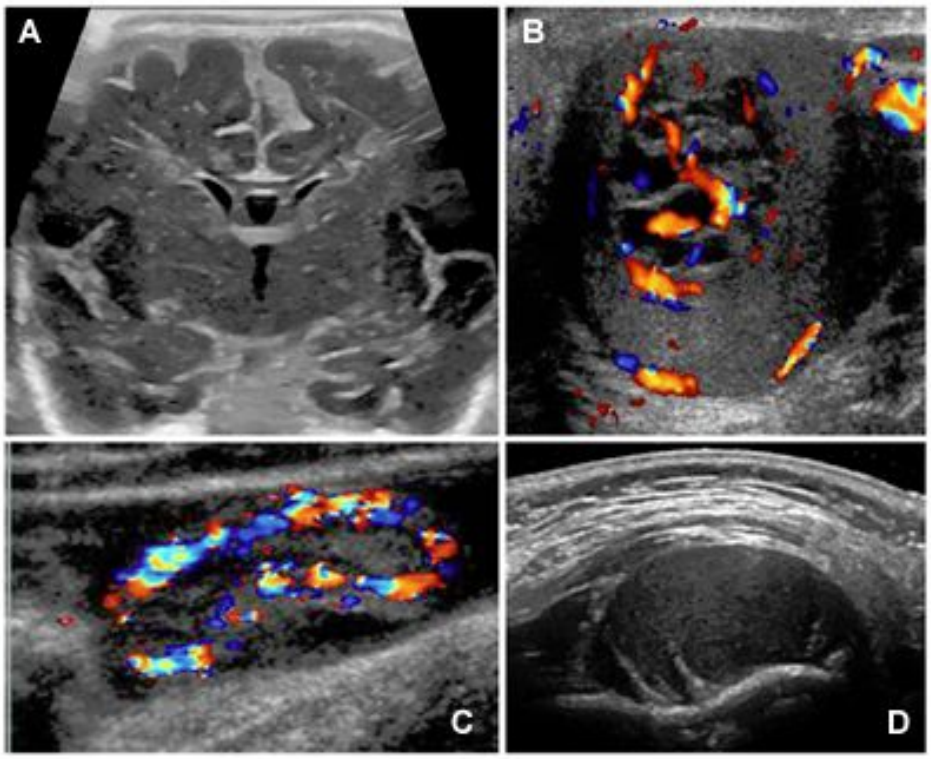
Brain Imaging Device, Drive Circuitry

Final Report



https://radiology.ucsf.edu/sites/radiology.ucsf.edu/files/import/wysiwyg/patientcare/sections/pediatric_radiology/

Team Number: DEC1619

Client/Advisor: Dr. Tim Bigelow

Team Members/Roles: Miguel Mondragon/Team Leader, Zechariah Pettit/Webmaster, Honghao 'Tom' Liu/Key Concept Holder

Team Email: DEC1619@iastate.edu

Team Website: http://dec1619.sd.ece.iastate.edu/

Table of Contents

1 Introduction

1.1 Project statement

This project is dedicated to continuing the development of the hardware of a pulse-echo ultrasound brain imaging device. The device uses a National Instruments computer system to create a serial input signal. This signal is then sent to the high voltage pulser which will use the input to create a series of signals with variable phase and duty cycle in a pulse width modulated signal form. These signals will then be converted to a sinewave, preserving the phase shift, while using the duty cycle to alter the voltage level. These signals will then be amplified and sent to a transducer to be turned into signals for imaging. These signals will then return after interacting the scanned object with altered phase shifts and energy levels which will then be reconverted into signals that can be used to create an image within the National Instruments system.

1.2 Purpose

The most prevalent form of brain imaging devices fall under the category of fMRI imaging which serves its purpose well but can be overly expensive to purchase and maintain. In addition the requirements for fMRI scanning can be make the service unusable for certain individuals. As an alternative the development of pulse-echo ultrasound systems serve as an alternative method of brain imaging with reduced costs and less limiting requirements but requires complex hardware and programming to use.

As such it can be said that our project sets out to create this complex hardware to serve as a less fiscally intensive alternative to fMRI brain imaging technology while making brain imaging services more available to patients with confining circumstances such as a mental health issues, phobias, and metal implants.

1.3 Goals

The goal of this particular senior design group is to develop and test the high voltage pulser of the device. This device will in other words serve as the source of the signals that will power the transducer of the device and thusly will be responsible for creating the electrical signals that will eventually be converted to ultrasonic waves.

2 System Requirements

2.1 Functional Requirements

- The ability to produce copies of a predetermined pulse width modulated signal.
    - Must be capable of altering the phase delay of these reproduced signals.
    - Must be capable of altering the pulse width of these reproduced signals.
    - Must be able to produce up to 8 output signals for this project but should be scalable for up 512 channels.
- Must convert these pulse width modulated signals into semi-accurate sinusoidal waveforms.
    - This method must also use the pulse width of the signals to alter the voltage level of these sinewaves.
- Filtering to prevent excessively high frequency signals exceeding 1.5 MHz.
- Voltage amplification up to 32 Vpp to at the 50% duty cycle.
- The design must be easily scalable up to the end design required 512 channels.

2.2 Non-Functional Requirements

- Circuit design made with noise reduction in mind.
- Cost efficiency examined when viable.
- Input and output connection methods researched and implemented.
- Considerations made for heat dissipation concerns.

2.3 Other Considerations

The requirements of the device underwent several shifts throughout the course of the project based on the interpretation of the intentions of our client in addition to occasional alterations to the initial plan for the development of this project.

3 System Overview

3.1 National Instruments System

The ‘brain’ of this system is the National Instruments PXI System which will not only serve to create the initial serial input that will be used as the beamforming input element of this device but also receive the return signals and use these signals to create an image. The PXI system boasts a modular design which allows the use of different National Instruments boards to fit the specific needs of the project.

The transmit side will utilize the NI PXI-7813R Virtex-II 3M Gate R Series Digital RIO Module to send the control signals towards the beamformer. This serial input will encode not only the base signal that is necessary to image the patient but also the variables and magnitude of difference in these variables that need to be altered in order to properly scan an object.

The receive side utilizes the NI 5752 Board will convert the receive side's analog signals into a digital readout which can then be utilized for signal processing to convert the signals into an image.

3.2 Beamformer

The beamformer in the traditional sense is the device that determines based on the purpose of the device what kind of signals need to be generated to perform it's function. In that sense the functionality of the beamformer is primarily carried out by the transmit side of the NI PXI system. However, the PXI system only determines what kind of signals need to be generated for the imaging of a patient. The actual creation of these signals falls under the purview of the Texas Instruments C2000 Launchpad which based upon a serial input will receive an initial signals and then generate up to 8 concurrent signals which have been altered slightly to fit the parameters of the scanning. It then outputs these signals as pulse width modulated signals.

3.3 High Voltage Pulser

After the beamforming process has been completed you will a series of digital signals with variable phase shifts and pulse width. Based upon these pulse widths the signals need to be converted into variable voltage sinewave signals while maintaining the phase shifts. These signals then need to be amplified by not only voltage but current to a lesser degree. These amplified sinewave signals will be used to excite the transducer and generate the ultrasonic waves that are used for the imaging process.

3.4 Transmit/Receive Switch

The transmit and receive switch serves to alternate the connection from the transmitted high voltage pulse waves to the receive side circuitry when the transducer is alternating from its transmit stage to its receive stage.

3.5 Transducer

The transducer serves as the part of the device that will directly interact with the patient. The device functions by receiving the high voltage pulses from the previous segment and has technology to convert these electrical signals into ultrasonic waves at a preselected frequency. These ultrasonic waves are sent into the body being imaged and are then reflected based on the the different materials the wave interact with inside of the body. These reflected signals return the transducer which captures them and uses them to create electrical system based

upon these waves. These new electrical signals are then sent to the receive side of the circuitry in order to be prepared for analog to digital conversion which can be used to create a proper image.

3.6 Receive Side Circuitry

The receive side circuitry has two primary tasks. Firstly it amplifies the received transducer signals into usable analog signals. Second it has a protection circuit in order to prevent signals with too large a magnitude from damaging the receive side NI PXI system. After both of these segments are passed the signal is then sent on to be converted into a digital signal.

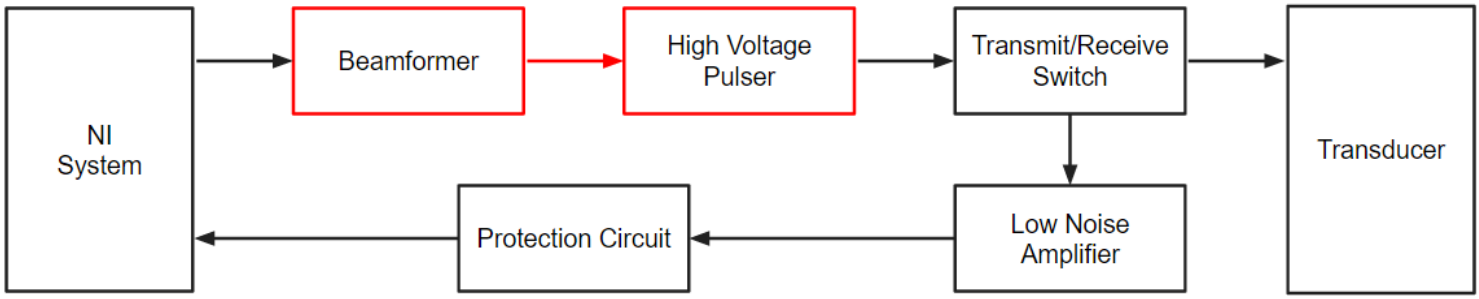


Figure 3-1: Block Diagram of Brain Imaging Device

4 Detailed System Design

This segment details the solutions developed by our senior design team. This includes the back segment of the beamformer and the whole of the high voltage pulser.

4.1 Texas Instrument C2000 LaunchPad (Beamformer Backend)

The Microcontroller on Texas Instrument C2000 LaunchPad will first generate a reference digital PWM signal based on the given frequency and duty cycle from I/O pins Digital(GPIO) transmitted through USB. To achieve desired PWM, the value in the active Counter-Compare Register is continuously compared to the time-base counter. When the values are equal, the counter-compare module generates a time-base counter equal to counter compare A event. This event is sent to the action-qualifier where it is qualified and converted it into more actions. Since the LaunchPad has 8 ePWM channels, 6KB on-board SAPAM,and 32KB flash onboard which is capable to store our short reference signal.

