Bob Delong
6400 Wilkinson Dr
Prescott, AZ 86301

Glenn Ferguson, Chris Taralov,
Bryan Woolsey and Surenthiran Bhuvanenthiran
April 22, 2010
1185 W. University Ave apt# 14-214
Flagstaff, AZ 86001
Ferguson1015@gmail.com

Thank you for being our sponsor for this project. We appreciate your sponsorship of this project.

This document contains an executive summary, a summary of what we did in this project, a list of any changes we made to the project, our budget, an updated schedule, and schematics for our design.

We successfully represented the project at the Undergraduate Symposium at Northern Arizona University by delivering a 30 min presentation about the project as well as having a poster session.

This report contains a detailed explanation of our capstone project, as well as what we were able to accomplish.

At this point we have completed each piece of our design, but have yet to test it as a whole. This is because our filters are so computationally heavy that we use all of the logic gates when trying to run the program. This problem could be circumvented if an external memory is implemented with the FPGA. We have completed as much of this project as we could in the time allotted and thank you again for working with us.

This project was a great opportunity to implement what we have learned from our communications and digital signal processing classes. We now have a better understanding of radio communication as well as digital filtering. We also gained valuable experience of what is the scope of projects in industry.

Additional recipients include: Dr. David Scott, and Dr. Paul Flikkema.

Final Report

Project:

Digital Receiver using a FPGA

Client:

Bob Delong, Wulfsberg Electronics, Cobham Avionics

Team members:

Glenn Ferguson
Surenthiran Bhuvanenthiran
Hristo Taralov
Bryan Woolsey

Table of Contents

I. Executive Summary

In this project, we developed a digital receiver using an FPGA. We chose to program the FPGA in Verilog instead of VHDL so you would be able to extract what we did more easily since they use primarily Verilog code. We also had a choice of Modulation techniques and decided to go with an 8-PAM (Pulse Amplitude Modulation) because it has a high information density and was familiar for the entire group. When actually completing our code, we divided the code into sections to allow you to be able to implement the pieces you choose, and make changes more easily.

The input to our FPGA is a stream of numbers (created in Matlab) that represent a signal with 8-PAM modulation, center frequency of 75 MHz, and pulse-shaped using a square wave. A picture of this can be seen in Figure 10 on page 15.

Programmed into the FPGA is code that begins by downconverting the center frequency to 100 kHz so that the bandwidth ranges from 0 to 200 kHz. The signal is then channel selected using band pass filters at frequencies: 8.33, 16.66, 25, 33.33, 41.66, 50, 58.33, 66.66, 75, 83.33, 91.66, and 100 kHz. After this, the signal is demodulated out of 8-PAM and into an information stream (an estimate of the stream of ones and zeroes that were sent). This information stream can then be converted back into a voice using a DAC and outputted using a speaker.

We have a project website complete with information about the team working on this project, as well as general descriptions about what we did.

II. Background Information

1. Why is this project Relevant?

The Receiver we are designing will be implemented in an aircraft or a helicopter. Good Communication is critical, especially in the air; for example the pilot needs to know when and where the plane can be landed in order to avoid collisions. Furthermore, a military aircraft pilot needs instructions for what target to attack or not. Miscommunications in situations like this can easily lead to unwanted casualties. Therefore it is important that the pilot is able to switch between frequencies if necessary and to have a clear signal on the desired channel.

What this project is accomplishing is a better overall communications system. Most conventional communication is still being done with analog circuitry which is getting outdated. The latest trend in technology is moving everything to the digital domain.

Receivers are an important device in our society since they are major components in products such as cell phones, televisions, etc. Historically, receivers have been constructed entirely out of analog devices but with advances in digital technology, large portions of receivers are now accomplished using Digital Signal Processing (DSP) technology.

Using DSP, many of the functions performed by analog electronics can be performed by software. The benefit is that software is not affected by temperature, manufacturing defects, physical variables, and electronic noise. The disadvantage is a loss of information when converting from analog to digital electronics.

2. Analog vs. Digital Filtering

In filtering there are several notable differences between analog and digital filtering. The first difference is the size and weight difference. Analog filters are bulky; they use up a lot of space and weigh a lot because they are mainly made of metals. Analog filters are made of a lot of different components that often need soldering. These joints can brake due to vibrations or external force. Not only can analog filters limit space and weight capacities, but they can become unreliable when the equipment is under duress. In comparison, digital filters use up practically no space and no weight. Furthermore, it is one solid unit, whose interconnects are virtually impossible to brake due to vibrations.

Another downside of analog filters is that they must be tuned to specific frequencies. If those frequencies change then the filters must be tuned or even changed out. Digital filters do not have to be tuned, and if the operation frequencies change, then digital filters can be reprogrammed.

Here is a sample schematic and filter box of an analog filter

Figure 1 Analog filter Box

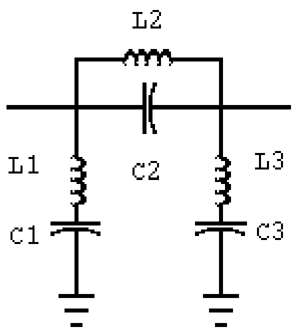


Figure 2 Analog Filter Schematic

An analog filter uses components like inductors and capacitors to physically change the signal, while a digital filter will use samples of the signals and perform mathematical operations such as addition and multiplication in order to achieve the same results, as shown in Figure 3.

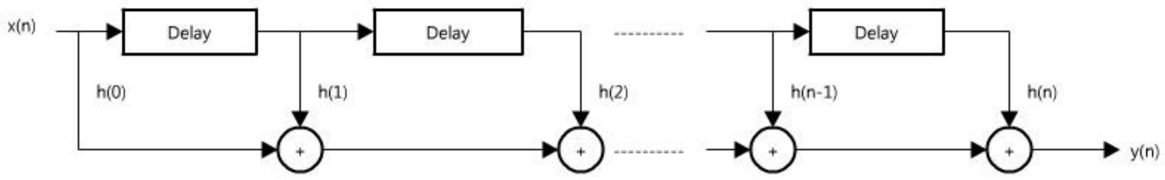


Figure 3 Example FIR filter

The advantage of analog signals is that they manipulate the entire wave and all of the operations they do are continuous, while digital filters lose some information when sampling.

However analog components are highly dependent on their surroundings. Factors like heat and electro – magnetic interferences will change the properties of the analog components and therefore change filter's performance. This is not typically a problem while using digital filters except in extreme cases.

3. FPGA vs. ASIC

There are two different chips that could have been put at the core of this digital filter: a FPGA or an ASIC. Field programmable gate arrays (FPGAs) have wider potential than application-specific integrated circuits (ASICs) because they can be programmed in the field even after customer installation, allowing for future upgrades and enhancements. FPGAs contain programmable logic in a hierarchy of reconfigurable interconnects which allow them to perform complex combinational functions as well as basic logic gates. Figure 4 below is the Cyclone II FPGA chip being used for this project.

Figure 4 Cyclone II chip

The chip above is being used along with the DE2 development board (picture below). The board makes testing and interfacing the chip easier.

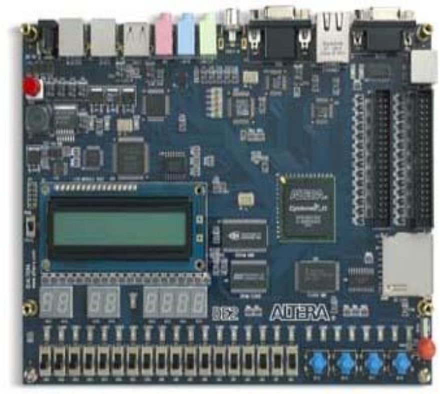


Figure 5 DE2 Development board

ASICs are major competitors with FPGAs. The difference between the two is that ASIC is designed with one purpose in mind. When it comes out of the manufacturer its function is already fixed and cannot be changed. It has no excess hardware, which makes it cheaper. Therefore, ASICs are better for high volume applications. Below is an example of a sample ASIC chip.

Figure 6 Sample ASIC Chip

ASICs do not have any external hardware like the FPGA does and therefore cost less per unit then and FPGA. However if there is a mistake in the engineering design or a change in the specifications, ASICs cannot be changed to reflect these new designs which forces the engineer to redesign the entire chip. In comparison FPGA can be easily reused just by adjusting the software for it.

Table 1 below summarizes the differences between FPGA and ASIC

Table 1 Comparison between FPGA and ASIC

	FPGA	ASIC
Cost per capita	High	Low
Engineering cost	Low	High
Reconfigurable Hardware	Yes	No
Volume	Low	High

III. Problem Overview

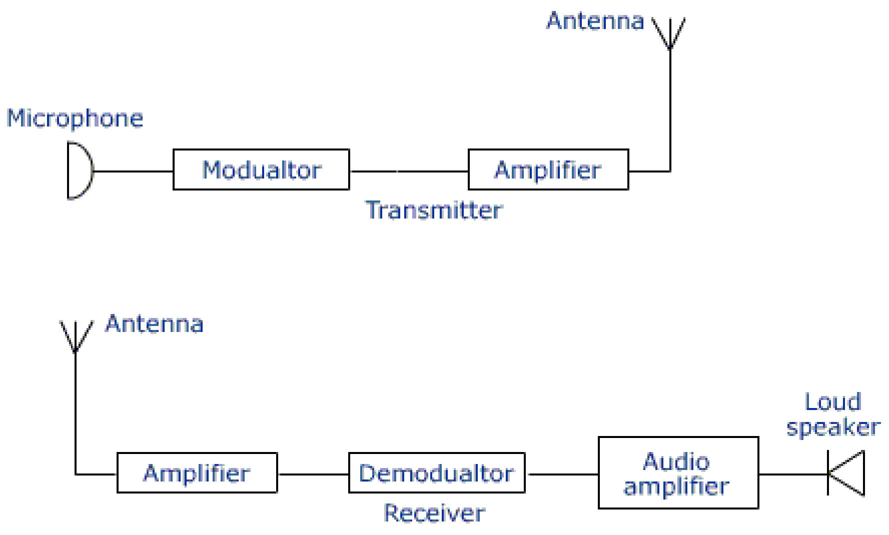


Figure 7 Project diagram

Fig 7 above shows the top system level diagram for this project. The upper part of the diagram is a transmitter. Audio is passed through a microphone, converted to a wave, modulated and passed via an antenna to the receiver. The receiver's antenna will pick up the signal and pass it to an analog portion which will consist of filtering, amplification and conversion to the digital domain. The last blocks will be done by the FPGA and will consist of demodulation, or extracting the information from the signal. Further processing includes channel selection, filtering amplifications; and finally outputting an audio wave to speakers or headphone set.

IV. Specifications

We received a specifications document from our client giving details about what was required in the finish product. The entire document can be found in Appendix V Client Specifications

We were tasked with developing a digital receiver using an FPGA, with the main focus on developing topology and working VHDL/Verilog code to meet the specifications.

The FPGA needed to be interfaced with an ADC converter for input, and a DAC converter for output. The FPGA also had to be interfaced with the user for channel selection

We were given a requirement that the channel frequency of the input had to be 75 MHz And the external receiver bandwidth had to be 200 kHz.

The following are Additional requirements that the client wished us to complete.

