_VECTOR (9 downto 0);
214     signal out_s3           : STD_LOGIC_VECTOR (9 downto 0);
215     signal out_s4           : STD_LOGIC_VECTOR (9 downto 0);
216     signal out_s5           : STD_LOGIC_VECTOR (9 downto 0);
217     signal out_s6           : STD_LOGIC_VECTOR (9 downto 0);
218     signal out_s7           : STD_LOGIC_VECTOR (9 downto 0);
219     signal out_s8           : STD_LOGIC_VECTOR (9 downto 0);
220     signal out_s9           : STD_LOGIC_VECTOR (9 downto 0);
221     signal out_s10          : STD_LOGIC_VECTOR (9 downto 0);
222     signal out_s11          : STD_LOGIC_VECTOR (9 downto 0);
223     signal out_s12          : STD_LOGIC_VECTOR (9 downto 0);
224     signal out_s13          : STD_LOGIC_VECTOR (9 downto 0);
```

```
225       signal out_s14          : STD_LOGIC_VECTOR (9 downto 0);
226       signal out_s15          : STD_LOGIC_VECTOR (9 downto 0);
227       signal out_s16          : STD_LOGIC_VECTOR (9 downto 0);
228       signal out_s17          : STD_LOGIC_VECTOR (9 downto 0);
229       signal out_s18          : STD_LOGIC_VECTOR (9 downto 0);
230       signal out_s19          : STD_LOGIC_VECTOR (9 downto 0);
231       signal out_s20          : STD_LOGIC_VECTOR (9 downto 0);
232       signal out_s21          : STD_LOGIC_VECTOR (9 downto 0);
233       signal out_s22          : STD_LOGIC_VECTOR (9 downto 0);
234       signal out_s23          : STD_LOGIC_VECTOR (9 downto 0);
235       signal out_s24          : STD_LOGIC_VECTOR (9 downto 0);
236       signal out_s25          : STD_LOGIC_VECTOR (9 downto 0);
237       signal out_s26          : STD_LOGIC_VECTOR (9 downto 0);
238       signal out_s27          : STD_LOGIC_VECTOR (9 downto 0);
239       signal out_s28          : STD_LOGIC_VECTOR (9 downto 0);
240       signal out_s29          : STD_LOGIC_VECTOR (9 downto 0);
241       signal out_s30          : STD_LOGIC_VECTOR (9 downto 0);
242       signal out_s31          : STD_LOGIC_VECTOR (9 downto 0);
243       signal out_s32          : STD_LOGIC_VECTOR (9 downto 0);
244       signal out_s33          : STD_LOGIC_VECTOR (9 downto 0);
245       signal out_s34          : STD_LOGIC_VECTOR (9 downto 0);
246       signal out_s35          : STD_LOGIC_VECTOR (9 downto 0);
247       signal out_s36          : STD_LOGIC_VECTOR (9 downto 0);
248       signal out_s37          : STD_LOGIC_VECTOR (9 downto 0);
249       signal out_s38          : STD_LOGIC_VECTOR (9 downto 0);
250       signal out_s39          : STD_LOGIC_VECTOR (9 downto 0);
251       signal out_s40          : STD_LOGIC_VECTOR (9 downto 0);
252       signal out_s41          : STD_LOGIC_VECTOR (9 downto 0);
253       signal out_s42          : STD_LOGIC_VECTOR (9 downto 0);
254       signal out_s43          : STD_LOGIC_VECTOR (9 downto 0);
255
256   --------------COMPONENT DECLARATION--------------
257       component Full_Adder is             -- Full adder declaration
258       Port ( X : in  STD_LOGIC;
259              Y : in  STD_LOGIC;
260              Z : in  STD_LOGIC;
261              SUM : out  STD_LOGIC;
262              CARRY : out  STD_LOGIC);
263       end component;
264
265       signal sum_inv              : STD_LOGIC_VECTOR (9 downto 0);
266       signal carry_inv            : STD_LOGIC_VECTOR (9 downto 0);
267
268   -----------------
269       component Adder_20_Bit is -- 15 Bit adder declaration
270       Port ( INPUT_A : in  STD_LOGIC_VECTOR (14 downto 0);
271              INPUT_B : in  STD_LOGIC_VECTOR (14 downto 0);
272              OUTPUT : out  STD_LOGIC_VECTOR (14 downto 0));
273       end component;
274
275       signal OUT_SIG0 : STD_LOGIC_VECTOR (14 downto 0); -- These signals are
276   involved in the addition process.