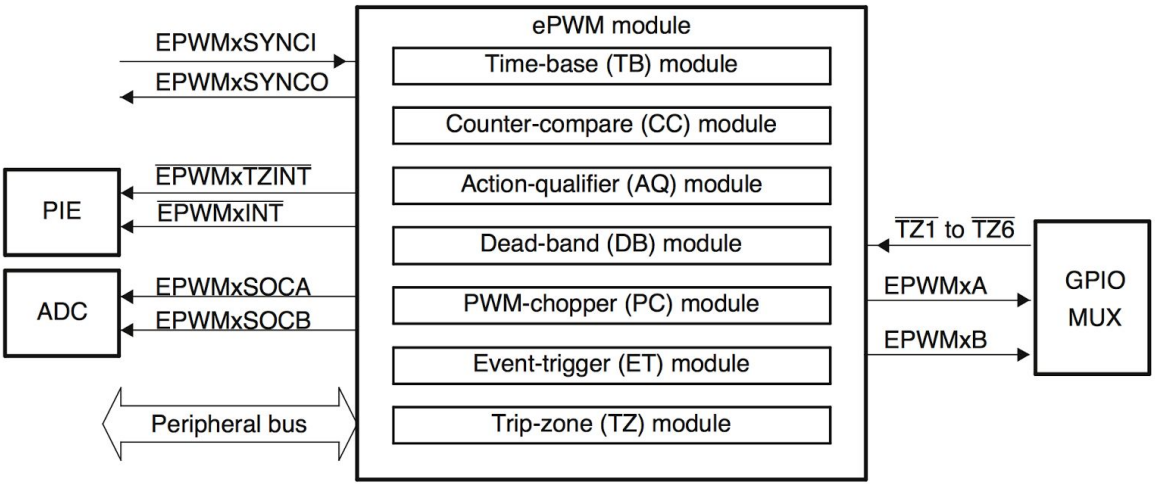


Figure 4-1: Submodules and Signal Connections for an ePWM Module

By accessing the stored reference signal, apply Time-Base Period Register (TBPRD) to duplicate given signal, microcontroller onboard can apply individual phase control to each duplicated PWMs by delay the PWM by using phase register (TBPHS). Our job is to programming appropriate values into TBPHS, each PWM modules can address another class of power topologies that rely on phase relationship between legs or stages for correct operation. For TI C2000 a PWM module can be configured to allow a `SyncIn` pulse to cause the TBPHS register to be loaded into the TBCTR register. As shown in Figure 4-2, a master and slave module with a phase relationship of 120°.

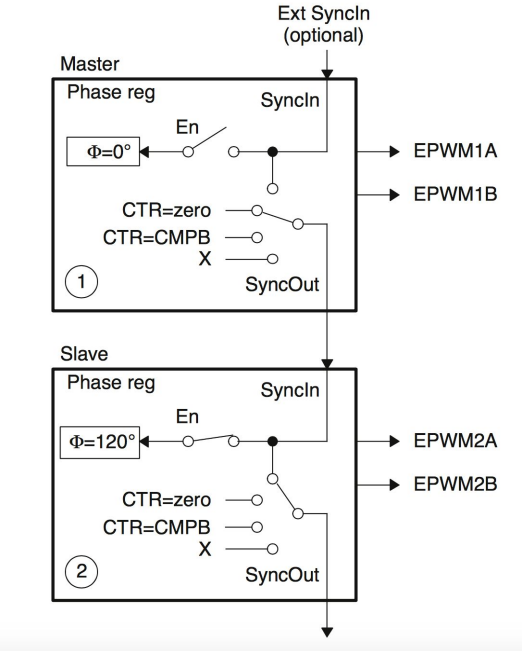


Figure 4-2:Configuring Two PWM Modules for Phase Control

For our tesing code, TBPRD = 600 for both master and slave. For the slave, TBPHS = 200 (i.e., 200/600 X 360° = 120°). Whenever the master generates a SyncIn pulse (CTR = PRD), the value of TBPHS = 200 is loaded into the slave TBCTR register so the slave time-base is always leading the master's time-base by 120° as shown in Figure 4-3.

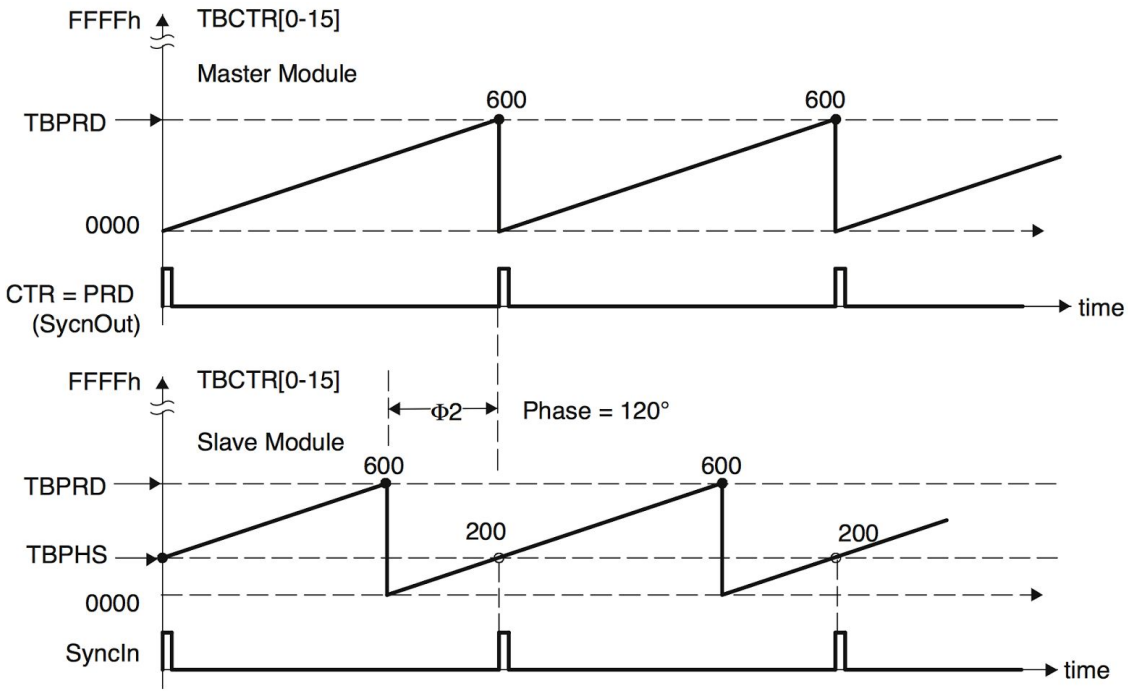


Figure 4-3:Timing Waveforms Associated With Phase Control Between 2 Modules

4.2 Digital to Analog Filter

	Expected Input	**Expected Output**
Voltage	1.75Vpp (with 1.75V DC Offset)	Variable (based on Duty Cyle), but 2Vpp at max.
Frequency	~1.5 MHz	Same as input.
Waveform Type	Pulse Width Modulated Square Wave (Variable Duty Cycle)	Sinewave
Phase Shift	Variable	Same as Input

The digital to analog filter is the front segment to our high voltage pulser. It serves two primary functions. The first is to convert the pulse width modulated signal we receive into a semi-accurate sinewave. The second is use the duty cycle (or pulse width) of this received signal as a control variable for voltage. This is alongside some small secondary goals including

the prevention of excessively high frequency signals and a small amount of voltage amplification in order to assist the amplifiers further down the schematic.

This design was realized through a series of active lowpass filters. These filter used a standard butterworth format with a Sallen Key design with the intention to obtain the fastest possible voltage settling time (to prevent damaging voltage spikes) while maintaining a monetarily feasible number of stages. The end result was a three stage set of lowpass filters utilize six (three active and three passive) low pass filters. This design allows for the duty cycle voltage control mentioned in the previous segment while providing a near perfect sinusoidal output waveform. However, as initial testing revealed that there was a somewhat large loss of voltage even when the duty cycle was set to 50% and as such a small gain of 1.5 V/V needed to implemented so that at the 50% duty cycle mark the output had the initially anticipated 2Vpp to be sent forward onto the amplifier segment. This gain also serves to counteract a side effect of setting the -3dB frequency at the expected 1.5 MHz signal. This was done so that signals exceeding this frequency would undergo major voltage reduction but meant that the expected signal also received a voltage reduction. The gain of 1.5 V/V was set to prevent this while maintaining the benefits of the -3dB frequency setting. The AD826 OpAmp was selected as the key component of this circuit for its high slew rate, settling time, and unlimited capacitive load. In addition this opamp was necessary as it has a high bandwidth to allow the small gain of 1.5 V/V to be applied and when settling time is considered allow for the opamp to survive potential voltage spikes.

	AD826, OpAmp
Channels	Dual Channel
Slew Rate	350 V/us
Settling Time	70ns
Unity Gain Bandwidth	50 MHz
Max Capacitive Load	Unlimited
Power Supply	Up to +/- 18V

Following these lowpass filter we also placed a single passive highpass filter to serve as a prevention tool for DC offset signals. Whenever you filter a square wave that does not have a 50% duty cycle you receive a resultant offset depending on whether the duty cycle is greater than or less than 50%. By placing a passive highpass filter in front of this you remove the DC offset as it essentially has no frequency to let it pass. The only necessary qualification of this

filter was that it was capable of receiving the signal and did not have an exceptionally high cutoff frequency.

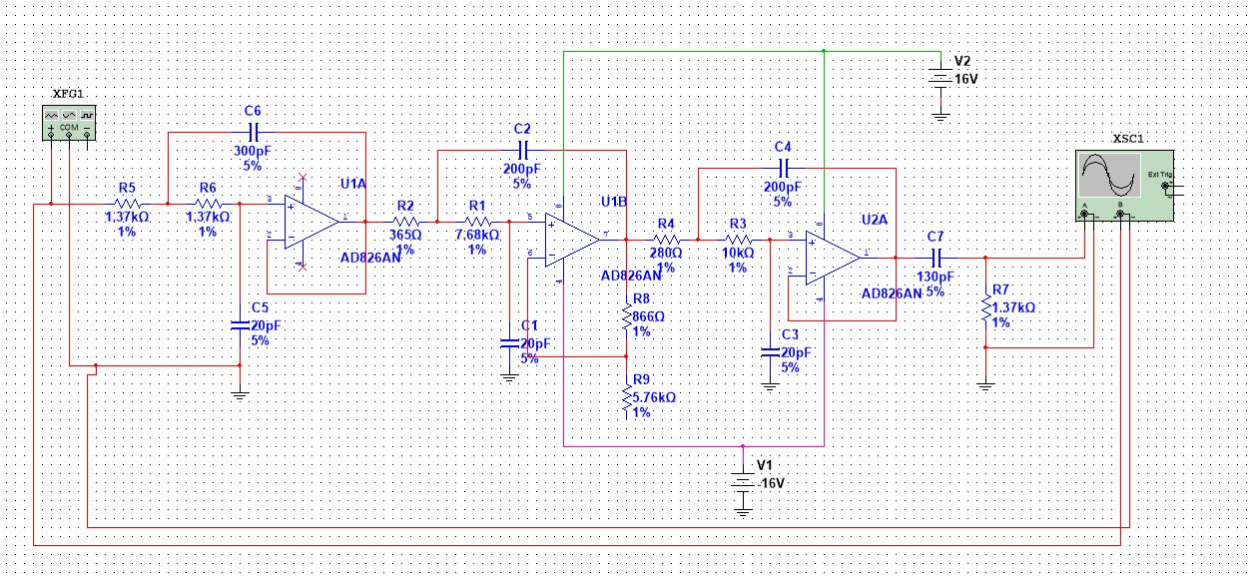


Figure 4-3: Simulation schematic for final ADC Filter

4.3 Voltage Amplifier

The signals produced by the launchpad and filtered by the Sallen Key design must have a power gain if they are to be sent and received by a transducer. The first part of power amplification is performed by a high speed operational amplifier. Many amplifiers were originally considered for this task, but no amplifier met the project's functional goals like the THS3001 op-amp. The THS3001 features a 6500-V/μs slew rate and power input of ±16 V. These aspects allow the filtered 1.5MHz signal to be amplified to a maximum value of 32V without suffering from distortion.