- Digital signal processing requirements
    - Handle two different channel spacing
        - 25 khz channels
            - -6 dB maximum @ ± 10 kHz
            - -40 dB minimum @ ± 17 kHz
            - -60 dB minimum @ ± 22 kHz
        - 8.33 khz channels
            - -6 dB maximum @ ± 2.78 kHz
            - -60 dB minimum @ ± 7.37 kHz
    - Audio leveling
        - The output should include automatic level control for less than 3 dB audio output variation
    - Dynamic range
        - 100 dB, of which 40 dB should be accomplished in the FPGA
    - Modulation
        - Choice of amplitude and/or frequency modulation
- Other requirements
    - Code language
        - Choice of Verilog or VHDL--Verilog is preferred
    - Operating temperature range:
        - -30°C to +85°C

V. Trade offs

1. VHDL vs. Verilog

The first trade-off that we came across was an option given to us by the client. This was whether to program the FPGA in VHDL or Verilog. We chose to go with Verilog, because our client uses it in their designs and thus would be able to understand what we were doing much easier. This would allow the client to be able to make changes to modules of our code much easier.

2. Full-range sampling vs. Selective sampling

The next decision that we were going to have to make was how we were receiving our information. This came down to two methods: Full-range sampling and Selective sampling. Selective sampling is when you a signal is at the IF frequency (in our case 75 MHz) and then downconverted and sent through the ADC. This allows the FPGA to use smaller frequencies, and thus less logic in the FPGA. The problem with this approach is that it requires external components. The other choice, full-range sampling, is where the signal at the IF frequency is sampled using an ADC and the results of this are sent straight to the FPGA. We felt that full-range sampling was more in line with what the client was looking for in this project.

3. Finite Impulse Response (FIR) vs. Infinite Impulse Response (IIR)

An Ideal filter is unstable, and needs infinite numbers of samples therefore it is impossible to implement. Two types of digital filters are FIR and IIR. IIR filters have a feedback loop which will address the infinitive amounts of sampling however delaying the single a few times this filter does not produce constant delay at the output which needs to be adjusted for. While a FIR filter may need more delays and is a higher order computation however it will produce a constant delay in the output. Refer to the Frequency Response Plots in Appendix I. The constant delay is indicated by the linear phase in the pass band region.

How and FIR filter operates

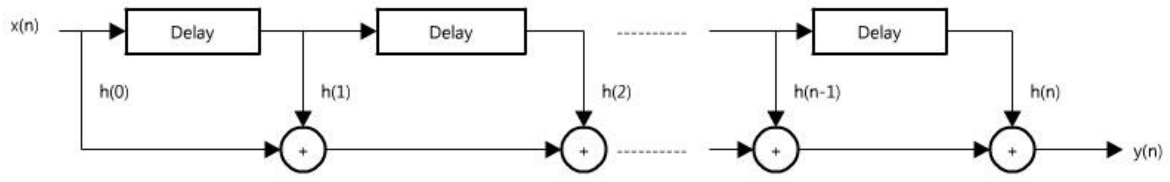


Figure 8 Fir filter schematic

The above figure is part of the Finite Impulse Response filter schematic implemented into the FPGA.. h(n) is referred as “taps”. x(n) is the sampled signal. The filter operates by multiplying different samples of the signal by the tap value and then summing the result

When you pass a signal through a filter the output is ideally a new signal without the unwanted or filtered frequencies. The filter response in the frequency domain shows what frequencies are passed wherever the wave is equal to one.

Design Process

This section summarizes the process flow of a two-way communication using the Digital receiver we have designed.

1. Information Generation
   This will be the voice of a pilot or control tower person talking to a microphone. Since implementing this is outside our scope we used a random generation of a binary stream of numbers.

2. Transmission
   In this section the information is encoded, applied to a carrier wave and sent via an antenna. The encoded part is needed to limit the effects of noise in the information; in our case, we use an 8pam modulation. It is normal to see a difference of 100 dB between transmitted signal and received signal, in order to ensure that the signal reaches the receiver it needs to be modulated with a carrier wave, for this project the carrier is a sinusoidal centered at 75 MHz

3. Channel
   Here the signal is in transition from the transmitter, travels through a medium (usually air), and gets received in the receiver. In this transmission, the signal endures various deformations due to interference frequencies , reflections, thermal noise generated by outside influences and even its own components. In order to simulate this we use an Added white Gaussian noise (AWGN)

4. Analog to Digital transformation
   In this section the received wave is converted from an analog wave to digital by sampling. This project uses an ADC that will take samples at a frequency of 300MHz

5. Down Conversion
   In this section, the signal is multiplied by the carrier wave. The results will be a signal that has two frequency components. One centered at the channel frequency (low frequency component) and one centered at twice the carrier frequency (high frequency component) the high frequency component will be filtered using a FIR filter.

6. Channel Selection
   Here the FPGA will filter all frequencies except the desired channel frequency. This is done by the use of FIR filters, The FIR coefficients will be stored in memory and the switches on the DE2 board will determine which filter is applied.

7. Demodulation

This section just reverses the modulation and outputs an information sequence consisting of the binary sequence that was sent across the medium.

8. Voice reconstruction
The binary sequence is sent through a Digital to Analog (DAC) to reconstruct the wave back to the voice wave.

VI. Design Details

Digital Signal Processing (DSP) is used in our project to extract the information out of a modulated wave. In our project, we do this using an FPGA. The signal that we receive is in the form of a string of ones and zeroes encoded into a wave with an IF frequency of 75 MHz, using an 8-PAM modulation technique, which is pulse shaped using a square wave. A picture of this can be referenced in Table 2 (page 14). The first thing the FPGA does upon receiving the wave is down conversion.

1. Downconversion

In order to downconvert a signal, the signal needs to be multiplied by another wave form at the in order to change the nature of the signal. Mathematically what this does is:

$\cos(2\pi * f_1) * \cos(2\pi * f_2) = \cos(2\pi f_1 - 2\pi f_2) + \cos(2\pi f_1 + 2\pi f_2)$

This means that if f_1 and f_2 are equal, then you will get:

$1 + \cos(2*(2\pi f))$

Which equals:

$(\cos(2\pi f))^2$

This makes sense considering that you are essentially multiplying a wave by itself.

The purpose of downconversion is to bring the center frequency down to make the signal easier to process. This process is not completed yet though. In order to complete the process, the signal needs to filter out the $\cos(2\pi f_1 + 2\pi f_2)$. A low pass filter will do just fine for this, and will leave just $\cos(2\pi f_1 - 2\pi f_2)$ as a result. This creates a new wave that has a frequency f3 = f1 – f2 and looks as follows: $\cos(2\pi(f_1 - f_2)) = \cos(2\pi f_3)$.

2. Channel Selection

For the next portion, we had to be able to select a channel. A channel is a specific frequency that is listened to in a specified bandwidth to the exclusion of all of the other channels. This could be compared to listening to stations on an FM radio: The listener wants to listen to one station at a time, not all at once.

In order to select a channel, the signal needs to be filtered by a band pass filter. This filter just filters a specific frequency to the exclusion of all of the others. Refer to the Filter Design section for more details.

3. Demodulation

At this point the signal still looks like the waveform in Figure 1, but at a lower frequency. In order to properly define how we demodulate, we first have to define how the wave is created in the first place.

To create the wave, we had to turn the voice into binary, and then took every three bits and represented this as a symbol as represented in table 2 below.

Table 2 8 pam symbol mapping

Bit block	8pam symbol
0 0 0	-7
0 0 1	-5
0 1 0	-3
0 1 1	-1
1 0 0	1
1 0 1	3
1 1 0	5
1 1 1	7

The right column represents the amplitude of each of the symbols. The difference between the positive and the negative is just a 180 degree phase change. We decided to create this using a square wave, but different combinations can be created. At this point, the transmitter should have something that looks like what we have below:

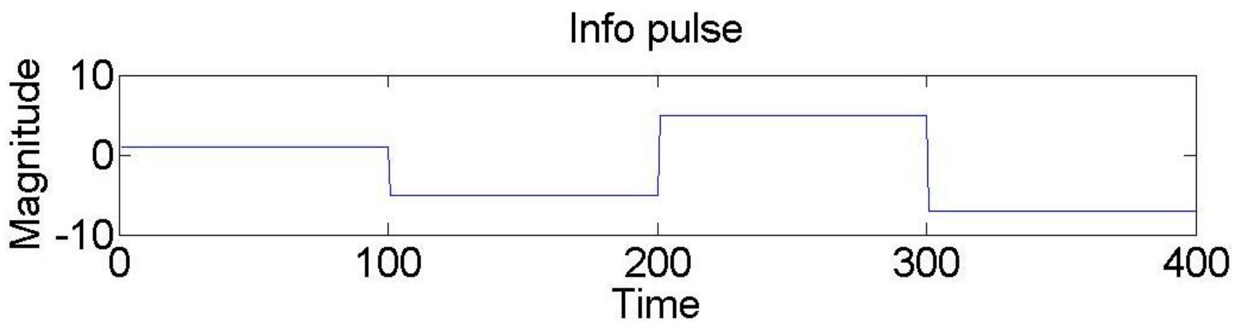


Figure 9 Pulse shaped symbol mapping

After this the carrier wave (in our case cos[2π*75000000])is then multiplied by this signal creating something like what we have below:

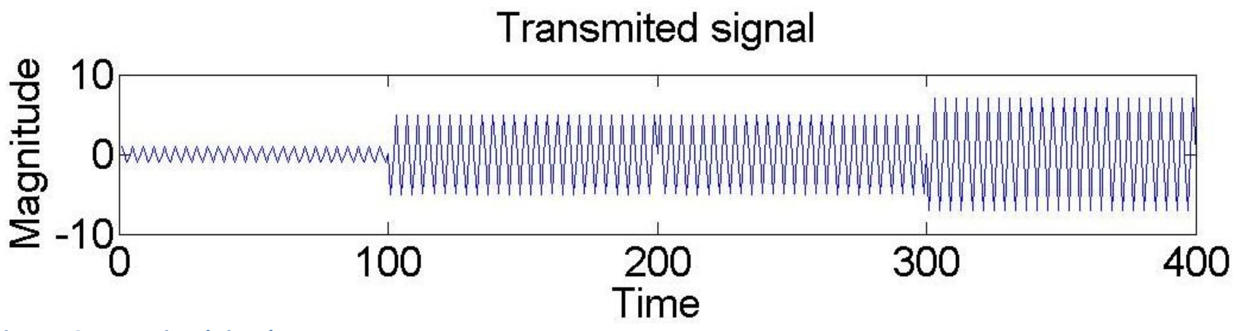


Figure 10 Transmitted signal

Now, in demodulation we essentially reversed this process. The first step of this process is to get rid of the carrier wave. This can be done by downconverting (see downconversion section above for more details) the carrier wave down to zero. If you multiply this wave by the carrier wave again and then filter out the high frequency component: $\cos(2\pi f_1 + 2\pi f_2)$. This will get us back to a signal very close to the square wave and will look very similar to Figure 3.

In order to get our symbol values back, we need to sample this wave form and round the result of that to the nearest symbol.

At this point we should just have a series of symbols; received in the order they were sent. The symbols can be converted using the same table we used to create the square waves. When the information is converted to binary, then the signal can be converted back into a sound wave.

4. Filter Design

In this project, two types of filters have been used. The first are Band Pass Filters (BPF), and the second are Low Pass Filters (LPF). Band Pass Filters are used to select a specific frequency range and block the rest. Low pass Filters are used to block high frequencies while allowing all low frequencies below a certain cutoff frequency.