277       signal OUT_SIG1 : STD_LOGIC_VECTOR (14 downto 0); -- OUT_SIG3 is the
278   signal used for sign determining
279       signal OUT_SIG2 : STD_LOGIC_VECTOR (14 downto 0);
280       signal OUT_SIG3 : STD_LOGIC_VECTOR (14 downto 0);
281
```

```
282  -----------------
283
284      component Clock_Division_1
285      Port(
286      CLKIN_IN : IN std_logic;
287      CLKDV_OUT : OUT std_logic;
288      CLK0_OUT : OUT std_logic
289      );
290      end component;
291
292  ------------------------------------------------------------
293
294  begin
295
296  ------------- EXTERNAL SYNC CLOCK -------------- Sets the sync for the LO's
297      Inst_Clock_Division_1: Clock_Division_1 PORT MAP(
298          CLKIN_IN => CLK,
299          CLKDV_OUT => SYNC_CLK,
300          CLK0_OUT => TEST0
301      );
302  ------------------------------------------------
303
304  ------------- CLOCK DIVISION ------------------- Sets a 5 MHz clock
305      division : process(clk,temp_clk)
306      variable count  : integer := (0);
307      begin
308          if(falling_edge(clk)) then
309              if(count = max_count) then
310                  temp_clk <= NOT temp_clk;
311                  count := 0;
312              else
313                  temp_clk <= temp_clk;
314                  count := count + 1;
315              end if;
316          end if;
317          ADC_CLK <= temp_clk;
318      end process division;
319  ------------------------------------------------
320
321      PRBS_SEQ <= "0110110001011100";
322  ------------- SIGNAL GENERATION ------------------- Sets a 250 KHz signal
323      division2 : process(clk,temp_clk_2)
324      variable count2 : integer := (0);
325      variable index    : integer range 0 to 31 := (0);
326      begin
327          if(falling_edge(clk)) then
328              if(index = index_count) then
329                  index := 0;
330              end if;
331              if(count2 = max_count_s) then
332                  case index is
333                      when 0 =>
334                          temp_clk_2 <= PRBS_SEQ(0);
335                      when 1 =>
336                          temp_clk_2 <= PRBS_SEQ(1);
337                      when 2 =>
338                          temp_clk_2 <= PRBS_SEQ(2);
```

```
339                     when 3 =>
340                         temp_clk_2 <= PRBS_SEQ(3);
341                     when 4 =>
342                         temp_clk_2 <= PRBS_SEQ(4);
343                     when 5 =>
344                         temp_clk_2 <= PRBS_SEQ(5);
345                     when 6 =>
346                         temp_clk_2 <= PRBS_SEQ(6);
347                     when 7 =>
348                         temp_clk_2 <= PRBS_SEQ(7);
349                     when 8 =>
350                         temp_clk_2 <= PRBS_SEQ(8);
351                     when 9 =>
352                         temp_clk_2 <= PRBS_SEQ(9);
353                     when 10 =>
354                         temp_clk_2 <= PRBS_SEQ(10);
355                     when 11 =>
356                         temp_clk_2 <= PRBS_SEQ(11);
357                     when 12 =>
358                         temp_clk_2 <= PRBS_SEQ(12);
359                     when 13 =>
360                         temp_clk_2 <= PRBS_SEQ(13);
361                     when 14 =>
362                         temp_clk_2 <= PRBS_SEQ(14);
363                     when 15 =>
364                         temp_clk_2 <= PRBS_SEQ(15);