The amplifier is used in a simple single channel non-inverting configuration. This design meets functional requirements while simultaneously staying low on cost, as only two resistors are needed to set the gain. As seen in figure 4-#.

THS3001	
420-MHz Bandwidth	THD = –96 dBc at f = 1 MHz
– 0.02° Differential Phase	6500-V/μs Slew Rate
Output Current = 100 mA	VCC = ±16 V

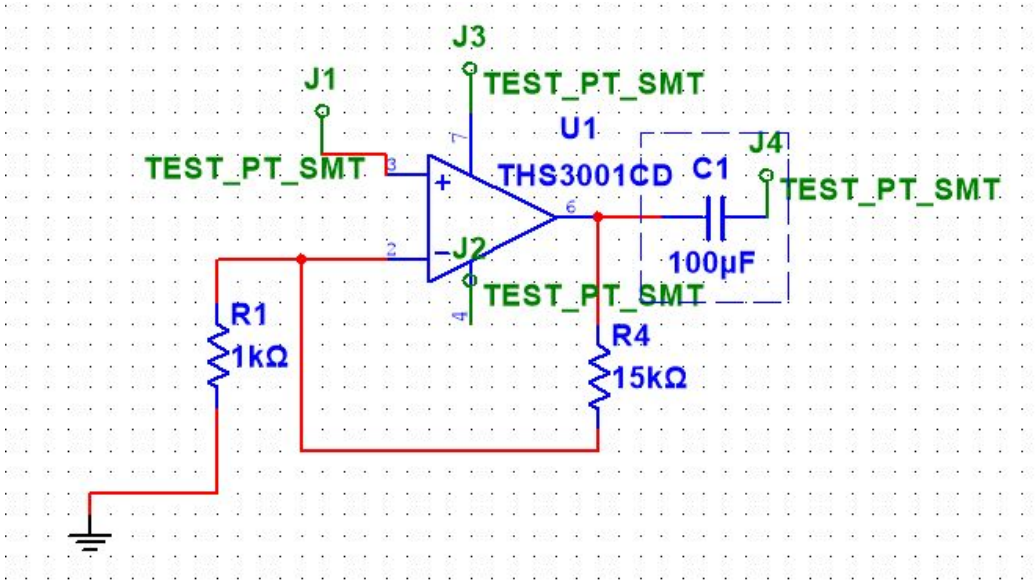


Figure 4-4 Voltage amplifier

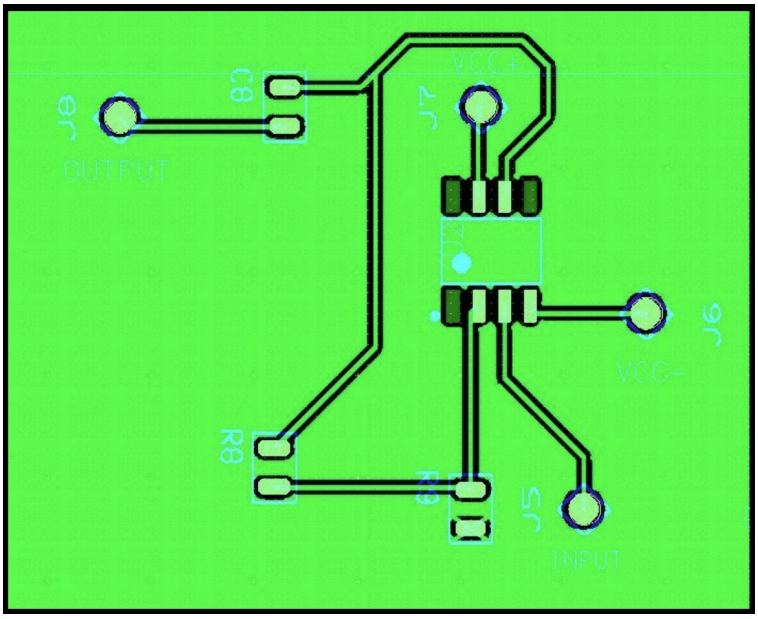


Figure 4-5 First Voltage amplifier PCB design

4.4 Current Amplifier

A current amplifier is also required to increase the power of our brain imaging signals. Without the current amplifier, the signals may reach a voltage level around 30V, however the maximum current output from the THS3001 is only 100mA. Without an additional amplifier the power output of the signals would be an approximate 3W. To solve this issue, two BJTs are used in a common collector configuration. This allows for high-speed signals such as the 1.5MHz to be amplified with little to no distortion. The push pull configuration is similar to that of a class AB

amplifier, allowing the NPN BJT to provide current to the positive half of our wave, while the PNP BJT pulls the current, amplifying the negative half. Working together they provide a power gain of approximately 5W/W.

MJL3281A (NPN) and MJL1302A (PNP)	
NPN/PNP Gain Matching within 10% from 50 mA to 5 A	High frequency for high amplifier bandwidth
Exceptional Safe Operating Area for reliable performance at higher powers	Excellent Gain Linearity for accurate reproduction of input signal

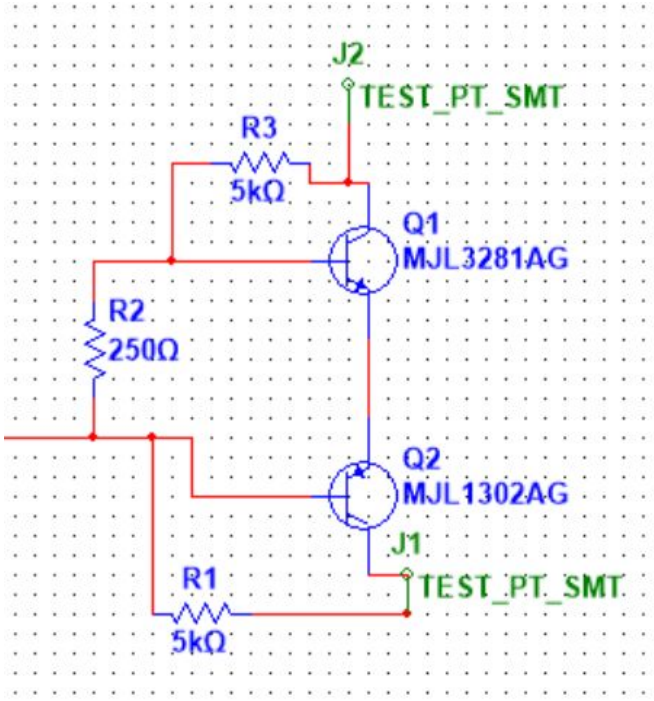


Figure 4-6 Current Amplifier

4.5 Full PCB Design

Below in Figure 4-7 is the final version of our 8 channel PCB. The design was made with consideration made for a proper grounding plane, space for heat dissipation, proper ports for source voltage input, and conversation of space where possible. Due to time constraints the board lacks some flourishes that would have made the design easier to parse for future work. However, the board passed several automated tests to search for design errors and is capable of being easily replicated for future production of multiple channels.

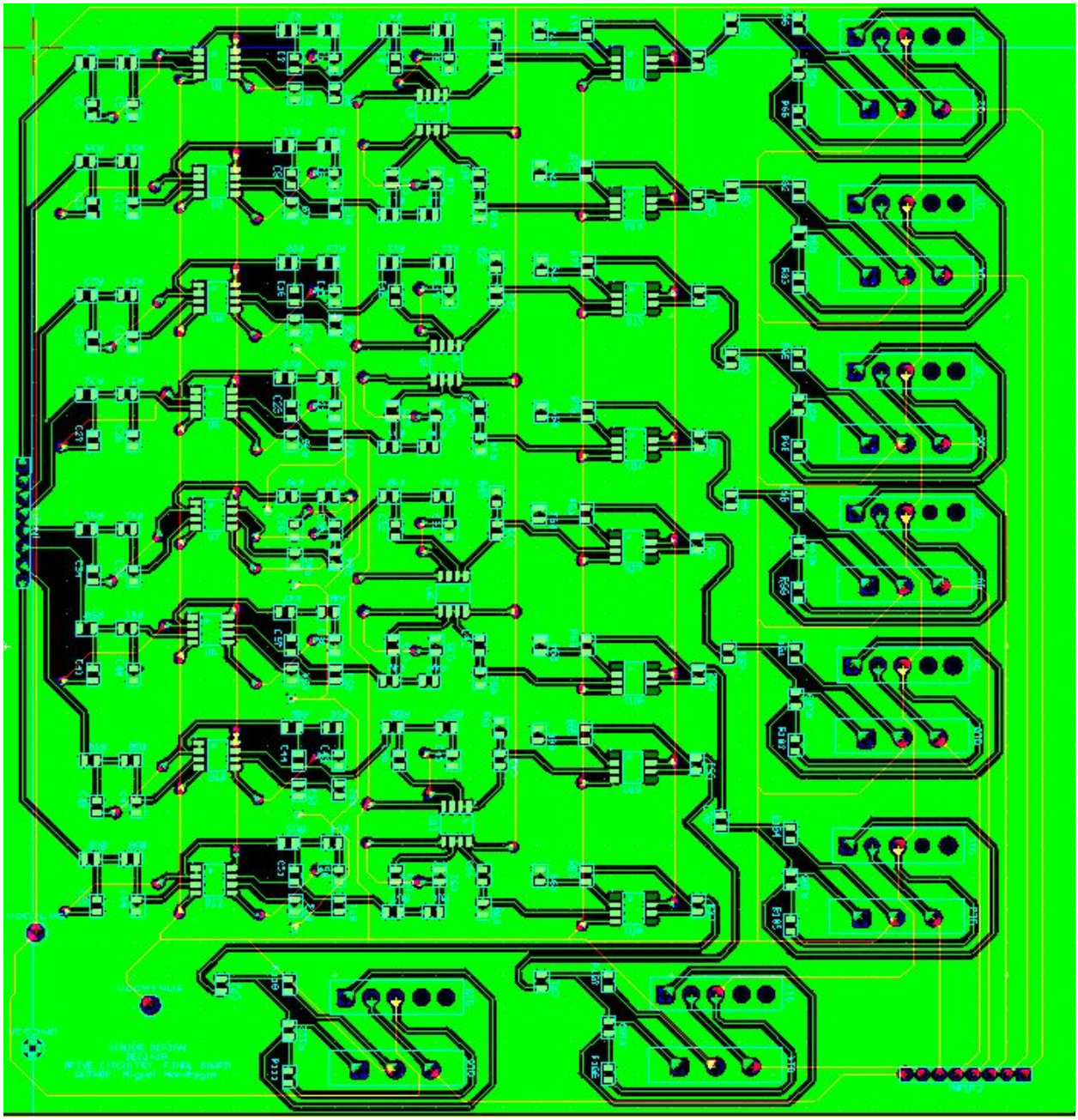


Figure 4-7: 8 Channel PCB

5 Challenges

5.1 Technical Challenges

One of the first and foremost technical challenges we experienced was the necessity to learn a number of technical skills that haven't been included within our engineering program. Learning individual software tools like Multisim and Ultiboard were a must.