In our project, both set of filters are implemented using Finite Impulse Response filters (FIR). For more details about FIR filters and their comparison to Infinite Impulse Response (IIR) filters refer to Trade off portion of the Design details section.

The band pass filters are used in channel selection and are designed using an Equal ripple approach. The low pass filters were designed using a Kaiser Windows approach. Both of these are explained below.

a. Band Pass Filter Design

Step 1: Use specifications to form the Frequency response.

Figure 11 below shows the general frequency response limits that each band pass filter has.

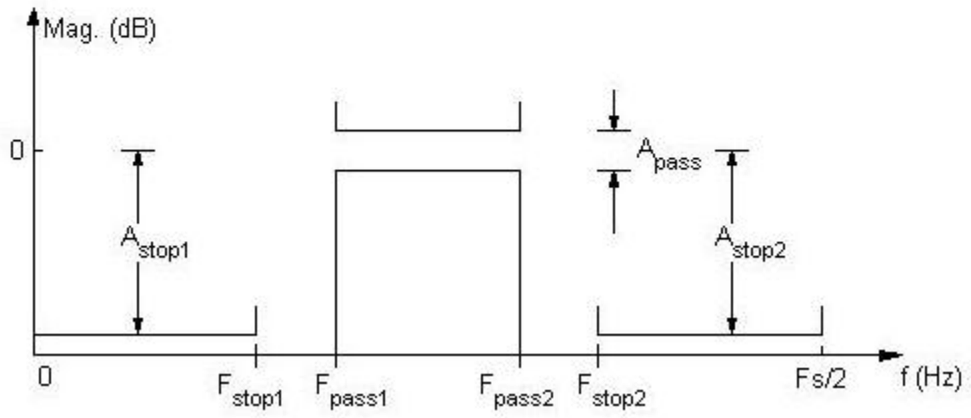


Figure 11 Band Pass Filter Specification Frequency Response

Table 3 below explains what each parameter is and what parameters we used in order to meet our specifications.

Table 3 Band pass filter specifications

Unit	Description	Value used for 25kHz channel spacing	Values used for 8.33 kHz channel spacing
Fs (kHz)	Sampling frequency	800	800
A_{pass} (dB)	Pass band ripple	1	1
A_{stop1} (dB)	Stop Band attenuation	65	65
A_{stop2}(dB)	Stop Band attenuation	65	65
F_{stop1} (kHz)	Stop frequency	Channel -17	Channel -7.37
F_{stop2} (kHz)	Stop frequency	Channel +17	Channel +7.37
F_{pass1}(kHz)	Pass frequency	Channel -10	Channel -2.78
F_{pass2}(kHz)	Pass frequency	Channel +10	Channel +2.78
F_{pass1}-F_{pass2}(kHz)	Pass band region		

0-F_{stop1}(kHz)	Stop Band region
F_{stop2}-Fs/2 (kHz)	Stop Band region
F_{sto1}-F_{pass1} (kHz)	Transition region
F_{pass2}- F_{stop2} (kHz)	Transition region

Where channel is the channel frequency ex: 25 kHz, 50 kHz, 75 kHz…200 kHz, 8.33 kHz, 16.66 kHz…200 kHz

Attenuation is how much power is changed in the signal, which means that the dB values are negative.

The specifications require a 60 dB attenuation in the stop band at +/- 22kHz, however the filter gradually gets to the value specified by A_{stop1} and A_{stop2}. In order to make the slope in the transition band more acute, the attenuation limit is increased by 5 dB and the stop band frequencies were chosen at +/- 17 kHz away from the channel frequency.

The sampling frequency was chosen at 800 kHz to minimize the order of the filter. The higher the sampling frequency, the more the filter has to reject specific frequencies. 800 kHz was chosen in order to have at least 4 samples per period at the highest channel frequency (200 kHz).

Step 2: Converting Frequency response to time response

In this step, the filter coefficients or taps are calculated using an Equiripple method. In this method, the ideal time response is approximated using the Ramez Exchange algorithm. The next few paragraphs will discuss basic theory of what the Ramez Exchange algorithm does.

In the following example the channel frequency is 100 kHz all other parameters are as in table 3

An ideal filter Frequency response will look like figure 12 below. Notice that an ideal filter will have both a negative and positive components. This is because cosine is an even function.

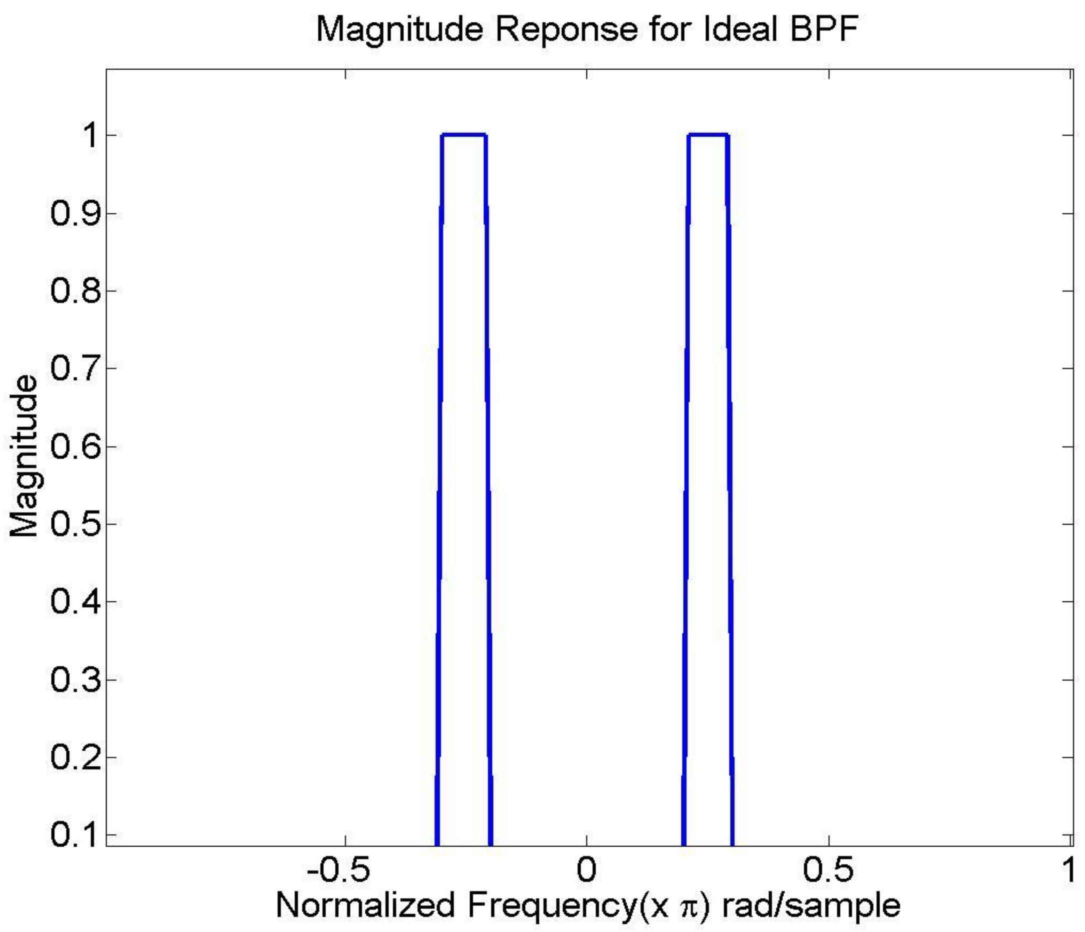


Figure 12 Magnitude Response for Ideal BPF

The first step of the algorithm is to transform the Ideal frequency response to time response. This is done by applying the Inverse Fourier transform

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\ e^{2\pi i x \xi}\, d\xi,$$

The result is a sync function plotted on the graph below (Figure 13)

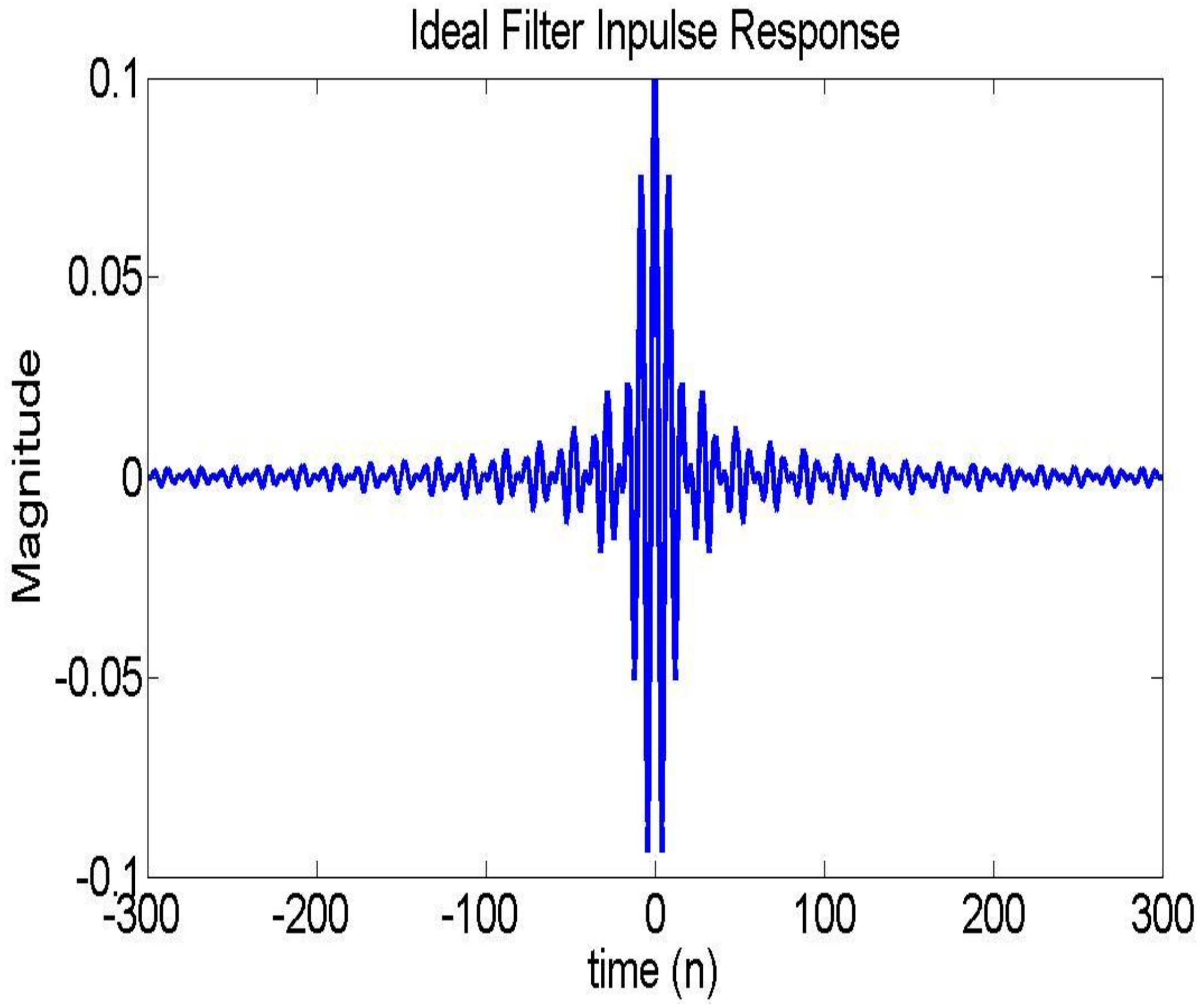


Figure 13 Ideal Filter Impulse Response

An ideal filter uses both past and present values, which will make the filter unstable. Furthermore the sync function spans from negative infinity to infinity, which makes it impossible to implement.
The Ramez Exchange algorithm will approximate the above impulse response by solving the following system of equations:

$$\begin{bmatrix} H(\omega_1) \\ H(\omega_2) \\ \vdots \\ H(\omega_K) \end{bmatrix} = \begin{bmatrix} 1 & 2\cos(\omega_1) & \dots & 2\cos(\omega_1 L) & \frac{1}{W(\omega_1)} \\ 1 & 2\cos(\omega_2) & \dots & 2\cos(\omega_2 L) & \frac{-1}{W(\omega_2)} \\ \vdots & & & & \\ 1 & 2\cos(\omega_K) & \dots & 2\cos(\omega_K L) & \frac{(-1)^K}{W(\omega_K)} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_L \\ \delta \end{bmatrix}$$