365                     when others =>
366                         null;
367                     end case;
368                 --temp_clk_2 <= data_prbs(index);
369                 count2 := 0;
370                 index := index+1;
371             else
372                 temp_clk_2 <= temp_clk_2;
373                 count2 := count2 + 1;
374             end if;
375         --index := index+1;
376         end if;
377         SIG_CLK_P <= temp_clk_2;
378         LOG_OUT_P <= temp_clk_2;
379         SIG_CLK_N <= NOT temp_clk_2;
380         LOG_OUT_N <= NOT temp_clk_2;
381     end process division2;
382 -------------------------------------------------
383
384 --------Flag Clock Generation for PGA------------
385     division_pga_flag : process(clk,temp_clk_3)
386     variable count_pga  : integer := (0);
387     begin
388         if(falling_edge(clk)) then
389             if(count_pga = max_count_pga) then
390                 temp_clk_3 <= NOT temp_clk_3;
391                 count_pga := 0;
392             else
393                 temp_clk_3 <= temp_clk_3;
394                 count_pga := count_pga + 1;
395             end if;
```

```
396         end if;
397     end process division_pga_flag;
398 
399 ------------ ADC SAMPLING ---------------------
400     sampling :process(temp_clk)
401     variable adc_count : integer := 0;
402     begin
403         if(falling_edge(temp_clk)) then -- check this clock
404             if(adc_count = max_count_1) then -- this if statement should be
405 carried out first before the case statement trips
406                 adc_count := 0;
407                 flag <= NOT flag; -- flag changes state upon bit sampling
408 completion. this should in theory trip the process sdet
409                 push_signal0 <= sampled_signal0;
410                 push_signal1 <= sampled_signal1;
411                 push_signal2 <= sampled_signal2;
412                 push_signal3 <= sampled_signal3;
413                 push_signal4 <= sampled_signal4;
414             end if;
415             case adc_count is
416                 when 0 =>
417                     sampled_signal0 <= ADC_DATA;
418                     pga_signal0 <= ADC_DATA;
419                 when 1 =>
420                     sampled_signal1 <= ADC_DATA;
421                     sampled_signal1 <= "0000000000";
422                     pga_signal1 <= ADC_DATA;
423                 when 2 =>
424                     sampled_signal2 <= ADC_DATA;
425                     sampled_signal2 <= NOT sampled_signal2; -- Ones
426 compliment. The extra bit is added in the pushed signal.
427                     pga_signal2 <= ADC_DATA;
428                 when 3 =>
429                     sampled_signal3 <= ADC_DATA;
430                     sampled_signal3 <= "0000000000";
431                     pga_signal3 <= ADC_DATA;
432                 when 4 =>
433                     sampled_signal4 <= ADC_DATA;
434                     pga_signal4 <= ADC_DATA;
435                 when others =>
436                     null;
437             end case;
438             adc_count := adc_count + 1;
439         end if;
440     end process sampling;
441 ------------------------------------------------
442 
443 -------------- ADDING/MULT ----------------------
444 -- This block adds a single bit the the ones complement to convert the
445 -- original number to its twos complement. This technique is used to invert
446 -- both positive and negative sequences.