In addition learning the basic concepts behind printed circuit board design was another difficult task as there has been little to no teaching in regards to this in our engineering program. As such the group had to devote a lot of time to learning the basic tips and tricks to creating a functional PCB.

From a design perspective there were a number of challenges that occurred as a result of the high frequency input that we needed to account for in our design. The number of available parts that were capable of the power amplification at the frequency we set out to meet were extremely limited and tracking down viable part selections constituted a major challenge.

5.2 Non-Technical Challenges

One of the primary challenges that was unrelated to the design aspect of our project was the necessity of soldering component to a printed circuit board. Experience between our group members in the 'art' of soldering was essentially zero and between this and the small surface mount components that were needed the group needed to spend a good chunk of time learning how to properly apply these components and solder them.

Ordering a small amount of specific components needs to be thought out and ordered months ahead of the planned "manufacturing" time. In addition due to the specific nature of these parts many times a part could be ordered for testing or plotted using simulations only to find that when the time came to purchase in bulk there were no longer any left to order. This resulted in situation where either substitutes needed to be found quickly or parts needed to be ordered well in advance.

6 Testing and Results

6.1 Testing Plan and Methodology

The main idea behind our testing methodology is outlined below, however this only applies to the design of physical components and does not pertain to the methods used when programming.

- Conceptualize initial designs.
- Simulate digitally with Multisim.

- Select and test complex parts of design.
- Create and test designs with physical PCB.
- When encountering design flaws, determine origin of failure then re-conceptualize based on the failure.
- Repeat Process.

6.2 Results

By compile the C code attached in Appendix III, first we were able to generate reference signal with various duty cycles which is achieved by using equal compare matches on the up count and down count portions of the waveform as shown in Figure 6-1. For our testing code, CMPA is used to make the comparison. When the counter is incrementing the CMPA match will pull the PWM output high. Likewise, when the counter is decrementing the compare match will pull the PWM signal low. When CMPA = 0, the PWM signal is low for the entire period giving the 0% duty waveform. When CMPA = TBPRD, the PWM signal is high achieving 100% duty.

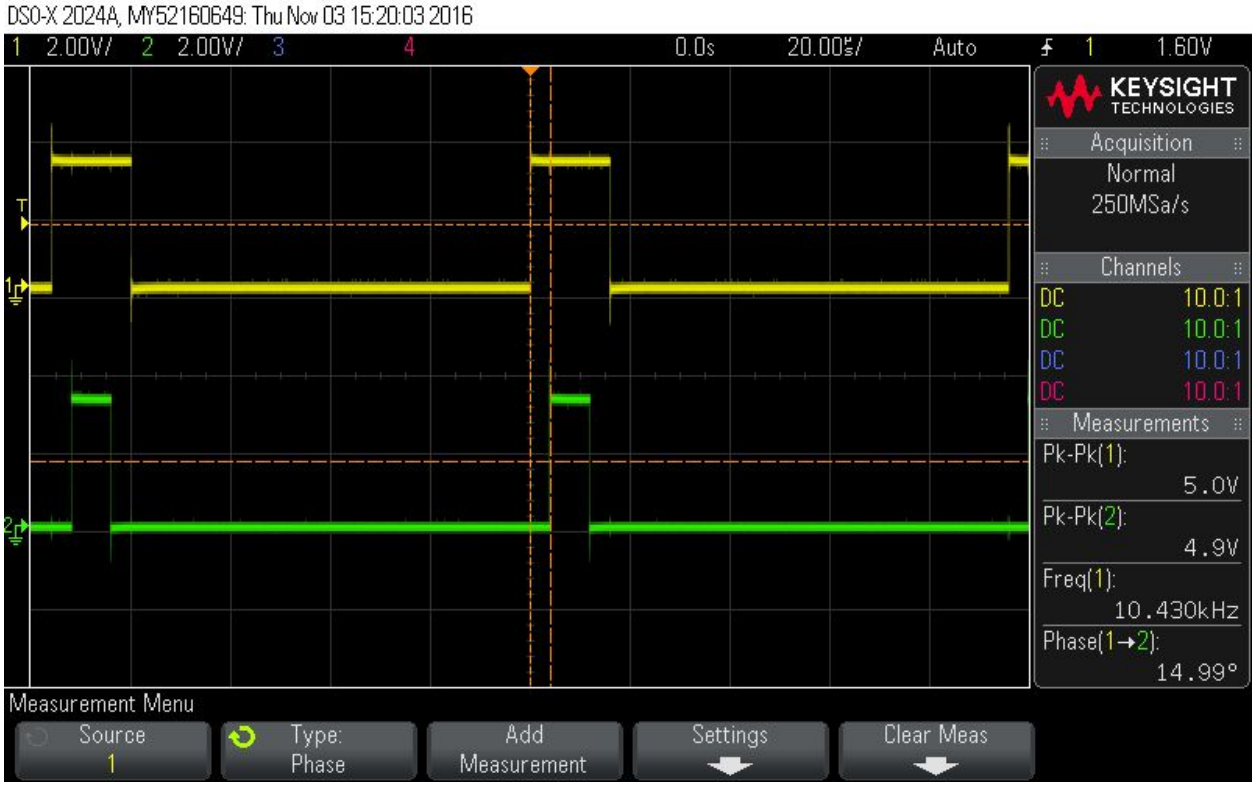


Figure 6-1: TI LaunchPad Output for 10% and 25% duty cycle

Now to add Lead or lag phase control to the waveforms generated by different ePWM modules to synchronize them. In up-down-count mode, the TBCTL bit configures the direction of the time-base counter immediately after a synchronization event. The new direction is independent

of the direction prior to the synchronization event. As shown in Figure 6-2, we were able to achieve precise 90° and 120° phase shift compared to the reference signal.

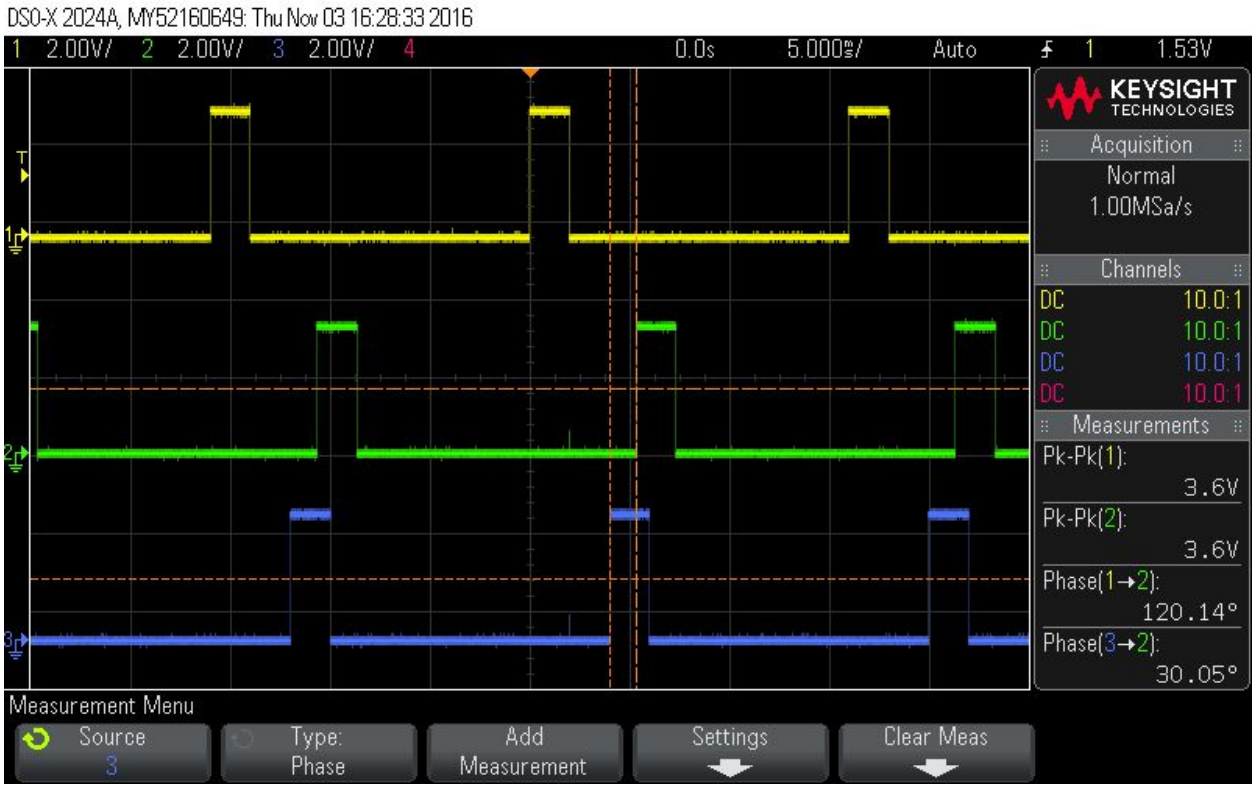


Figure 6-2: TI LaunchPad Output for applied phase shift.

The recorded results of the DAC filter are centered around the simulation results. The success of the simulation is based on two major factors. Firstly whether or not the circuit is able to successfully convert the circuit into a sinusoidal like waveform from the square wave that was received. Second whether or not the output voltage level is able to be controlled through use of the duty cycle. Additionally, when the input frequency exceeds 1.5 MHz does the circuit respond by curbing the voltage output. As you can see from the oscilloscope simulations below where the light blue line is the initial input and the red line is the output we were able to successfully perform all three of these functions within the context of the expected input within the simulation. The simulation itself details voltage level within the span of microseconds (1 microsecond per square) and measures voltage (1 V per square). As we increase the duty cycle you will see that the output voltage level increases. In addition Figure 6-5 details that when the 1.5 MHz frequency is exceeded the voltage level drops to near zero. In addition to you can use this graph to observe the fast settling time for this circuit which is a fraction of a microsecond. Additionally below is a table that details the voltage level of the output as the duty cycle is changed.