Where $W(\omega_k)\,\delta$ is the weighted ripple amplitude at frequency ω_k . $W(\omega_k)$ is a ripple weighting function defined by the parameters listed . $H_{(wk)}$ is the frequency response and h_n is the impulse response

The result is then shifted, in order to use only past values. This makes the filter causal and stable. The result is plotted on Figure 14 below

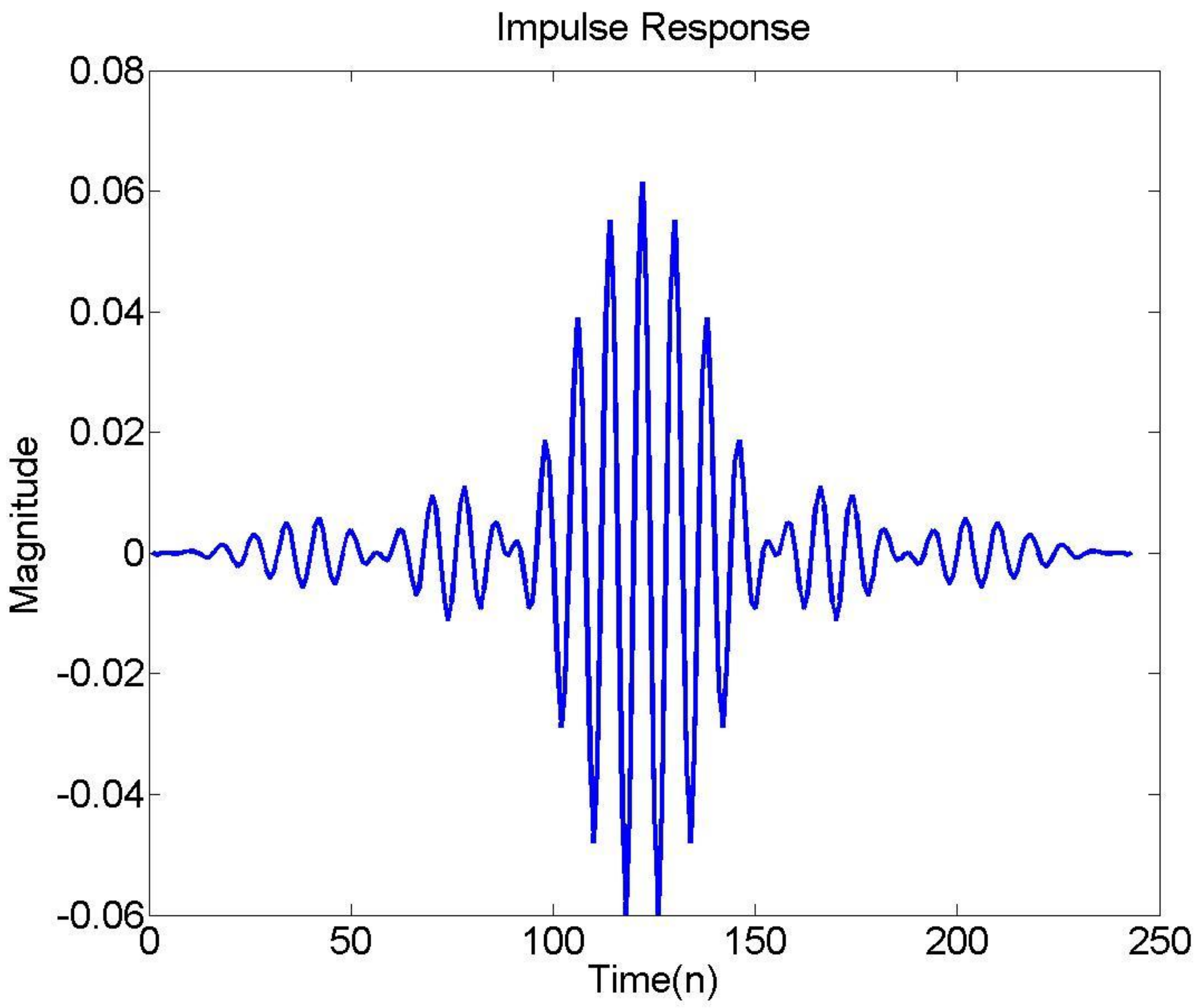


Figure 14 Impulse response

Step 3. Convert Impulse response to frequency response.
This is done by applying the Fourier transform

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\ e^{-2\pi i x \xi}\,dx,$$

The result magnitude and phase are plotted on Figure 15 below

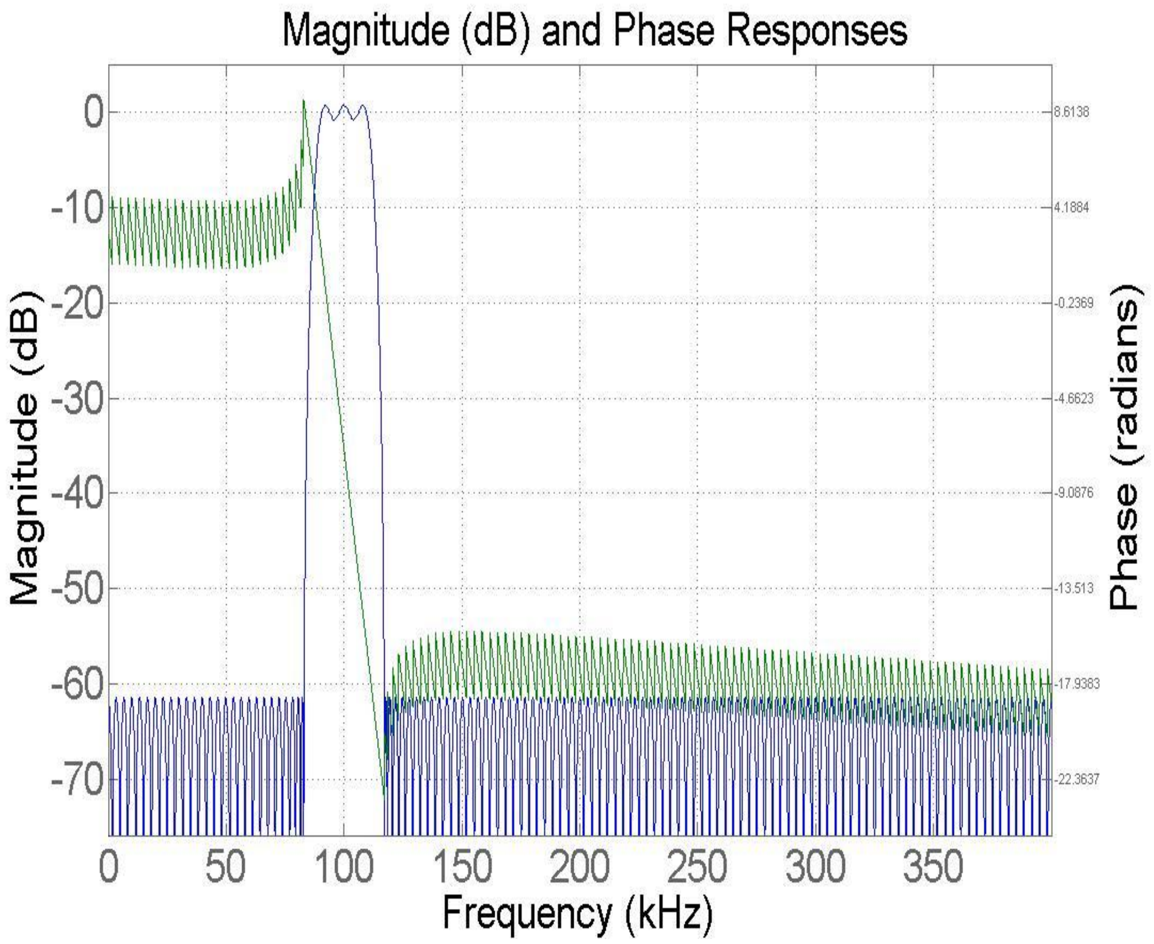


Figure 15 Magnitude and Phase responses

This graph helps evaluate how good of a filter was designed and weather it meets the client specification.

b. Low Pass Filter Design

Step 1: Use specifications to from the Frequency response

Figure 16 below shows the general frequency response limits that each low pass filter has.

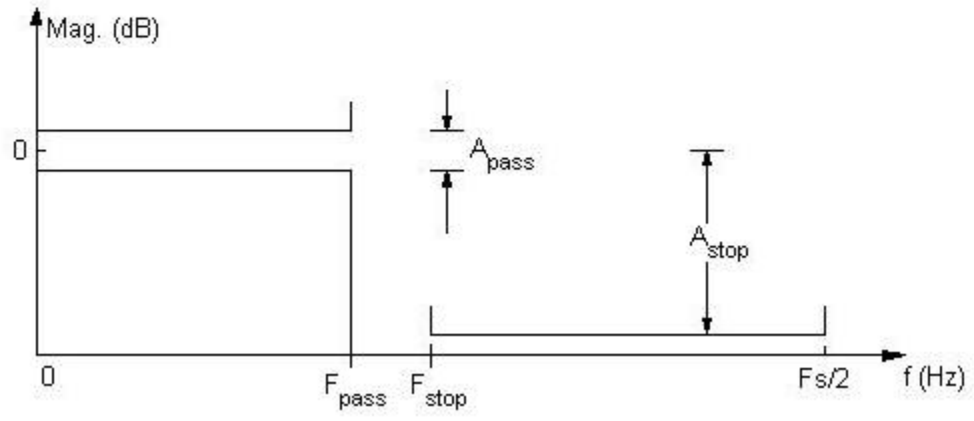


Figure 16 Low Pass Filter Specifications

Table 4 below explains what each parameter is and what parameters we used

Table 4 Low pass filter properties

Unit	Description	LPF1	LPF2
Fs (kHz)	Sampling frequency	300 000	800
A_{pass} (dB)	Pass band ripple	1	1
A_{stop} (dB)	Stop Band attenuation	90	90
F_{stop} (Hz)	Stop frequency	600	200

F_{pass}(Hz)	Pass frequency	5000	250
0-Fpass (Hz)	Pass Band region		
Fpass -Fs/2(Hz)	Stop Band region		
Fpass-Fstop (Hz)	Transition region		

Fs for the LPF1 Is the same as the external ADC speed 300 MHz in order to filter the high frequency component after down conversion. The reason why the transition region is so large is because the frequency being filtered is centered at around 150 MHz. For the second LPF however the sampling frequency is the same as the one in the channel selections in order to guarantee a bandwidth of 200 kHz.