447 
448     U1: Full_Adder
449          port map (X => push_signal2(0), Y => '1', Z => '0', SUM =>
450 sum_inv(0), CARRY => carry_inv(0));
451     U2: Full_Adder
```

```
452             port map (X => push_signal2(1), Y => '0', Z => carry_inv(0), SUM =>
453     sum_inv(1), CARRY => carry_inv(1));
454         U3: Full_Adder
455             port map (X => push_signal2(2), Y => '0', Z => carry_inv(1), SUM =>
456     sum_inv(2), CARRY => carry_inv(2));
457         U4: Full_Adder
458             port map (X => push_signal2(3), Y => '0', Z => carry_inv(2), SUM =>
459     sum_inv(3), CARRY => carry_inv(3));
460         U5: Full_Adder
461             port map (X => push_signal2(4), Y => '0', Z => carry_inv(3), SUM =>
462     sum_inv(4), CARRY => carry_inv(4));
463         U6: Full_Adder
464             port map (X => push_signal2(5), Y => '0', Z => carry_inv(4), SUM =>
465     sum_inv(5), CARRY => carry_inv(5));
466         U7: Full_Adder
467             port map (X => push_signal2(6), Y => '0', Z => carry_inv(5), SUM =>
468     sum_inv(6), CARRY => carry_inv(6));
469         U8: Full_Adder
470             port map (X => push_signal2(7), Y => '0', Z => carry_inv(6), SUM =>
471     sum_inv(7), CARRY => carry_inv(7));
472         U9: Full_Adder
473             port map (X => push_signal2(8), Y => '0', Z => carry_inv(7), SUM =>
474     sum_inv(8), CARRY => carry_inv(8));
475         U10: Full_Adder
476             port map (X => push_signal2(9), Y => '0', Z => carry_inv(8), SUM =>
477     sum_inv(9), CARRY => carry_inv(9));
478     ------------------------------------------------
479
480     -- THE SAMPLES SIGNALS ARE NOW
481     -- push_signal0
482     -- push_signal1
483     -- sum_inv
484     -- push_signal3
485     -- push_signal4
486
487
488         TEST <= add_signal0;
489         TEST1 <= add_signal1;
490         TEST2 <= add_signal2;
491         TEST3 <= add_signal3;
492         TEST4 <= add_signal4;
493         TEST5 <= carry_inv(9);
494
495
496
497         -- All the signals are in two's complement now. Therefore, to
498     concactenate the signals, a check
499         -- must be performed to add '1's instead of '0's for negative signals.
500
501         concat_check:
502     process(clk,push_signal0,push_signal1,sum_inv,push_signal3,push_signal4)
503         begin
504             if(push_signal0(9) = '1') then
505                 add_signal0 <= "11111" & push_signal0;
506             else
507                 add_signal0 <= "00000" & push_signal0;
508             end if;
```

```
509             if(push_signal1(9) = '1') then
510                 add_signal1 <= "11111" & push_signal1;
511             else
512                 add_signal1 <= "00000" & push_signal1;
513             end if;
514             if(sum_inv(9) = '1') then
515                 add_signal2 <= "11111" & sum_inv;
516             else
517                 add_signal2 <= "00000" & sum_inv;
518             end if;
519             if(push_signal3(9) = '1') then
520                 add_signal3 <= "11111" & push_signal3;
521             else
522                 add_signal3 <= "00000" & push_signal3;
523             end if;
524             if(push_signal4(9) = '1') then
525                 add_signal4 <= "11111" & push_signal4;
526             else
527                 add_signal4 <= "00000" & push_signal4;
528             end if;
529         end process concat_check;
530 
531         -- The following signals prevent multiple drivers
532         int_signal0 <= add_signal0;
533         int_signal1 <= add_signal1;
534         int_signal2 <= add_signal2;
535         int_signal3 <= add_signal3;
536         int_signal4 <= add_signal4;
537 
538 
539 -------------- ADDER -------------------------------- -- this is essentially
540 the integration step.