Duty Cycle	Output Voltage
5%	308.949mVpp
10%	606.489mVpp
30%	1.611Vpp
50%	1.987Vpp

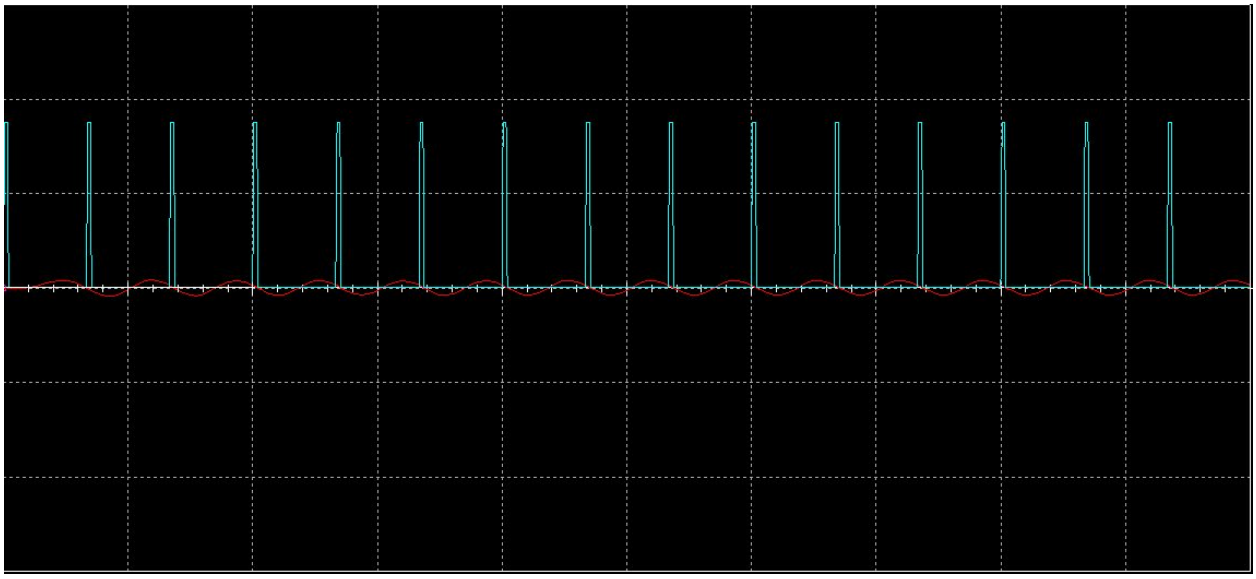


Figure 6-3:Voltage Simulation Results for 5% Duty Cycle at expected input.

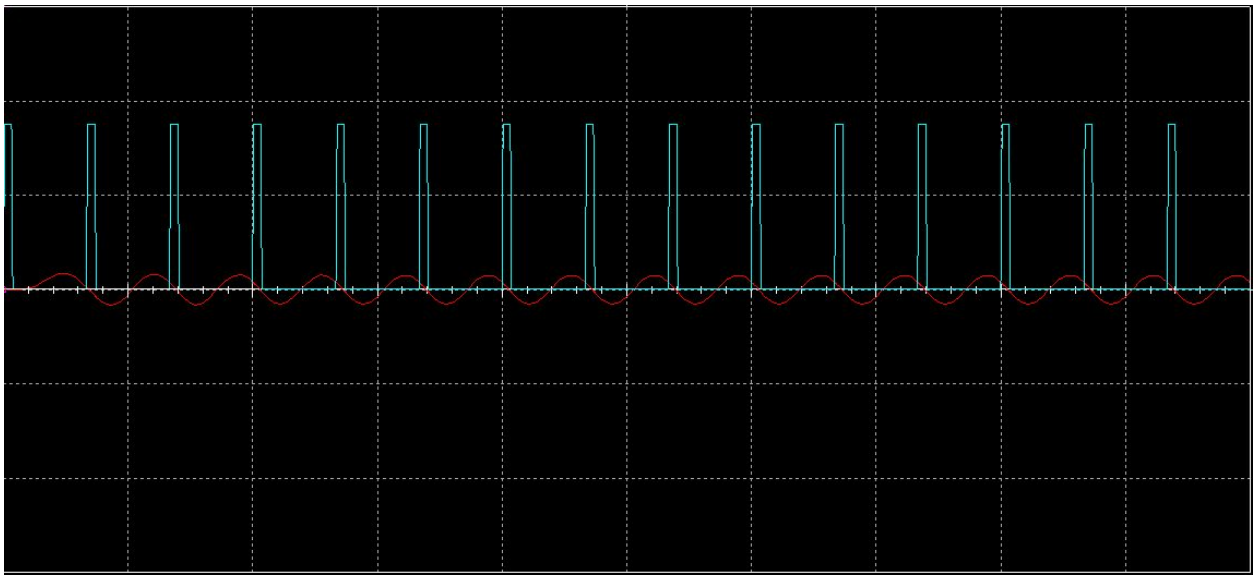


Figure 6-4:Voltage Simulation Result at 10% Duty Cycle at expected input.

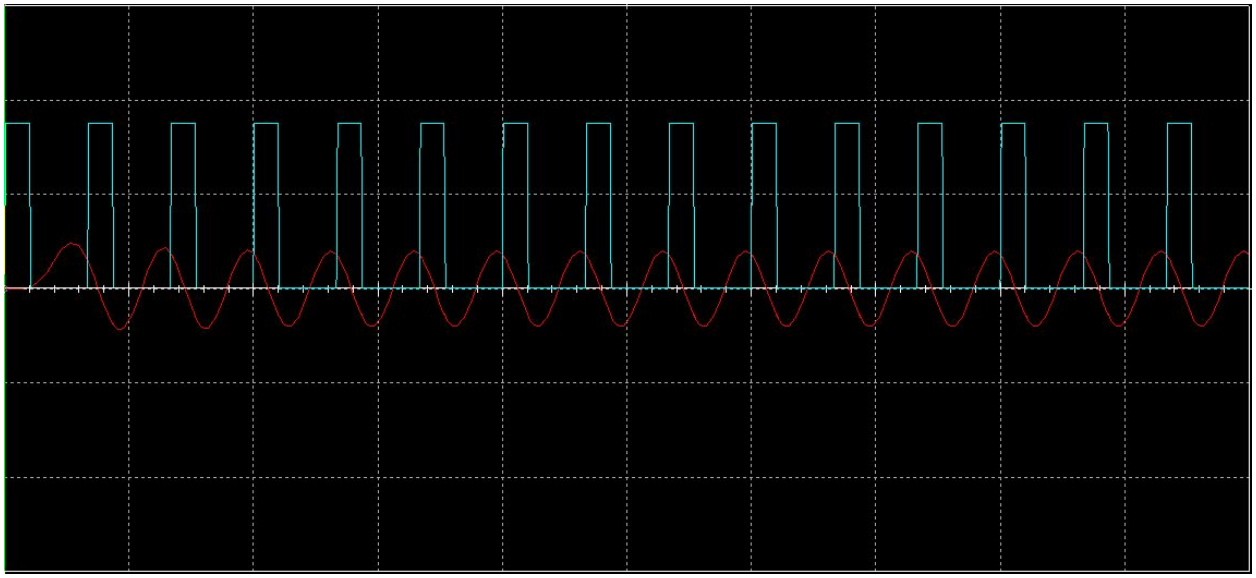


Figure 6- 5:Voltage Simulation Result at 30% Duty Cycle at expected input.

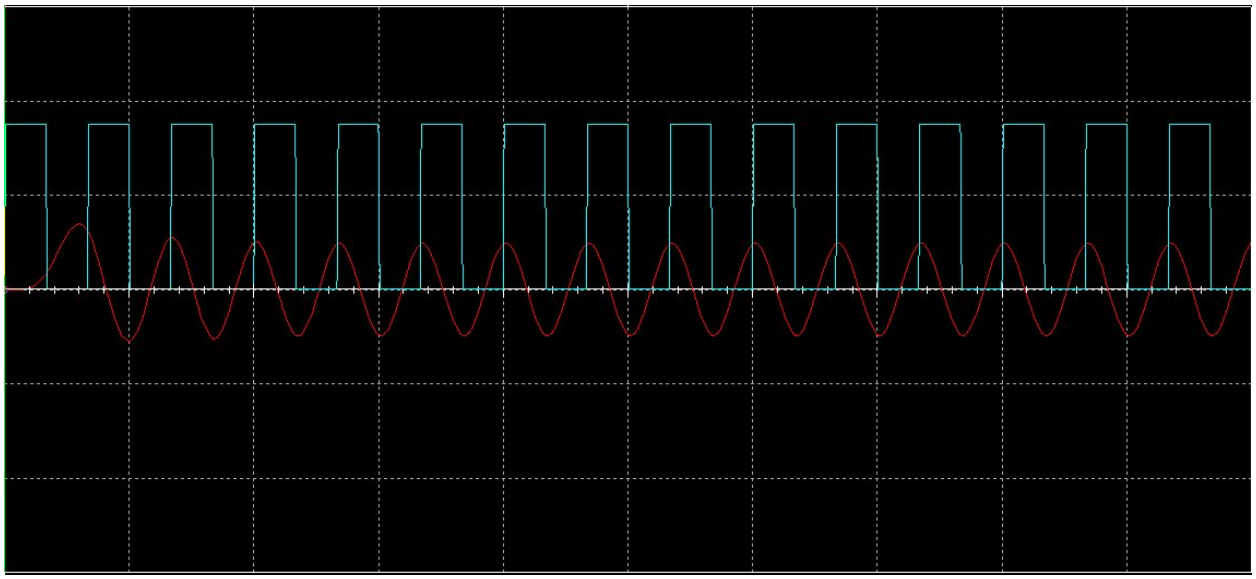


Figure 6-6: Voltage Simulation Result at 50% Duty Cycle at expected input.

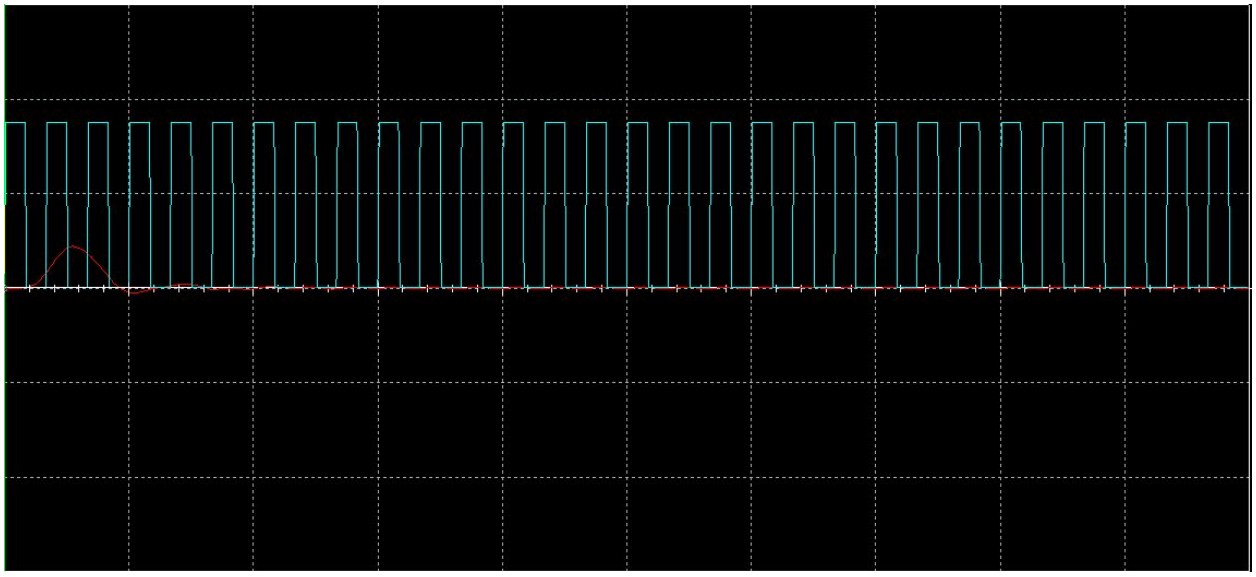


Figure 6-7: Voltage Simulation Result at excess frequency.