Step 2: Converting Frequency response to time response
Below is a graph of an Ideal Frequency response for and LPF filter.

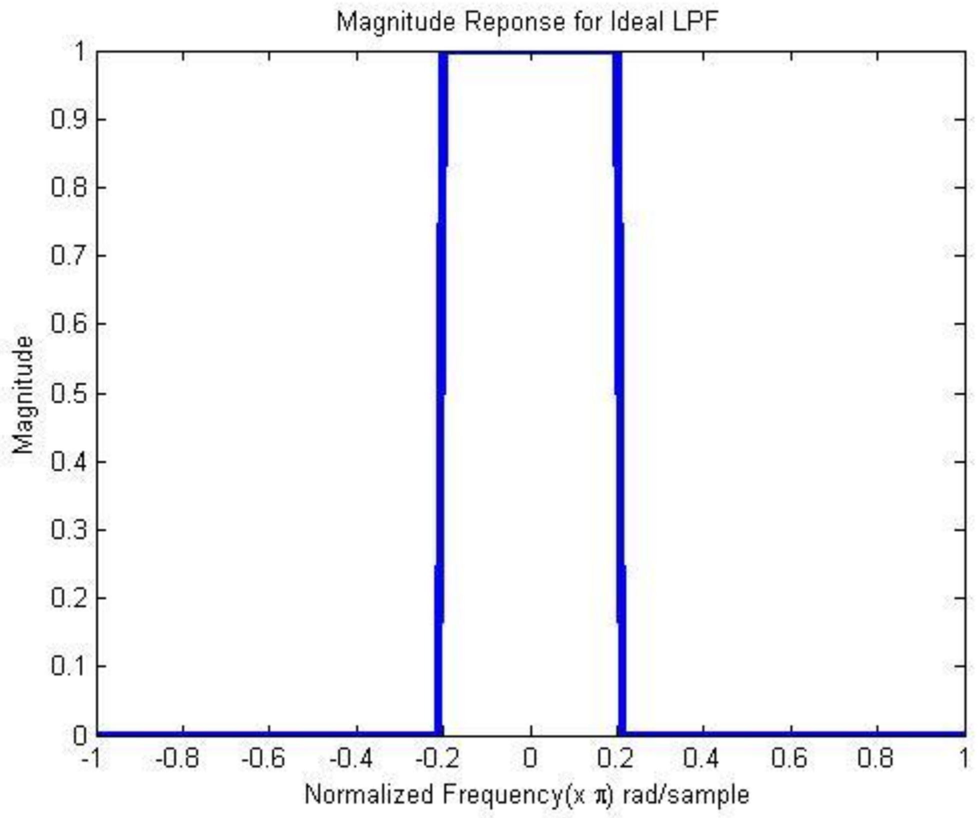


Figure 17 Magnitude Response for Ideal LPF

The first step of the algorithm is to transform the Ideal frequency response to time response. This is done by applying the Inverse Fourier transform

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\ e^{2\pi i x \xi}\, d\xi,$$

The result is a sync function plotted on the graph below (Figure XX)

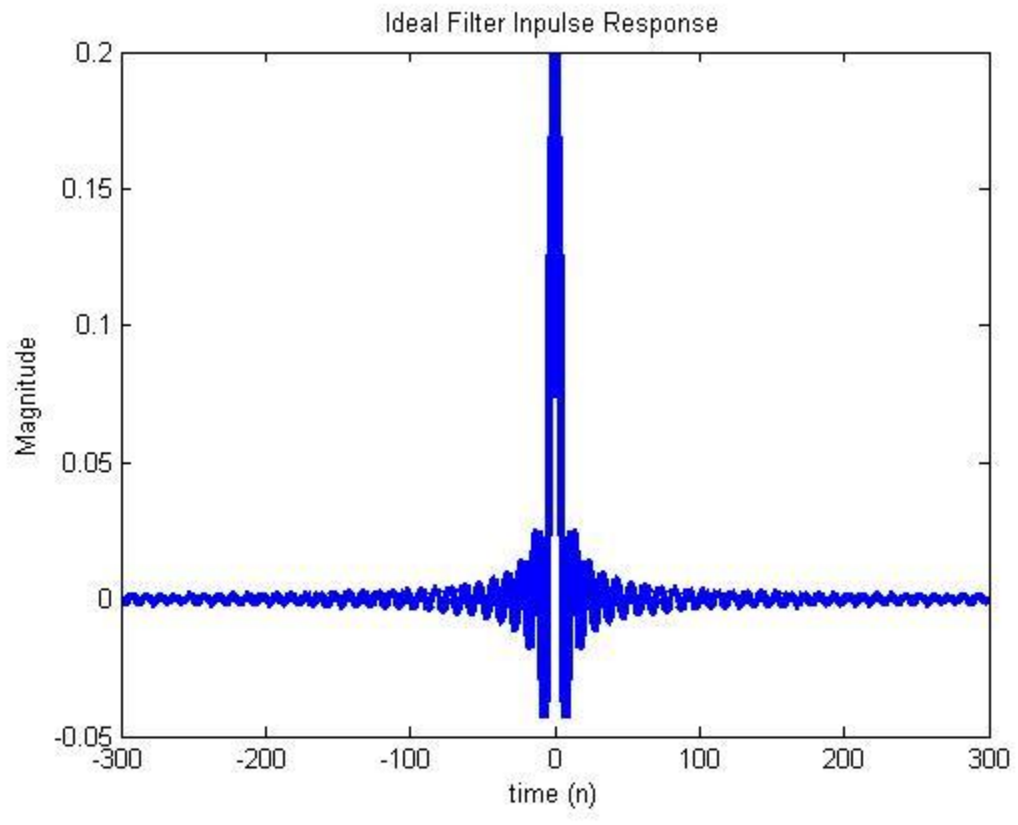


Figure 18 Ideal Filter Impulse Response

Next, the Impulse response is multiplied by a window function.
The window being used is called a Keiser window. Defined by the following equation:

$$w_n = \begin{cases} \dfrac{I_0\left(\pi\alpha\sqrt{1-\left(\frac{2n}{M}-1\right)^2}\right)}{I_0(\pi\alpha)}, & 0 \le n \le M \\ 0 & \text{otherwise} \end{cases}$$

The function is plotted below. For this example we used $\pi\alpha$ of 80 (order of 80). In this case the ideal response is shifted first in order to make a casual filter.

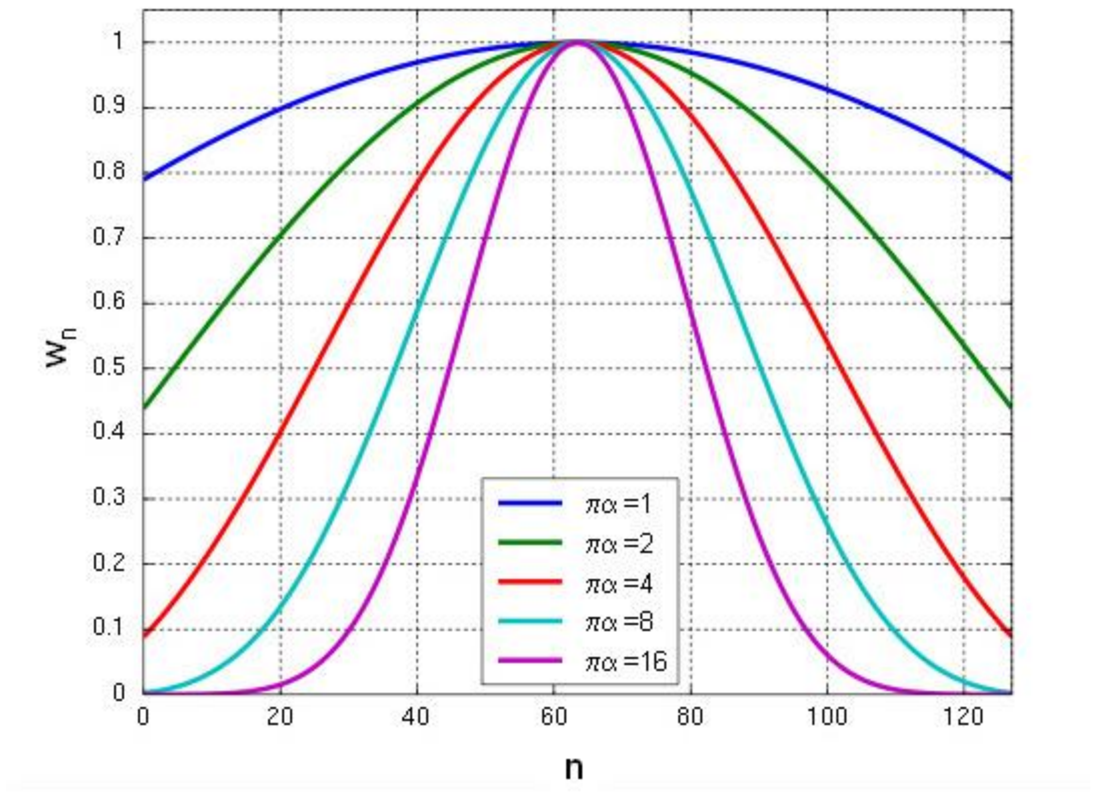


Figure 19 Keizer Widow for different α

Above is the Kaiser Window Function for different alpha values

The results of multiplying the two functions together and is plotted below in Figure 20

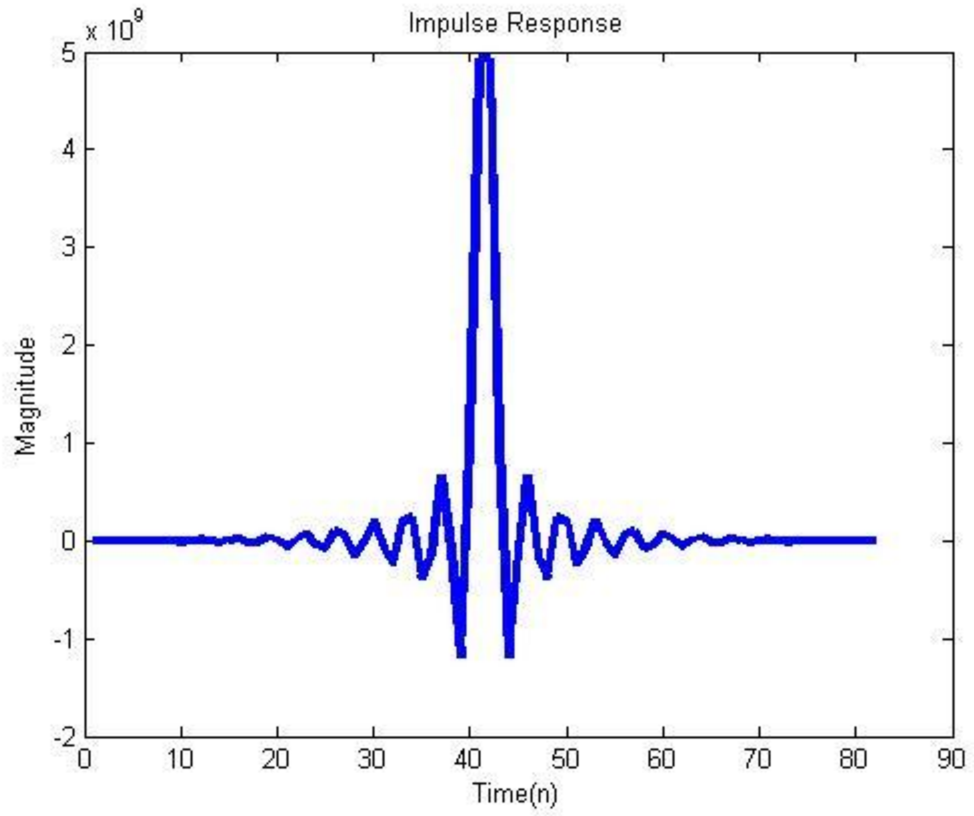


Figure 10 Impulse Response

Step 3: Convert Impulse response to frequency response.