541             K1: Adder_20_Bit
542                 port map (int_signal0, int_signal1, OUT_SIG0);
543             K2: Adder_20_Bit
544                 port map (OUT_SIG0, int_signal2, OUT_SIG1);
545             K3: Adder_20_Bit
546                 port map (OUT_SIG1, int_signal3, OUT_SIG2);
547             K4: Adder_20_Bit
548                 port map (OUT_SIG2, int_signal4, OUT_SIG3);
549 ------------------------------------------------------
550 
551 -- OUT_SIG3 Is the final output signal into the sign determiner in this case
552 
553         TEST6 <= OUT_SIG0;
554         TEST7 <= OUT_SIG1;
555         TEST8 <= OUT_SIG2;
556         TEST9 <= OUT_SIG3;
557 
558 
559 -------------- DETERMINE THE SIGN----------------------
560         sdet: process(flag, OUT_SIG3)   -- trips on any flag event
561         begin
562 
563             if(OUT_SIG3(14) = '1') then
564                 DATA <= '0';
565             else
```

```
566             DATA <= '1';
567         end if;
568
569     end process sdet;
570 -------------------------------------------------------
571
572
573 ------------------- PGA CONTROL--------------------------
574     pga_proc: process(flag,pga_signal0,pga_signal1,pga_signal2,
575     pga_signal3,pga_signal4,comp_signal0,comp_signal1,comp_signal2,
576     comp_signal3,comp_signal4,comp_signal5,comp_signal6,comp_signal7,
577     comp_signal8,comp_signal9,comp_signal10,comp_signal11,comp_signal12,
578     comp_signal13,comp_signal14,comp_signal15,comp_signal16,comp_signal17,
579     comp_signal18,comp_signal19,comp_signal20,comp_signal21,comp_signal22,
580     comp_signal23,comp_signal24,comp_signal25,comp_signal26,comp_signal27,
581     comp_signal28,comp_signal29,comp_signal30,comp_signal31,comp_signal32,
582     comp_signal33,comp_signal34,out_s33)    -- trips on any flag event
583
584     variable index_pga : integer := (0);
585     variable index_pga_process :integer := (0);
586     variable flag_compare_ready : integer := (0);
587     variable pga_flag    : integer := (0);
588
589     begin
590
591
592
593         if(falling_edge(flag)) then -- every falling edge, the index is
594 incremented and a new set of signals gets the pga_signals generated in
595             if(index_pga_process = max_count_pga_process) then
596                 PGA_CNTRL <= "0000"; -- RESET to the normal state and 'poll'
597 signal.
598                 PGA_RSSI <= "0000";
599                 pga_flag := 1;
600                 index_pga_process := 0;
601             else
602                 index_pga_process := index_pga_process + 1;
603             end if;
604
605
606     -- this polling is done because if the system is not reset then positive
607 feedback will occur
608     -- and PGA will begin to oscillate. by reseting, the system will take in
609 the new data and adjust pga to
610     -- that signal input. the drawback of this method is that there will be
611 periodic instances where
612     -- the signal will degrade.
613
614
615     -- SOFTWARE FLOW:
616     -- an event on flag trips this process
617     -- master if loop on top checks for falling edge
618     -- if no, then moves on to second if state check
619     -- since the compare flag is 0, nothing will happen.
620
621     -- if falling edge of the flag then the second if loop checks if the
622 index_pga_process counter has reached a max limit (arbitrary)
```

```
623         -- if yes then the PGA is reset such that the signal is pass through un
624     altered. this allows the ADC to recal.
625         -- the pga flag is set to 1 so that a new comparison can happen
626         -- the index_pga_process is set to 0 again for a new count. so
627     essentially, while the compare process is happening, the aforementioned
628         -- if loop keeps checking for a reset trip. this overlap is negligible if
629     the max_count_pga_process is sufficiently large
630 
631         -- the next if loop keeps checking if the pga_flag is raised. once
632     raised, signals are assigned the sampled values. this repeats till the index
633         -- pga counter has reached the number 7. once that happens, no more
634     signals are assigned, the index_pga is set to 0 and the flag is reset to 0.
635         -- the comparator sequence is initiated and the max values is assigned to
636     out_s33.
637 
638         -- the last execution in the aforementioned if statement is the setting
639     of the compare_complete flag to 1. once this is set, the state check
640         -- if loop can run and the final PGA value is output to the pga. since
641     the compare sequence is now done, the master pga_process counter continues
642         -- to run until the next trip value where the PGA is reset and the whole
643     compare process is repeated.