With these results in mind it can be said that the circuit succeeds in creating the desired output in the simulation. Due to time constraints it isn't possible to test this design as a PCB before the end of the senior design project however with the soldering of the final PCB it will be relatively easy to determine if the physical version of the board is capable of matching the simulation's results.

Each segment of the amplifier was first designed and tested in the CAD program Multisim. The current amplifier output driving the load can be seen below. The voltage amplifier was the first design to be perfected and was able to be apart of test PCB. The desired output of the physical voltage amplifier was obtained at a frequency of 1.5MHz can be seen in figure 6-8. The expected results were achieved in both simulation and physical testing of the amplifier.

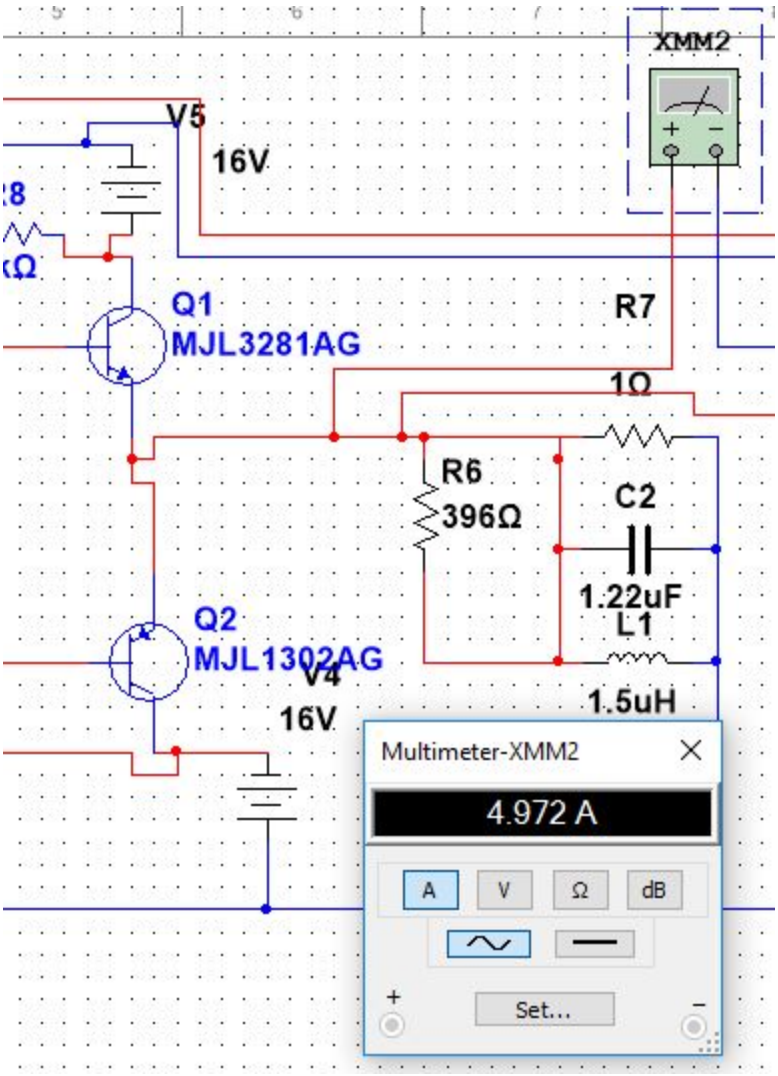


Figure 6-8 Current amplifier supplying power to load

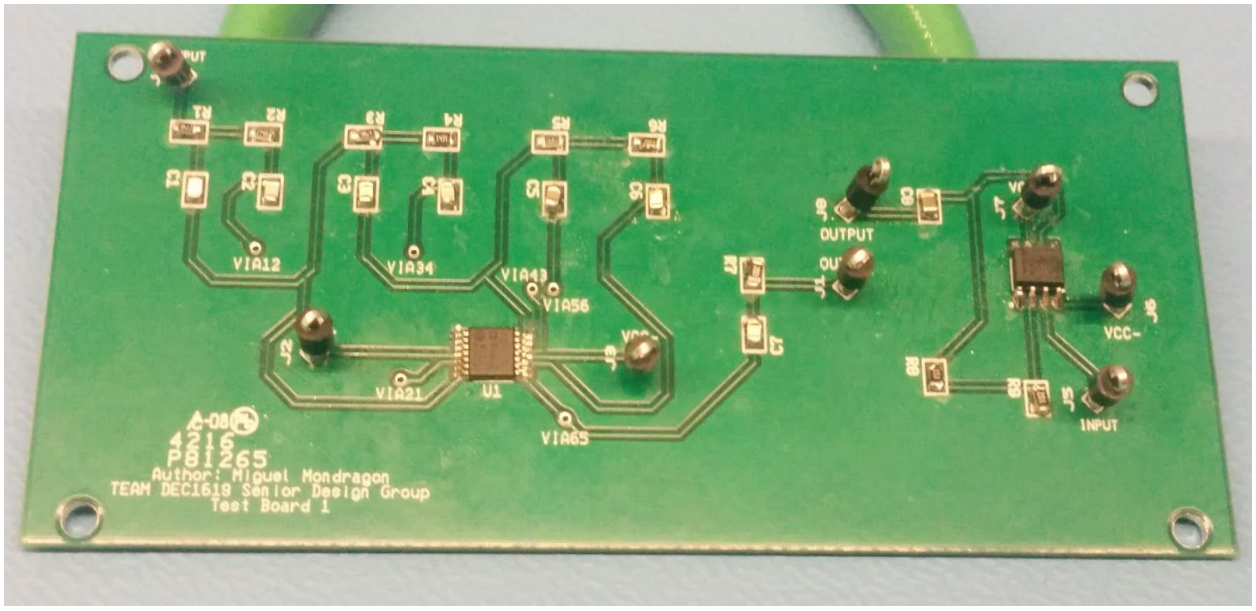


Figure 6-9 Sallen-Key Filter and Voltage amplifier PCB

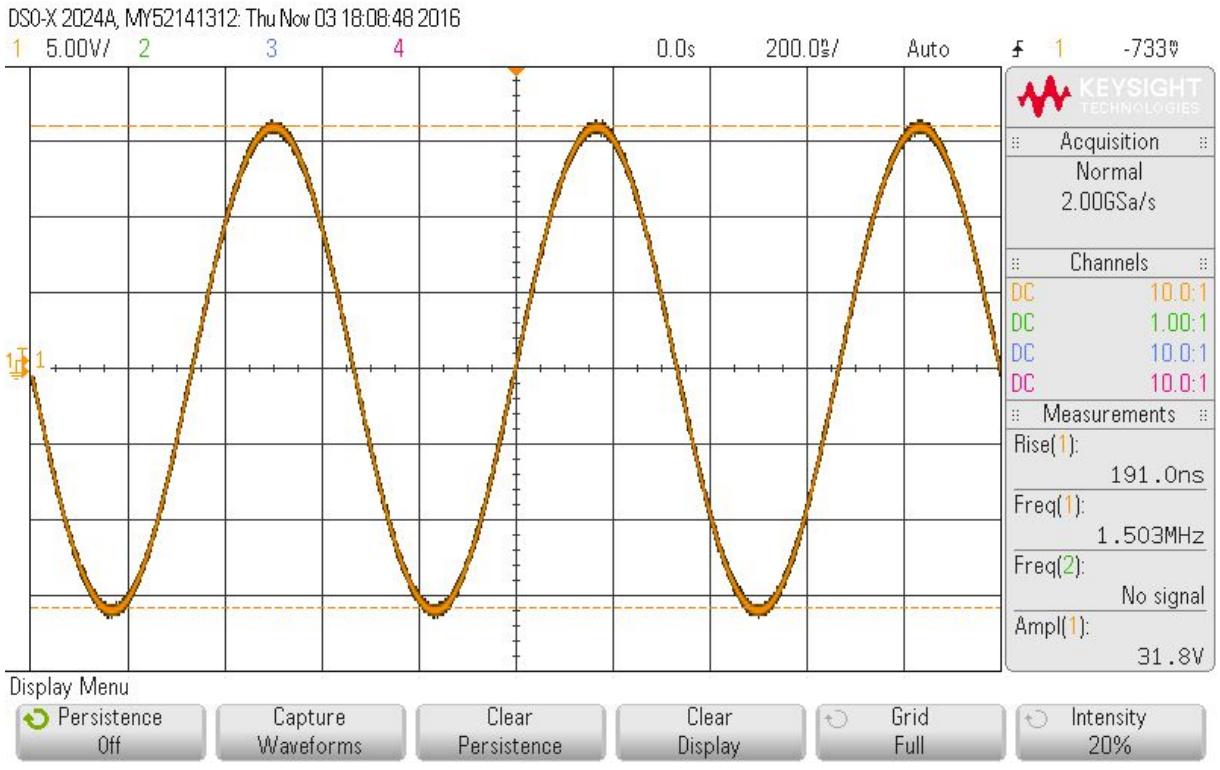


Figure 6-10 Voltage Amplifier Output when Input = 1.5 MHz 1Vpp

As section 4.5 details we have completed a PCB that contains the final design of the high voltage pulser with 8 channels. However, we have been unable to test this PCB due to a delay

in the ordering of the board resulting in the arrival of the PCB the day before the current draft of this document (12/6/2016). Currently the soldering of the board in underway and once the board has been soldered the testing of the device will commence. Upon completion of that testing this document will be updated to reflect the results.

7 Budget Information

7.1 Overview

Our group had an initial funding supply of 300$ to use for the purchase of parts and for the production of necessary hardware. Our initial expenditures were focused on the testing of physical components to determine if, when placed in mock designs, were capable of functioning under the constraints of our system. In addition as we progressed further into the project it became necessary to test single channel models of certain designs in order to further assess their viability as a solution to our challenges. However with the high cost of PCBs and Op Amps that could run in high frequencies it became difficult to avoid costs exceeding our budget. It should be noted that from our client's perspective exceeding budget was not a large issue as it was predicted that we would not have enough funds to both test components, designs, and a final product. As such with the final PCB and parts order we exceeded our budget by a rather large margin mostly when taking into account the high cost of BJTs and Op Amps that were capable of running at the necessary frequency. With these factors in mind and totalling the costs of the parts listed below as well as the two PCBs that were ordered (each at a cost of 50$) the end result was a total expenditure amount of 492.95$ a total of 192.95$ over budget. However, as previously stated our client considered this overdraft well within reasonable expectation for the production of hardware.