This is done by applying the Fourier transform:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x \xi}\, dx,$$

The resulting magnitude and phase are plotted on Figure 21 below

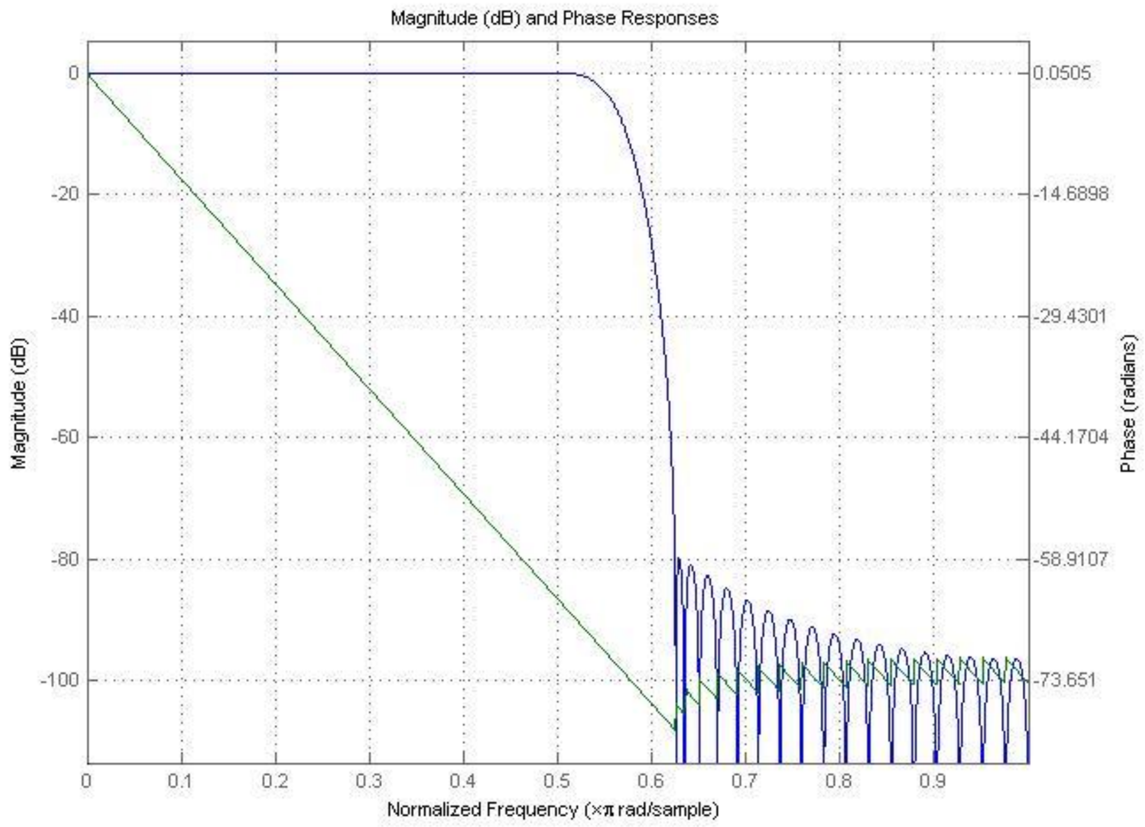


Figure 11 Magnitude and Phase Responses

This graph helps evaluate how good of a filter was designed

5. Programming

When programming the FPGA, all of the above concepts had to be incorporated into the code. We broke the code up into sections much like above, so the client could see which block of code goes with what, and can change one of the modules without having to change the entire code.

In the downconversion section, we read in a text file consisting of one period of the carrier wave and multiply this by the bits entering the FPGA. The result of this is sent through a low pass filter with its taps stored in a lookup table on the chip and are convoluted with the incoming data.

In the Channel select portion, the taps for this filter are also in a lookup table on the chip and convoluted with the data, but here the code checks which switches are selected using a binary code for the first 4 switches. The code ranges from [0 0 0 0] to [1 0 0 0] (numbers refer to the switches on the board, 1 for an “on” position and 0 for an “off” position).

In the demodulation section, the downconversion is implemented just the way it is explained above. In the sampling portion, the wave is sampled every 50 bits, and is averaged to get the result. In the symbol conversion portion, the signal is sent through a case statement, 4 bits at a time.

The result of this is outputted from the FPGA and can be reconstructed back into a sound.

6. Sound reconstruction

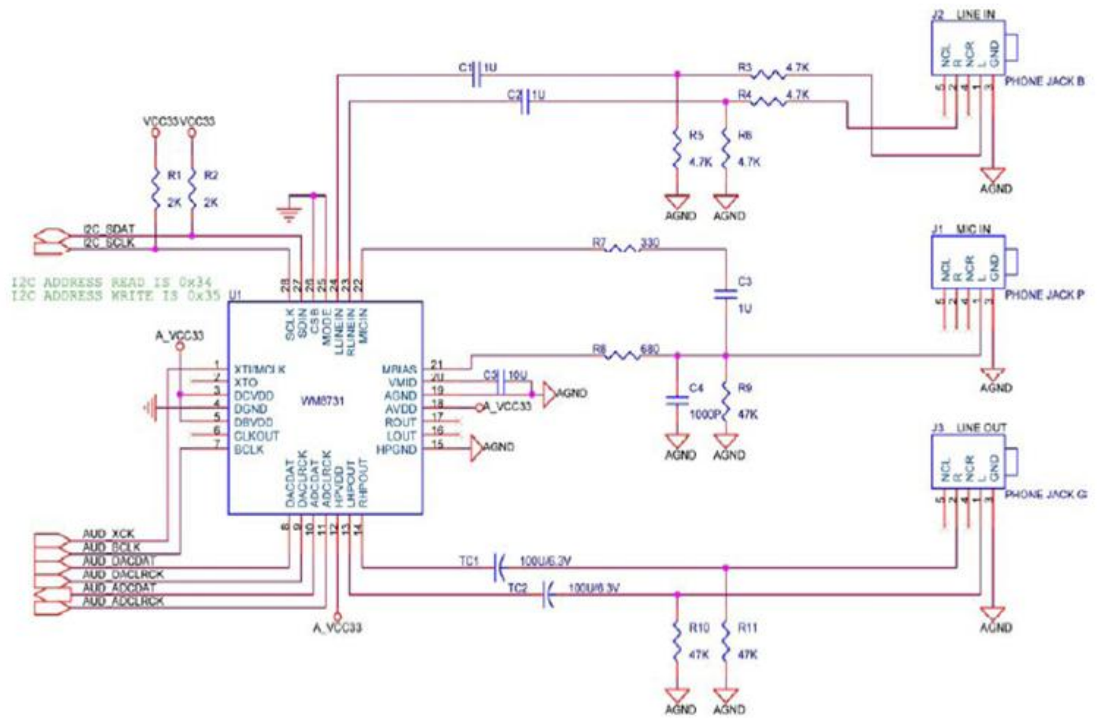
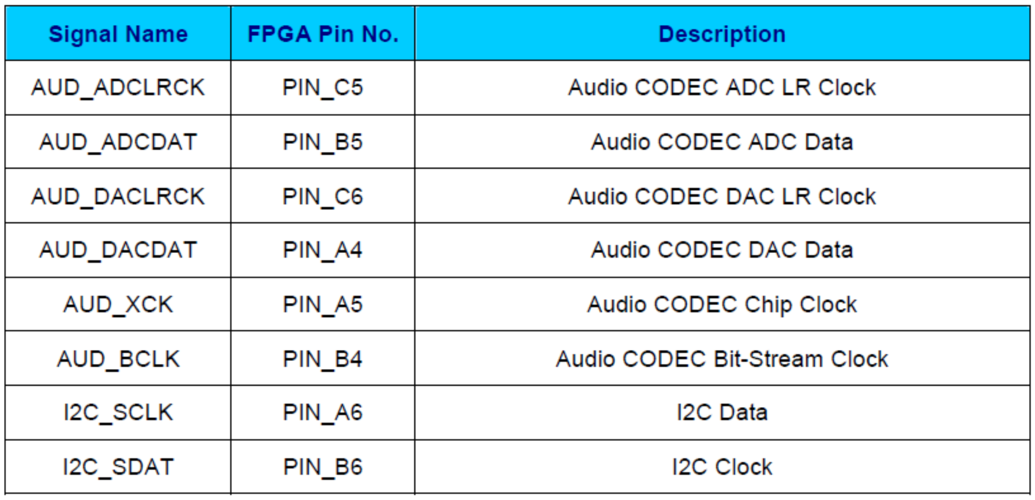


Figure 12 I/O for Codex

The DE2 board provides high-quality 24-bit audio via the Wolfson WM8731 audio CODEC (enCOder/DECoder). This chip supports microphone-in, line-in, and line-out ports, with a sample rate adjustable from 8 kHz to 96 kHz. The WM8731 is controlled by a serial I2C bus interface, which is connected to pins on the Cyclone II FPGA. A schematic diagram of the audio circuitry is shown in Fig 12 and the FPGA pin assignments are listed in Table below:



Signal Name	FPGA Pin No.	Description
AUD_ADCLRCK	PIN_C5	Audio CODEC ADC LR Clock
AUD_ADCDAT	PIN_B5	Audio CODEC ADC Data
AUD_DACLRCK	PIN_C6	Audio CODEC DAC LR Clock
AUD_DACDAT	PIN_A4	Audio CODEC DAC Data
AUD_XCK	PIN_A5	Audio CODEC Chip Clock
AUD_BCLK	PIN_B4	Audio CODEC Bit-Stream Clock
I2C_SCLK	PIN_A6	I2C Data
I2C_SDAT	PIN_B6	I2C Clock