644 
645 
646                 if(pga_flag = 1) then                       -- the ADC sampling
647     process
648                     case index_pga is
649                     when 0 =>
650                         comp_signal0 <= pga_signal0;
651                         comp_signal1 <= pga_signal1;
652                         comp_signal2 <= pga_signal2;
653                         comp_signal3 <= pga_signal3;
654                         comp_signal4 <= pga_signal4;
655                     when 1 =>
656                         comp_signal5 <= pga_signal0;
657                         comp_signal6 <= pga_signal1;
658                         comp_signal7 <= pga_signal2;
659                         comp_signal8 <= pga_signal3;
660                         comp_signal9 <= pga_signal4;
661                     when 2 =>
662                         comp_signal10 <= pga_signal0;
663                         comp_signal11 <= pga_signal1;
664                         comp_signal12 <= pga_signal2;
665                         comp_signal13 <= pga_signal3;
666                         comp_signal14 <= pga_signal4;
667                     when 3 =>
668                         comp_signal15 <= pga_signal0;
669                         comp_signal16 <= pga_signal1;
670                         comp_signal17 <= pga_signal2;
671                         comp_signal18 <= pga_signal3;
672                         comp_signal19 <= pga_signal4;
673                     when 4 =>
674                         comp_signal20 <= pga_signal0;
675                         comp_signal21 <= pga_signal1;
676                         comp_signal22 <= pga_signal2;
677                         comp_signal23 <= pga_signal3;
678                         comp_signal24 <= pga_signal4;
679                     when 5 =>
```

```
680                     comp_signal25 <= pga_signal0;
681                     comp_signal26 <= pga_signal1;
682                     comp_signal27 <= pga_signal2;
683                     comp_signal28 <= pga_signal3;
684                     comp_signal29 <= pga_signal4;
685                 when 6 =>
686                     comp_signal30 <= pga_signal0;
687                     comp_signal31 <= pga_signal1;
688                     comp_signal32 <= pga_signal2;
689                     comp_signal33 <= pga_signal3;
690                     comp_signal34 <= pga_signal4;
691                 when 7 =>
692                     comp_signal35 <= pga_signal0;
693                     comp_signal36 <= pga_signal1;
694                     comp_signal37 <= pga_signal2;
695                     comp_signal38 <= pga_signal3;
696                     comp_signal39 <= pga_signal4;
697                 when 8 =>
698                     comp_signal40 <= pga_signal0;
699                     comp_signal41 <= pga_signal1;
700                     comp_signal42 <= pga_signal2;
701                     comp_signal43 <= pga_signal3;
702                     comp_signal44 <= pga_signal4;
703                 when others =>
704                     null;
705             end case;
706             if(index_pga = max_count_pga) then -- If the index counter hits
707 the sample number then the comparator is performed
708                 index_pga := 0;
709                 pga_flag := 0;
710 
711                 -------- COMPARATOR SEQUENCE--------------------
712                 if(comp_signal0 > comp_signal1) then
713                     out_s0 <= comp_signal0;
714                 else
715                     out_s0 <= comp_signal1;
716                 end if;
717                 if(out_s0 > comp_signal2) then
718                     out_s1 <= out_s0;
719                 else
720                     out_s1 <= comp_signal2;
721                 end if;
722                 if(out_s1 > comp_signal3) then
723                     out_s2 <= out_s1;
724                 else
725                     out_s2 <= comp_signal3;
726                 end if;
727                 if(out_s2 > comp_signal4) then
728                     out_s3 <= out_s2;
729                 else
730                     out_s3 <= comp_signal4;
731                 end if;
732                 if(out_s3 < comp_signal5) then
733                     out_s4 <= out_s3;
734                 else
735                     out_s4 <= comp_signal5;
736                 end if;
```

```
737                 if(out_s4 > comp_signal6) then
738                     out_s5 <= out_s4;
739                 else
740                     out_s5 <= comp_signal6;
741                 end if;
742                 if(out_s5 > comp_signal7) then