7.2 Budgetary Tables

Spring 2015 Parts Order :

Description	Digikey Part Number	Number Requested	Price Per Unit	Price
TI Launchpad	296-34797-ND	1	17.70	17.70
OP4584T OpAmp	296-23089-5-ND	4	15.01	60.04
ADA4891-3ARUZ OpAmp	ADA4891-3WARUZ-R7-ND	4	1.43	5.72
ADA4533 OpAmp	ADA4522-2ARZ-ND	4	4.31	17.24
			Total Cost: 100.70$	

Fall 2016 Parts Orders :

Description	Digikey Part Number	Number Requested	Price Per Unit	Price
274Ohm Resistor	311-274CRCT-ND	18	0.021	0.38
100pF Capacitor	PCF1316CT-ND	27	0.3384	9.14
1500pF Capacitor	PCF1298CT-ND	25	0.3908	9.77
78.7Ohm Resistor	P78.7CCT-ND	10	0.10	1.00
1.43kOhm Resistor	311-1.43KCRCT-ND	10	0.021	0.21
1000pF Capacitor	PCF1296CT-ND	18	0.3908	9.77
56.2Ohm Resistor	311-56.2CRCT-ND	10	0.021	0.21
2kOhm Resistor	311-2.00KCRCT-ND	18	0.021	0.38
Surface Mount Test Points	36-5006-ND	10	0.316	3.16
1kOhm Resistor	311-1.0KARCT-ND	10	0.018	0.18
15kOhm Resistor	RMCF0805JT15K0CT-ND	10	0.017	0.17
100uF Capacitor	587-3978-1-ND	10	1.538	15.38
THS3001CD OpAmp	296-1762-5-ND	9	10.41	93.69
			Total Cost: 143.44$	

Digi key Part Number	Number Requested	Price per unit	Price	Details
MJL1302AGOS-ND	8	3.50	28	PNP BJT
NJL3281DGOS-ND	8	4.59	36.72	NPN BJT
311-249CRCT-ND	10	0.021	0.21	249 Res
P4.99KCCT-ND	50	.022	1.10	4.99k Res
490-13311-1-ND	10	.165	1.65	.22uf cap
AD826ARZ-REELCT-ND	12	5.9480	71.38	AD826 OpAmp
311-1.37KCRCT-ND	25	.01520	.038	1.37k Res
P365CCT-ND	10	.1	1.00	365 Res
311-7.68KCRCT-ND	10	.021	0.21	7.68k Res
311-866CRCT-ND	10	.021	0.21	866 Res
311-5.76KCRCT-ND	10	.021	0.21	5.76k Res
311-280CRCT-ND	10	.021	0.21	280 Res
RMCF0805JT10K0CT-ND	10	.017	0.17	10k Res
490-10741-1-ND	16	.0.106	1.70	200p Cap
399-9220-1-ND	10	.25	2.50	300p Cap
1276-1829-1-ND	25	.03960	0.99	20p Cap
490-1602-1-ND	10	0.117	1.17	130p Cap
A31118-ND	2	.5	1.00	Ribbon Cable Pins
	TOTAL COST (before tax): 148.81$			

8 Future Work

8.1 Hardware Considerations

- Mounting points, eventually as the system with be part of several other pieces of hardware there will need to be the addition of mounting holes in order to properly secure the board to whatever other hardware is in the system. At the present moment there is no grand design for how these hardware pieces will fit together and in what sort of casing so the addition of any mounting holes was dubbed unnecessary.
- Heat dissipation, a major concern for our device is the product of heat from the high voltage pulser. As such the addition of a heat sink system or other cooling method would be a recommended addition in order to prevent a large system (such as the 512 channel system) from overheating and damaging system components.
- “True” BJT pair. The design incorporates the complementary set of BJTs : MJL3281A (NPN) MJL1302A (PNP). However the NPN was unavailable for purchase. Instead the NJL3281A was used. Theoretically the bases are the same, but realistically the manufacturing process for the MJL and NJL vary.

8.2 Software Considerations

- Software interface between the National Instruments HUB and the TI launchpad. This interface's primary goal is to create a user friendly environment to set values for the beamformer, such as Phase and Duty Cycle.
- Programming of National Instruments PXI Platform. This is both to create the serial input of the segment but also to perform the digital signal processing necessary to create the B-Mode imaging of the device.

9 Conclusion

The brain imaging system requires many components including the beamformer and amplifier. The beamformers requirements are to create and set the phase and duty cycle of a 1.5Mhz signal. This was accomplished by programming a TI launchpad, which has the additional capability of creating 8 separate signals. The output of the launchpad is a digital signal, read as pulse width modulated square wave.. The wave will then need to take an analog form and be amplified for the transducer. The Sallen-Key filter takes the 1.5MHz square wave and convert it into a sinewave using a 6th order butterworth filter method. It also applies a small gain of 1.5 V/V in order to help it reach the necessary output for later stages. Then the output of the filter reaches the THS3001 op-amp that has a set gain of 16V/V. This ensures that the signal at 5% duty cycle will output at 5V. With the maximum output reaching 32 Volts. The signal then travels through a pair of BJTs set up in a push pull configuration. This combination of transistors provides a current gain of 100A/A providing the load with approximately 5A. The design took many iterations of CAD reworks before reaching the first PCB board. Our PCB creation was a large learning experience which lead to a better designed 8 channel board. Within the context of the requirements we set out to achieve we can say that we've completed the beamforming element and that is satisfactory to the specifications of client while the high voltage pulser element remains under continued testing to determine functionality.

Appendix I, Operation Manual

TI Launchpad Setup

Code Composer Studio is TI's integrated development environment (IDE) for TI embedded processors which is required for this project. After import all attached code, PWM frequency, duty cycle and phase shift can be manually typed into the program. To compile and upload the C code, debug mode will be used. Output of the TI LauchPad will be on port J6-J14. All PWM output channels share the same ground.

High Voltage Pulser Setup

The setup for the remainder of the high voltage pulser signal is fairly straightforward. Consult the board to determine the locations of the positive, negative, and ground source banana plug jack. An input DC voltage of +/- 16V must be provided to the respective source voltage plugs and the ground should be plugged as well. The using a 2.54mm pitch ribbon cable to connect the PWM output pins on the TI launchpad to the input pins of the pin header (see the results section for the outline of the board). The output pins are on the opposite side.

Appendix II, Previous Design Iterations

Beamformer

Initially design of the beamformer was constructed on Matlab Simulink. Since the Simulink model cannot transform smoothly into real world component, the TI launchpad with built in microcontroller was introduced into the project. Theoretically Matlab can connect to the TI CCS software and control the LaunchPad directly from the Simulink model but the in our case, Matlab cannot keep track the memory and it failed to duplicate the reference signal. Therefore, we switched the direction to write the entire control program in C using TI CCS.

DAC Filter

Alternate considerations for the digital to analog converter ranged from the usage of a microcontroller using an analog output to the creation or purchase of an individual IC (integrated circuit) that could perform the digital to analog conversion. Although an individual IC (a traditional DAC chip) would certainly have performed the digital to analog conversion it would have been difficult to find a pre-prepared IC that could do so. In addition the construction of a custom IC would have not only been out of our range of expertise but also been expensive with high chance of failure. Additionally using a microcontroller with a built in digital to analog converter would have been expensive as most microcontroller systems boast only 1 output channel for such converters. This would have made the programming of the microcontroller especially cumbersome.

Initial designs using the lowpass filter idea were from the get go active low pass filter circuits as it was determined that the part values necessary to perform this in a passive low pass filter format would have difficult to obtain and the design itself would have been less flexible towards adjustment as the project progressed.

The second design that was created and simulated was a design that functioned similarly to the current design with the exception that it lacked the high pass filter portion. This filter used the ADA4891-3ARUZ opamp which was capable of handling the necessary frequency of the device. The device itself also had no gain and as such the three stage filter segment was only meant for filtering purposes. The main problem which resulted in this design being thrown out what the application of a DC offset when the duty cycle deviated from 50%. This led decision to add the highpass filter of the previous segment.

	ADA4891-3ARUZ, OpAmp
Channels	Triple Channel
Slew Rate	170 V/us
Settling Time	28ns
Unity Gain Bandwidth	220 MHz
Max Capacitive Load	6.8pF
Power Supply	Up to +/- 5.5V

This third design (see Figure A-1) was the only design that received a PCB test board and was initially projected to be the final design. The design uses the ADA4891-3ARUZ opamp in the three stage layout with a passive high pass filter following it. The circuit itself had unity gain and was meant to receive a source voltage of +/- 5V. While the simulation resulted in semi-acceptable results the testing of the physical PCB (see Figure 6-7) resulted in the immediate destruction of the opamp. After a series of tests were run the cause was determined to be the high pass filter. The capacitor in the filter paired with the load in the opamp resulted in the circuit exceeding the capacitive load of the opamp and a harmonic oscillator that ruined the opamp. The fix that was implemented after this was the change of opamp to one that was capable of receiving a higher capacitive load.

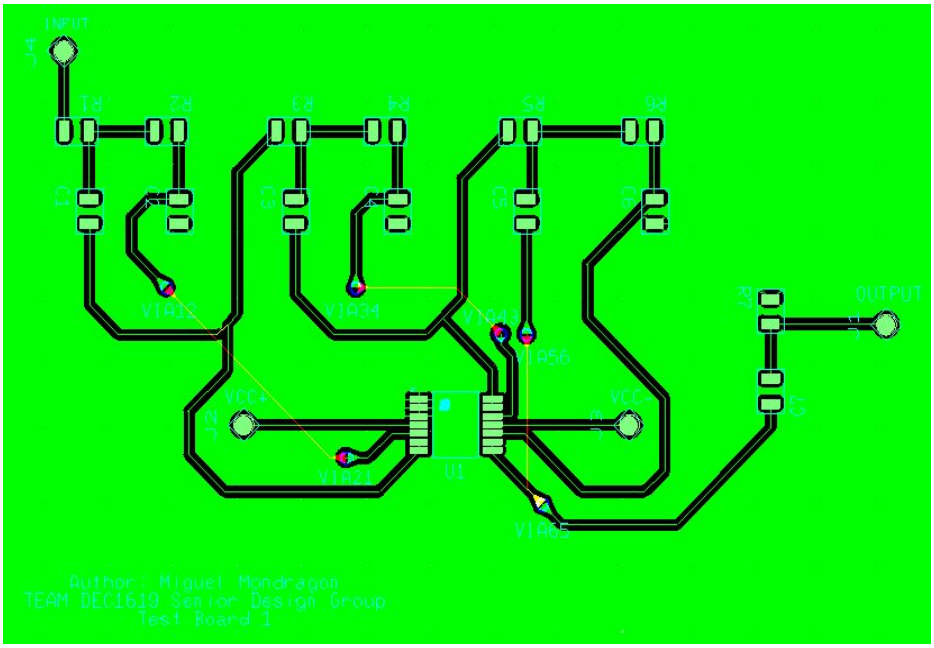


Figure A-1: Previous PCB design for DAC Filter

Amplifier

Initial CAD designs incorporated an “all-in-one” operational amplifier. This amplifier would be capable of raising both the voltage and current to an acceptable level to power the transducer. While the design would have been extremely simple and cost effective. An overlooked low slew rate and small gain bandwidth product made it infeasible.

Secondary design used a three stage amplification system. These stages were built in multisim as an attempt to fix the initial design of the single op-amp. Using a faster and lower powered operational amplifier as a preamplifier in order to circumvent the short fallings of the high-powered amplifier

After deciding to not use the high powered op amp, which had an expected current output of approximately 3A, a new solution to provide current gain would be required. A few circuits designed around bipolar junction transistors were developed, but only the common-collector design met the functional requirements of the project.