Figure 13 Pin-out for Codex

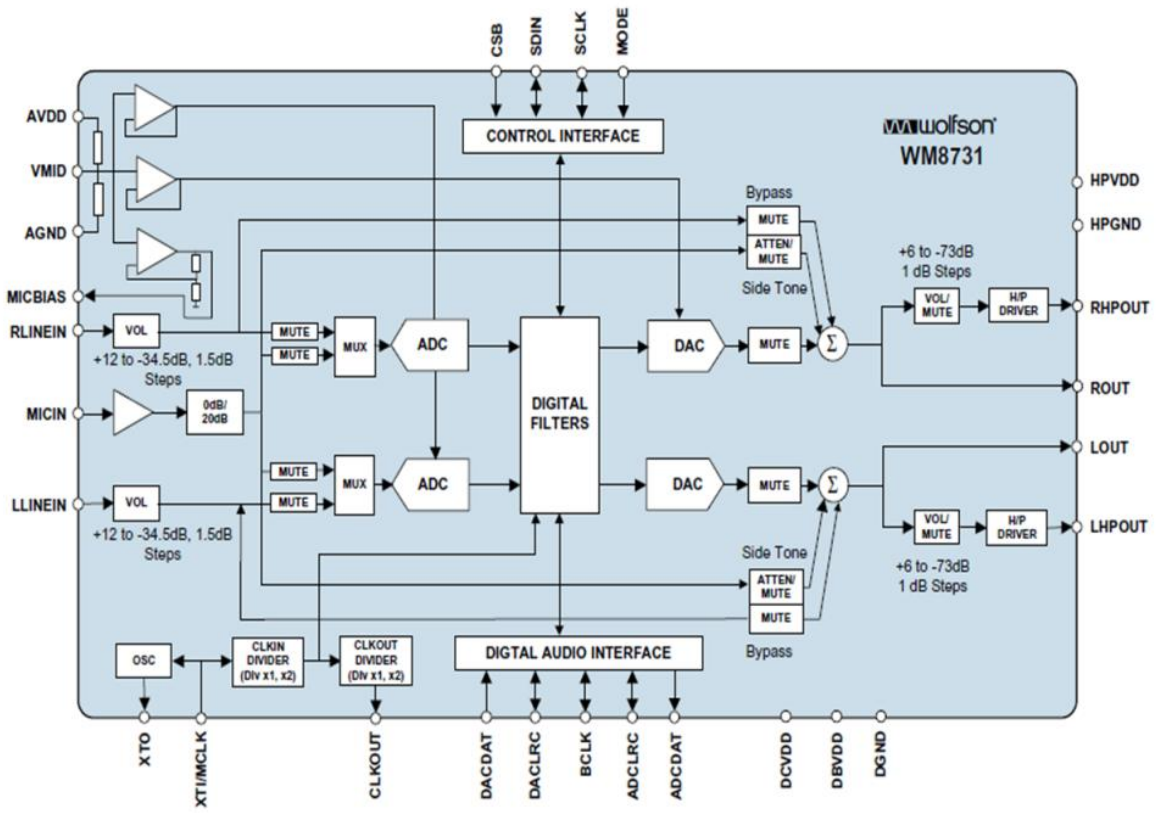


Figure 14 Codex Hardware diagram

VII. Testing

The written code will simulate a transmitted signal, by generating a random sequence of binary values. A square wave will be created from the binary signal, and that signal will be multiplied by a carrier frequency. We then simulate a channel by adding a white Gaussian noise.

The transmission testing code can be viewed by referring to Appendix II: TX Matlab Code.docx.

Down conversion

For the down conversion section, we have Verilog code that will create a stream of data 12 bits long, and convert it into a decimal value. We then send that data through 2 programs, one Matlab, and the other ModelSim. Matlab will then down convert the signal, and display a wave, based on the data being input. For the ModelSim, the data will be sent through the Verilog down conversion code, and that will subsequently output a wave. We then compare the wave from Matlab with the wave from ModelSim, to verify that our Verilog code works. We could not successfully run the data through the Verilog code as we did not have sufficient time to address an error with the Verilog code. For more information on the code, refer to Appendix III: Verilog.docx.

Filtering

The testing for the filter is incomplete due to the fact that the program we compiled uses more memory than the FPGA (Field Programmable Gate Array) can handle. Currently we have the capability to generate, modulate and send information on multiple channels simultaneously. The filters can be tested by convolving the simulated received sequence with the filter tabs, however we were not able to run the signal trough the FPGA and therefore we could not compare them to the Matlab simulations. For more information on the code, refer to Appendix II: TX Matlab Code.docx.

VIII. Results

1. Transmission Code

To view the transmission code please refer to the Appendix I: TX Matlab Code.docx
The actual Matlab file and generated files by it are combined into the file Transceiver.

Figure 25 below shows what an example of what the transmission code will do.

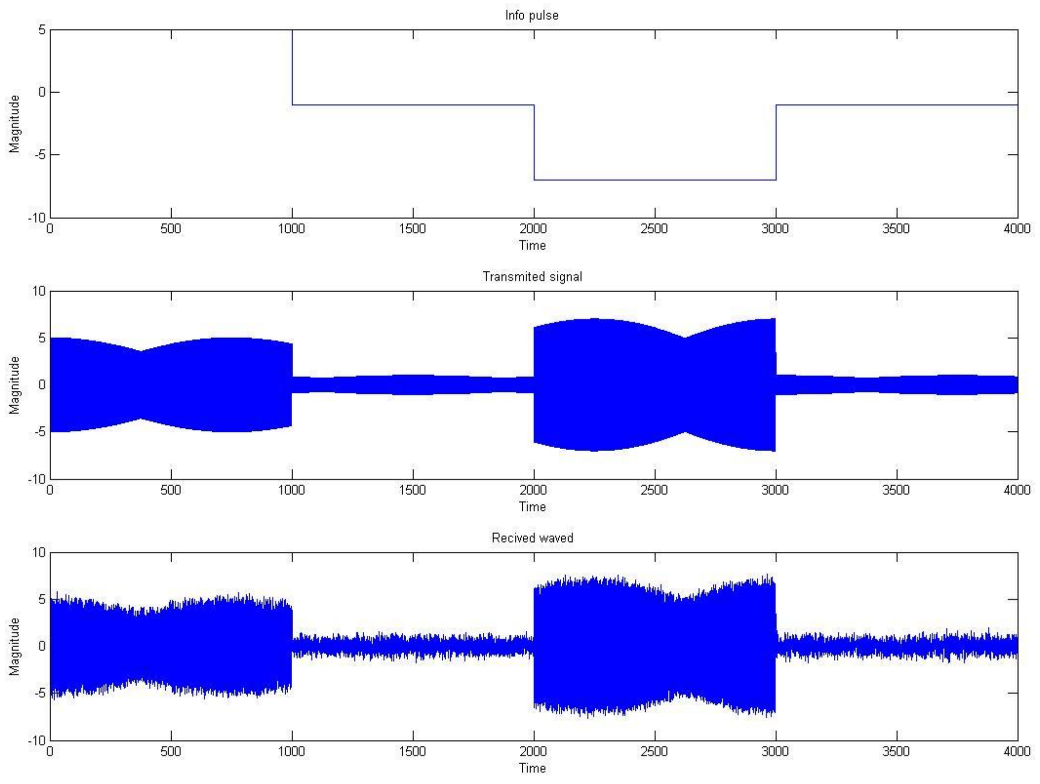


Figure 15 Info pulse, Sampled TX signal, Sampled RX signal

First, the code will generate a binary sequence of zeros and ones, simulating the information sent across the radio. That sequence will be sent 3 bits at a time using 8pam modulation. For more detailed information on the modulation technique being used, please refer to the section: Design Details.

The top graph in figure 25 shows the information sent in rectangular wave form, where the width of the pulse is 1000 symbols long. In other words each value will repeat 1000 before moving to the next one.

The Middle graph in figure 25 represents the transmitted signal. This signal is achieved by multiplying the square signal with the carrier wave. The carrier wave is a cosine wave with frequency of:

f = fc -BW/2 + channel

Where fc is the center frequency of the Carrier wave, in our case 75MHz.
BW is the band width of the signal. In our case BW = 200 kHz, and channel is the specific channel frequency the receiver is listening to. The Channels are multiples of 8.33 kHz or 25 kHz bounded within the bandwidth. This way the transmitted signal will be centered at 75 MHz, varying to +/- 100 kHz and the exact distance from the 75 MHz will be determine by the channel frequency. The signal is represented in a discrete form and not continues. The sampling rate at which it is displayed is fs = 300 MHz. This is the reason why the wave is in a bowtie shape. The number of points displayed per period is fs/f. Since this is not always an integer value there will be an unequal number of samples from period to period, causing the amplitude of the wave to be displayed screwed.

The final wave is a simulation of the signal seen at the receiver end. The wave here is much more random since it has noise added to it. The noise has a constant spectral density (has equal power in all frequencies) and a Gaussian distribution of amplitude with sigma = 0.1 and on offset (mean) of 0, in other words the value of the noise follows a bell curve.

2. Channel Selection

The filters designed are FIR or Finite impulse response filters. The FPGA implements it by delaying the input signal, multiplying it by a tap value and summing all the results. For further information on how an FIR filter operates refer to Design Approach section.

The band pass filters were designed by an Equiripple technique with density factor of 20. In comparison to Windowing techniques this method will optimize the order of the filter. This is possible by employing Ramez exchange algorithm to approximate the impulse response of an ideal filter.

The filters will operate on a sampling frequency of 800 kHz
All the filters will have an approximate attenuation of 65 dB in the stop band region and a linear phase in the band pass region

The filters were designed to minimize the order of the filter for the given specifications which for the 25 kHz channel spacing, the number of taps were 242 and for the 8.33 kHz spacing, the number of taps were 369.

Each filter frequency and time response is listed in Appendix III: ImpulsesandFrequencyResponse.docx. The code and output files can be accessed from FilterDesign.zip and the Filter taps are listed in Appendix IV ChannelFilterTaps.xlsx

3. Low Pass Filtering

There were two low pass filters designed. Both of them use the Kaiser Window method. For information on how the Keiser window method operates refer to the Design Approach section. The first filter will takes the signal after demodulation to remove the high frequency component; the signal is then down sampled and passed through the second low pass filter which will ensure that the band pass is limited to 200kHz. The reason why those filters are separate is because at high sampling rates it will take around 8000 taps to implement the same functionality. This way we optimize the filtering capacity without exhausting the memory of the FPGA.

The first low pass filter is needed to preselect the input wave. This filter will operate on the same sampling frequency as the external ADC which is 300MHz. The filter will mainly pass frequency below 600 kHz and it will have a transition band until 5 MHz See Appendix III: Impulse and Frequency Response for frequency response. This filter may does not have a sharp transition as the channel selection filters because it needs to filter frequencies much farther apart. The filter is targeting to filter the high frequency component result from demodulation which is center at 150 MHz and therefore fast transition band is not needed. The filter has an order of 271 and a stop band suppression of -90dB. See Appendix III for Frequency response graph.

The second low pass filter will ensure that the frequency will be within the Bandwidth specified. The filter operates on a sampling rate of 800 KHz, has an order of 81 with pass frequency of 200 kHz and stop frequency of 250 kHz and a stop band suppression of an additional -90dB. This filter has much sharper transition band of 50 kHz, in order to eliminate any interfering frequency close to our transmitted signal.

4. User Interface

The user interface is very simple. As it is made specifically for the development test board, the program may have to be changed when actually placed into production.

The first control uses four of the on-board switches. The switches allow the user to select the channel they wish to listen to. Channel selection on the switches is done in binary. For example, if the user changes the switches to off, on, on, off, then the user would be listening to channel 6. The program also displays the current channel using the seven-segment display located on the board itself.

5. Website

A website was created to publish information regarding our project to the public. Currently, the website can be found at the following link:

http://www.cefns.nau.edu/Research/D4P/EGR486/EE/10-Projects/Wulfsberg/

The website contains general information about our project, including design, short biographies of the team members, and information regarding the undergraduate symposium.

Website Development

The website itself was developed by using a combination of hand-coding and Adobe Dreamweaver CS4 & CS5 beta. The general layout of the website is based on the layout of the College of Engineering, Forestry, and Natural Sciences' website (http://cefns.nau.edu). The color scheme was changed to a darker color of black and gold.