743                     out_s6 <= out_s5;
744                 else
745                     out_s6 <= comp_signal7;
746                 end if;
747                 if(out_s6 > comp_signal8) then
748                     out_s7 <= out_s6;
749                 else
750                     out_s7 <= comp_signal8;
751                 end if;
752                 if(out_s7 > comp_signal9) then
753                     out_s8 <= out_s7;
754                 else
755                     out_s8 <= comp_signal9;
756                 end if;
757                 if(out_s8 > comp_signal10) then
758                     out_s9 <= out_s8;
759                 else
760                     out_s9 <= comp_signal10;
761                 end if;
762                 if(out_s9 > comp_signal11) then
763                     out_s10 <= out_s9;
764                 else
765                     out_s10 <= comp_signal11;
766                 end if;
767                 if(out_s10 > comp_signal12) then
768                     out_s11 <= out_s10;
769                 else
770                     out_s11 <= comp_signal12;
771                 end if;
772                 if(out_s11 > comp_signal13) then
773                     out_s12 <= out_s11;
774                 else
775                     out_s12 <= comp_signal13;
776                 end if;
777                 if(out_s12 > comp_signal4) then
778                     out_s13 <= out_s12;
779                 else
780                     out_s13 <= comp_signal14;
781                 end if;
782                 if(out_s13 > comp_signal5) then
783                     out_s14 <= out_s13;
784                 else
785                     out_s14 <= comp_signal15;
786                 end if;
787                 if(out_s14 > comp_signal16) then
788                     out_s15 <= out_s14;
789                 else
790                     out_s15 <= comp_signal16;
791                 end if;
792                 if(out_s15 > comp_signal17) then
793                     out_s16 <= out_s15;
```

```
794                 else
795                     out_s16 <= comp_signal17;
796                 end if;
797                 if(out_s16 > comp_signal8) then
798                     out_s17 <= out_s16;
799                 else
800                     out_s17 <= comp_signal18;
801                 end if;
802                 if(out_s17 > comp_signal19) then
803                     out_s18 <= out_s17;
804                 else
805                     out_s18 <= comp_signal19;
806                 end if;
807                 if(out_s18 > comp_signal20) then
808                     out_s19 <= out_s18;
809                 else
810                     out_s19 <= comp_signal20;
811                 end if;
812                 if(out_s19 > comp_signal21) then
813                     out_s20 <= out_s19;
814                 else
815                     out_s20 <= comp_signal21;
816                 end if;
817                 if(out_s20 > comp_signal22) then
818                     out_s21 <= out_s20;
819                 else
820                     out_s21 <= comp_signal22;
821                 end if;
822                 if(out_s21 > comp_signal23) then
823                     out_s22 <= out_s21;
824                 else
825                     out_s22 <= comp_signal23;
826                 end if;
827                 if(out_s22 > comp_signal24) then
828                     out_s23 <= out_s22;
829                 else
830                     out_s23 <= comp_signal24;
831                 end if;
832                 if(out_s23 > comp_signal25) then
833                     out_s24 <= out_s23;
834                 else
835                     out_s24 <= comp_signal25;
836                 end if;
837                 if(out_s24 > comp_signal26) then
838                     out_s25 <= out_s24;
839                 else
840                     out_s25 <= comp_signal26;
841                 end if;
842                 if(out_s25 > comp_signal27) then
843                     out_s26 <= out_s25;
844                 else
845                     out_s26 <= comp_signal27;
846                 end if;
847                 if(out_s26 > comp_signal28) then
848                     out_s27 <= out_s26;
849                 else
850                     out_s27 <= comp_signal28;
```

```
851                 end if;
852                 if(out_s27 > comp_signal29) then
853                     out_s28 <= out_s27;
854                 else
855                     out_s28 <= comp_signal29;
856                 end if;
857                 if(out_s28 > comp_signal30) then