Appendix III, Code

Below is the code used to program the TI Launchpad.

```
/*
Seinor Design project
DEC - 19
PWM
Hongaho Liu
*/

//###########################################################################
#####
//To include the TI provided lib, in CCS project properties - C2000
Compiler - Include Options
//- add dir "C:\ti\controlSUITE\development_kits\C2000_LaunchPad"
//Then go to C2000 Linker - File Search Path
//-include library
"C:\ti\controlSUITE\development_kits\C2000_LaunchPad\f2802x_common\lib\dri
verlib.lib"
//###########################################################################
#####

#include "DSP28x_Project.h"     // Device Headerfile and Examples Include
File
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/pwm.h"
#include "f2802x_common/include/wdog.h"
void pwm_Init_();
void pwm_Init_two();
//void set_freq(int freq);
//void set_dutycycle(int duty);
#define TBPRD  12500  // Period register
#define CMPA   2500    //start the pwm at 20% duty cycle


CLK_Handle myClk;
GPIO_Handle myGpio;
PWM_Handle myPwm1,myPwm2;
```

```
void main(void)
{
    CPU_Handle myCpu;
    PLL_Handle myPll;
    WDOG_Handle myWDog;
    // Initialize all the handles needed for this application
    myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
    myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
    myGpio = GPIO_init((void *)GPIO_BASE_ADDR, sizeof(GPIO_Obj));
    myPll = PLL_init((void *)PLL_BASE_ADDR, sizeof(PLL_Obj));
    myPwm1 = PWM_init((void *)PWM_ePWM1_BASE_ADDR, sizeof(PWM_Obj));
    myPwm2 = PWM_init((void *)PWM_ePWM2_BASE_ADDR, sizeof(PWM_Obj));
    myPwm3 = PWM_init((void *)PWM_ePWM3_BASE_ADDR, sizeof(PWM_Obj));
    myPwm4 = PWM_init((void *)PWM_ePWM4_BASE_ADDR, sizeof(PWM_Obj));
    myPwm5 = PWM_init((void *)PWM_ePWM5_BASE_ADDR, sizeof(PWM_Obj));
    myPwm6 = PWM_init((void *)PWM_ePWM6_BASE_ADDR, sizeof(PWM_Obj));
    myPwm7 = PWM_init((void *)PWM_ePWM7_BASE_ADDR, sizeof(PWM_Obj));
    myPwm8 = PWM_init((void *)PWM_ePWM8_BASE_ADDR, sizeof(PWM_Obj));
    myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));
    // Perform basic system initialization
    WDOG_disable(myWDog);
    CLK_setOscSrc(myClk, CLK_OscSrc_Internal);
    PLL_setup(myPll, PLL_Multiplier_1, PLL_DivideSelect_ClkIn_by_4);
    CPU_disableGlobalInts(myCpu);
    CPU_clearIntFlags(myCpu);
    // Initalize GPIO
    GPIO_setMode(myGpio, GPIO_Number_0, GPIO_0_Mode_EPWM1A);
    GPIO_setMode(myGpio, GPIO_Number_1, GPIO_2_Mode_EPWM2A);
    GPIO_setMode(myGpio, GPIO_Number_2, GPIO_2_Mode_EPWM2A);
    GPIO_setMode(myGpio, GPIO_Number_3, GPIO_2_Mode_EPWM2A);
    GPIO_setMode(myGpio, GPIO_Number_4, GPIO_2_Mode_EPWM2A);
    GPIO_setMode(myGpio, GPIO_Number_5, GPIO_2_Mode_EPWM2A);
    GPIO_setMode(myGpio, GPIO_Number_6, GPIO_2_Mode_EPWM2A);
    GPIO_setMode(myGpio, GPIO_Number_7, GPIO_2_Mode_EPWM2A);
    GPIO_setMode(myGpio, GPIO_Number_8, GPIO_2_Mode_EPWM2A);
    //set_freq(10);
    //set_dutycycle(75);
    CLK_disableTbClockSync(myClk);
    pwm_Init_
      pwm_Init_two
    CLK_enableTbClockSync(myClk);
    while(1);
}
```

```
/*void set_freq(int freq)
{
 float Tpwm,c;
 Tpwm = 1/freq;
 c = Tpwm/0.000008;
 TBPRD = (int)12500.98;
}
void set_dutycycle(int duty)
{
 float div;
 div = 100/duty;
 CMPA = (int)(TBPRD/div);
 CMPA = (int)6250.123;
}*/
void pwm_Init_()
{
    CLK_enablePwmClock(myClk, PWM_Number_1);
    // Setup TBCLK
    PWM_setPeriod(myPwm1, TBPRD);   // Set timer period 801 TBCLKs
    PWM_setPhase(myPwm1, 0x0000);   // Phase is 0
    PWM_setCount(myPwm1, 0x0000);   // Clear counter
    // Set Compare values
    PWM_setCmpA(myPwm1, CMPA);      // Set compare A value
    // Setup counter mode
    PWM_setCounterMode(myPwm1, PWM_CounterMode_UpDown);  // Count up and
down
    PWM_disableCounterLoad(myPwm1);                      // Disable phase
loading
    PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio to
SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm1, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm1, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm1, PWM_ActionQual_Clear);
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm1, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}

void pwm_Init_two()
{
      CLK_enablePwmClock(myClk, PWM_Number_2);
      //set TBCLK
      PWM_setPeriod(myPwm2, TBPRD);   // Set timer period 801 TBCLKs
```

```
    PWM_setPhase(myPwm2, 1334);   // set phase direction
    PWM_setCount(myPwm2, 0x0000);   // Clear counter
      //set Compare value
      PWM_setCmpA(myPwm2, CMPA);      // Set compare A value
      // Setup counter mode
      PWM_setCounterMode(myPwm2, PWM_CounterMode_UpDown);  // Count up and
down
      PWM_disableCounterLoad(myPwm2);                     //Disable phase
loading
      PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio
to SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm2, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm2, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm2, PWM_ActionQual_Clear);
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm2, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}
//===========================================================================
// output channel at J6 and GND.

//===========================================================================
void pwm_Init_three()
{
      CLK_enablePwmClock(myClk, PWM_Number_3);
      //set TBCLK
      PWM_setPeriod(myPwm3, TBPRD);   // Set timer period 801 TBCLKs
    PWM_setPhase(myPwm3, 1334);   // set phase direction
    PWM_setCount(myPwm3, 0x0000);   // Clear counter
      //set Compare value
      PWM_setCmpA(myPwm3, CMPA);      // Set compare A value
      // Setup counter mode
      PWM_setCounterMode(myPwm3, PWM_CounterMode_UpDown);  // Count up and
down
      PWM_disableCounterLoad(myPwm3);                     //Disable phase
loading
      PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio
to SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm2, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm2, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm2, PWM_ActionQual_Clear);
```

```
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm2, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}
void pwm_Init_four()
{
      CLK_enablePwmClock(myClk, PWM_Number_4);
      //set TBCLK
      PWM_setPeriod(myPwm4, TBPRD);   // Set timer period 801 TBCLKs
    PWM_setPhase(myPwm4, 1334);   // set phase direction
    PWM_setCount(myPwm4, 0x0000);   // Clear counter
      //set Compare value
      PWM_setCmpA(myPwm4, CMPA);      // Set compare A value
      // Setup counter mode
      PWM_setCounterMode(myPwm4, PWM_CounterMode_UpDown);  // Count up and
down
      PWM_disableCounterLoad(myPwm4);                      //Disable phase
loading
      PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio
to SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm2, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm2, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm2, PWM_ActionQual_Clear);
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm2, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}
void pwm_Init_five()
{
      CLK_enablePwmClock(myClk, PWM_Number_5);
      //set TBCLK
      PWM_setPeriod(myPwm5, TBPRD);   // Set timer period 801 TBCLKs
    PWM_setPhase(myPwm5, 1334);   // set phase direction
    PWM_setCount(myPwm5, 0x0000);   // Clear counter
      //set Compare value
      PWM_setCmpA(myPwm5, CMPA);      // Set compare A value
      // Setup counter mode
      PWM_setCounterMode(myPwm5, PWM_CounterMode_UpDown);  // Count up and
down
      PWM_disableCounterLoad(myPwm5);                      //Disable phase
loading
      PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio
to SYSCLKOUT
```

```
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm2, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm2, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm2, PWM_ActionQual_Clear);
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm2, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}
void pwm_Init_six()
{
      CLK_enablePwmClock(myClk, PWM_Number_6);
      //set TBCLK
      PWM_setPeriod(myPwm6, TBPRD);   // Set timer period 801 TBCLKs
    PWM_setPhase(myPwm6, 1334);   // set phase direction
    PWM_setCount(myPwm6, 0x0000);   // Clear counter
      //set Compare value
      PWM_setCmpA(myPwm6, CMPA);      // Set compare A value
      // Setup counter mode
      PWM_setCounterMode(myPwm6, PWM_CounterMode_UpDown);  // Count up and
down
      PWM_disableCounterLoad(myPwm6);                      //Disable phase
loading
      PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio
to SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm2, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm2, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm2, PWM_ActionQual_Clear);
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm2, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}
void pwm_Init_seven()
{
      CLK_enablePwmClock(myClk, PWM_Number_7);
      //set TBCLK
      PWM_setPeriod(myPwm7, TBPRD);   // Set timer period 801 TBCLKs
    PWM_setPhase(myPwm7, 1334);   // set phase direction
    PWM_setCount(myPwm7, 0x0000);   // Clear counter
      //set Compare value
      PWM_setCmpA(myPwm7, CMPA);      // Set compare A value
      // Setup counter mode
      PWM_setCounterMode(myPwm7, PWM_CounterMode_UpDown);  // Count up and
```

```
down
      PWM_disableCounterLoad(myPwm7);                        //Disable phase
loading
      PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio
to SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm2, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm2, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm2, PWM_ActionQual_Clear);
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm2, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}
void pwm_Init_eight()
{
      CLK_enablePwmClock(myClk, PWM_Number_8);
      //set TBCLK
      PWM_setPeriod(myPwm8, TBPRD);   // Set timer period 801 TBCLKs
    PWM_setPhase(myPwm8, 1334);   // set phase direction
    PWM_setCount(myPwm8, 0x0000);   // Clear counter
      //set Compare value
      PWM_setCmpA(myPwm8, CMPA);      // Set compare A value
      // Setup counter mode
      PWM_setCounterMode(myPwm8, PWM_CounterMode_UpDown);  // Count up and
down
      PWM_disableCounterLoad(myPwm8);                        //Disable phase
loading
      PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_10); // Clock ratio
to SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);              // Setup
shadowing
    PWM_setShadowMode_CmpA(myPwm2, PWM_ShadowMode_Shadow);
    PWM_setLoadMode_CmpA(myPwm2, PWM_LoadMode_Zero);     // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm2, PWM_ActionQual_Clear);
// Set PWM1A on event A, up count
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm2, PWM_ActionQual_Set);  //
Clear PWM1A on event A, down count
}
//============================================================================
// output channel at J14 and GND.

//============================================================================
```