Once the layout was created, it was converted into a template using the template feature in Dreamweaver. Then each page was created based on that template, and the appropriate information was added to the page.

Undergraduate Symposium

Our team also participated in CEFNS' Undergraduate Symposium on Friday, April 23rd, 2010. During the symposium, our team presented our topic to a group of professors and fellow peers at 10:00am. The presentation lasted about 30 minutes, and was considered a huge success by us. Later that day, our team also held a poster session, in which we presented a poster containing information about our topic. Both the presentation and the poster are attached to this document in Appendix 7 and 8.

Problems Solutions

Memory limitations

In retrospect, there are several ways to avoid running out of memory on the FPGA. First, one could simply use an FPGA with more on chip memory, or use an external memory to store the filter taps. However, this will drive the cost of the product higher.

Another solution would be to use Infinite Impulse Response filters. IIR filters require less taps than FIR filters, however they do not generally have linear phase in the pass band. This causes problems for the modulation chosen, since the modulation is dependent on the phase. For example, a symbol equal to 5 has the same magnitude as a symbol equal to -5 when modulated. The only difference is a phase change of 180 degrees. There are two ways to approach this new problem, first a modulation can be chosen such that the phase will be irrelevant, such as amplitude modulation sent with the carrier frequency. The current 8-pam modulations being used can be modified to include only positive values by adding an offset. This way the values transmitted will be 1, 3, 5, 7, 9, 11, 13, and 15. This way the phase of the signal will be irrelevant when the symbol is demodulated. If the modulation cannot be changed, there is a second approach, correcting for the phase. Since it easy to determine the phase of a digital FIR filter (phase is plotted in Appendix I) an all pass filter can be design to approximate linear phase. An all pass filter has the same number of poles as zeros placed in reciprocal manner around the unit circle. An example of one pole one zero is shown below in Figure 26. The x indicates a pole and o indicates a zero.

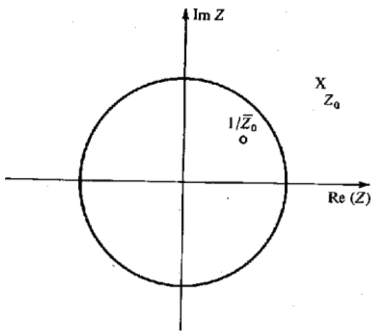


Figure 16 Pole Zero Diagram

This filter will pass all frequencies equally, because the pole and zero cancel each other out, however it will change the phase of the signal passed.

A third option is to calculate the filter taps when the channel is selected. The channel selection filters have the same amount of taps within the same channel spacing. The filters have the same number of poles and number of zeros, and the shape the poles and zeros take stays the same, the filters differ from each other by rotating the poles/zeros around the origin. In such a case, the FPGA will need to store only two filter taps, one for the 25 kHz channel spacing and one for the 8.33 kHz channel spacing, those taps will be used as a base to calculate the taps for the filter desired. Although mathematically sound, this method may have some practical issues. The FPGA will need to allocate sufficient resources to calculate the filter taps, each time a channel is changed. These transition calculations will have to happen very fast and errors may occur depending on the stress operation of the FPGA. Furthermore, storing all the taps on memory allows for more dynamic overall process.

6. Testing

a. Transmitter testing

The transmission simulations created a waveform designed to simulate a signal sent into the digital portion of our project. The first step was to create a Matlab file that creates values representing a sinusoidal wave. This wave is a representation of the wave that would be received from an external circuit. The values are then converted into binary format and then saved into a text file. This file is then transmitted into our receiver.

b. Receiver Testing

We wrote Verilog code that reads the text file created in the transmission section. We named this code Convert.v. The next code written is named Demod.v, which performs down conversion, low pass filtering, sampling, channel selection and wave reconstruction. Demod.v is integrated to receive the text file from Convert.v.

c. Down conversion

We managed to write the down conversion code, but were not able to test it, as we encountered some coding problems with the test bench we wrote. Therefore, this portion does not have any simulated results.

d. Low pass filtering

We were unable to fully test the low pass filter, as the DE2 board we used did not have enough memory to perform the necessary operations. This portion therefore does not have any simulated results.

e. Sampling

Sampling involves taking a stream of binary data, converting it into 4 bit binary digits (3 bits data and one bit a signed bit) and then using the 8 – PAM (Pulse Amplitude Modulation) method to convert the data from values ranging from 0 to 7 into values of -7, -5, -3, -1, 1, 3, 5, 7. The data below reflects a test performed.

<u>Stream of data input</u>

101111000001010000111110101011011101;

<u>Data separated into 4 bit binary</u>

101 111 000 001 010 000 111 110 101 011 011 100;

<u>Results of data after 8 - PAM symbol conversion</u>

101 → 5 = 3
111 → 7 = 7
000 → 0 = -7
001 → 1 = -5
010 → 2 = -3
000 → 0 = -7
111 → 7 = 7
110 → 6 = 5
101 → 5 = 3
011 → 3 = -1
011 → 3 = -1
100 → 4 = 1

f. Wave reconstruction

We were not able to perform wave reconstruction as…

7. Codex Implementations

Unfortunately we did not got as far in the project in order to implement the codex.

IX. Budget

The budget for our project consisted of Component Budget, comprising of components we needed, and Time Budget, comprising of the man power put into this project, with regards to time.

1. Components

We initially decided to design the analog circuitry to prep the data that would be sent into the Digital part of our project, but after research we determined that the Analog section was not necessary, and was not in the scope of the requirements handed to us by our client. Due to this decision, we did not need to purchase any components, as the necessary components needed to implement our project were made available to us by Northern Arizona University.

2. Time Budget

For the time budget of our project, we split it into 4 quarters, spanning throughout Fall 2009 and Spring 2010.

For the **first quarter**, we established communication with our client and attained the requirements for this project. We spent the majority of this quarter on technical research, where we went over the general requirements to figure out how we would approach this project. We then did individual research to have a solid understanding of the project as a whole. **Figure x1** is an overview of activities performed throughout the first quarter.

Table 5 The first Quarter

1st Quarter

Name	Time spent
Client Communication	5 hours
Technical Research (overall)	45 hours
Brainstorming	14 hours
Documentation	20 hours
Total	**84 hours**

For the **second quarter**, we did some research to determine if we would need to purchase any components. We spent most of this quarter doing a more detailed technical research. We looked at methods to implement our ideas into the project. We also ran systems level simulations of the transmitter section of our project. **Figure x2** is an overview of activities performed throughout the second quarter.

Table 6 The Second Quarter

2nd Quarter

Name	Time spent
Parts Research	5 hours
Technical Research (detailed)	30 hours
Systems level Simulations	15 hours
Documentation	25 hours
Total	**75 hours**

For the **third quarter**, we implemented our design plans. We spent a substantial amount of time writing up the code for the digital section of our project in verilog form. We started with the filter designs and updated transmission simulations for our project. We then performed some research for ideas on designs as well as necessary content for the website. **. Figure x3** is an overview of activities performed throughout the second quarter.

Table 7 The third Quarter

3rd Quarter

Name	Time spent
Verilog code	120 hours
Filter Design	5 hours
Transmission simulation	48 hours
Documentation	30 hours
Website	80 hours
Total	**283 hours**

For the **fourth quarter**, we made adjustments to our filter design and ran simulations to ensure that the updated design was functional. We completed the team website by implementing the design ideas and inputting content. Testing was performed for the individual modules of our design. We prepared for the conference by making a poster entailing the specifics of our project and also making a presentation to verbally communicate the concept of our project to an audience comprising of technical and non – technical people. **Figure x4** is an overview of activities performed throughout the second quarter.

Table 8 The fourth Quarter

4th Quarter

Name	Time spent
Filter Design	68 hours
Transmission Simulation	5 hours
Website	60 hours
Testing	150 hours
Documentation	78 hours
Conference Preparation	32 hours
Total	**393 hours**

Throughout the execution of this project, we kept up with documentation, writing requirements reports, client status reports, team memos, team agendas, and team biweekly reports. This was for the purpose of staying on schedule with the project and keeping our professor as well as the client well informed about our progress of the project.

X. Research

Our team used a variety of sources for research. Most of our research was done using the internet, and can be found in Appendix X. We also used many professors here at NAU, including Dr. Paul Flikkema, Dr. Sheryl Howard, Dr. Elizabeth Brauer, Dr. Niranjan Venkatraman, and Dr. Phil Mlsna

Meeting with Professors:

Dr. Flikkema

Dr. Flikkema's specialty is in networked communication and computation systems. Dr. Flikkema helped lead us along the right path, including his suggestion that we complete a simulation in Matlab.

Dr. Howard

Dr. Howard's specialty is communication systems, and she proved to be a valuable resource in developing our digital filter.

Dr Brauer

Dr. Brauer teaches both of the FPGA classes here at NAU. She helped us convert our ideas and math we had on paper to a VHDL/Verilog programs.

Dr. Venkatraman

Dr. Venkatraman helped us with understanding the fundamentals about Automatic Gain Control (AGC), which we ultimately did not use in this project.

Dr. Mlsna

Dr. Mlsna's specialty is digital signal processing, and like Dr. Howard, was invaluable in the development of this project. He expanded on what we learned from Dr. Howard and gave us several different options at approaching our project.

XI. Conclusion

This team consisted of individuals who were fascinated by the challenge set forth by the project requirements. We took on this project with no experience of signal processing, but were confident that this task could be accomplished. Throughout the project, we implemented ideas that worked well towards completing this project successfully. We used an online tool (dropbox.com) to keep track of our documentation. This tool was very versatile in the sense that it worked as a shared folder. We also established more than one way of communicating with each other, which was vital in ensuring everyone on the team was kept up to date. We did a lot of research before starting the project, which in the long run saved us a lot of time. Meetings that were held were extremely productive, with meeting agendas used as a guideline for tasks that needed to be completed before team meetings. The team had some lessons to learn from this project, which will be greatly valuable to take into the working field, upon graduation. One of the things we did not account for was dropbox.com working against us, due to our inefficiency. Too many versions of a document were being uploaded into dropbox.com, making it almost

impossible to identify the final, edited version. The team had another dilemma, which almost disrupted the third quarter of our project. The members made a mistake of starting up multiple tasks all at once, and not being able to complete these tasks, since they were overwhelmed by the workload. The team managed to scramble and complete all the pending tasks, and asserted that only one goal should be addressed at a time.
We were also naïve when it came to scheduling, as we did not account for problems that could arise throughout the project. The team rectified this by making more reasonable deadlines, after analyzing all the possible delays that could occur.

1. Project critique

Despite the setbacks we faced, the team was able to face these challenges and overcome it. Although the team does not have a finished product, we will look at this project as a success, and take with us some valuable technical and non – technical experience.

List of Abbreviations and their definitions

Table 9 List of Abbreviations and their definitions

Abbreviation	Description
ADC	Analog to Digital convertor
AGC	Automatic Gain Control
ASIC	Application Specific Integrated Circuit
BPF	Band Pass Filter
DAC	Digital to Analog Convertor
dB	Decibel
DE2	Development & Education 2 (by Altera)
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
Hz	Hertz
IIR	Infinite Impulse Response
kHz	Kilo Hertz
LPF	Low Pass Filter
MHz	Mega Hertz
NAU	Northern Arizona University
PAM	Pulse Amplitude Modulation
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

XII. List of Appendices