858                     out_s29 <= out_s28;
859                 else
860                     out_s29 <= comp_signal30;
861                 end if;
862                 if(out_s29 > comp_signal31) then
863                     out_s30 <= out_s29;
864                 else
865                     out_s30 <= comp_signal31;
866                 end if;
867                 if(out_s30 > comp_signal32) then
868                     out_s31 <= out_s30;
869                 else
870                     out_s31 <= comp_signal32;
871                 end if;
872                 if(out_s31 > comp_signal33) then
873                     out_s32 <= out_s31;
874                 else
875                     out_s32 <= comp_signal33;
876                 end if;
877                 if(out_s32 > comp_signal34) then
878                     out_s33 <= out_s32;
879                 else
880                     out_s33 <= comp_signal34;
881                 end if;
882                 ----------------------------
883                 if(out_s33 > comp_signal35) then
884                     out_s34 <= out_s33;
885                 else
886                     out_s34 <= comp_signal35;
887                 end if;
888                 if(out_s34 > comp_signal36) then
889                     out_s35 <= out_s34;
890                 else
891                     out_s35 <= comp_signal36;
892                 end if;
893                 if(out_s35 > comp_signal37) then
894                     out_s36 <= out_s35;
895                 else
896                     out_s36 <= comp_signal37;
897                 end if;
898                 if(out_s36 > comp_signal38) then
899                     out_s37 <= out_s36;
900                 else
901                     out_s37 <= comp_signal38;
902                 end if;
903                 if(out_s37 > comp_signal39) then
904                     out_s38 <= out_s37;
905                 else
906                     out_s38 <= comp_signal39;
907                 end if;
```

```
908                 if(out_s38 > comp_signal40) then
909                     out_s39 <= out_s38;
910                 else
911                     out_s39 <= comp_signal40;
912                 end if;
913                 if(out_s39 > comp_signal41) then
914                     out_s40 <= out_s39;
915                 else
916                     out_s40 <= comp_signal41;
917                 end if;
918                 if(out_s40 > comp_signal42) then
919                     out_s41 <= out_s40;
920                 else
921                     out_s41 <= comp_signal42;
922                 end if;
923                 if(out_s41 > comp_signal43) then
924                     out_s42 <= out_s41;
925                 else
926                     out_s42 <= comp_signal43;
927                 end if;
928                 if(out_s42 > comp_signal44) then
929                     out_s43 <= out_s42;
930                 else
931                     out_s43 <= comp_signal44;
932                 end if;
933             flag_compare_ready := 1;
934                 ------------------------------------------------
935             else
936                 index_pga := index_pga + 1;
937             end if;
938         end if;
939     end if;
940
941
942
943
944         ------------ STATE DETERMINER ----------------------
945     if(flag_compare_ready = 1) then
946         if (out_s43 < "0000101000") then
947             PGA_CNTRL <= "0011";
948             --PGA_RSSI <= "1000";
949         elsif (out_s43 > "0000101000" and out_s43 < "0001010000") then
950             PGA_CNTRL <= "0111";
951             --PGA_RSSI <= "1100";
952         elsif (out_s43 > "0001010000" and out_s43 < "0001010011") then
953             PGA_CNTRL <= "0010";
954             --PGA_RSSI <= "1110";
955         elsif (out_s43 > "0001010011" and out_s43 < "0001111000") then
956             PGA_CNTRL <= "1011";
957             --PGA_RSSI <= "1111";
958         elsif (out_s43 > "0001111000" and out_s43 < "0001111101") then
959             PGA_CNTRL <= "0001";
960             --PGA_RSSI <= "0111";
961         elsif (out_s43 > "0001111101" and out_s43 < "0010100110") then
962             PGA_CNTRL <= "0110";
963             --PGA_RSSI <= "0011";
964         else
```

```
965                 PGA_CNTRL <= "0000";
966                 --PGA_RSSI <= "0001";
967             end if;
968             flag_compare_ready := 0;
969             out_s43 <= "0000000000";
970         end if;
971 
972 
973         end process pga_proc;
974     -------------------------------------------------------
975 
976 
977     end Behavioral;
978 
```