Design Document Version 2

Low-Cost High Accuracy Spectral Test System

Iowa State University

Senior Design: 2015-2016

Team May1623

Advisor/Client: Degang Chen

Group Members: Tao Chen, Scott Poder, Yifan Jiang

Table of Contents

Project Statement

Accurate spectral test is widely used in science and engineering. In particular, accurate spectral test is critical to high performance data converters used base stations, medical instruments, seismic signal detection, military applications, and so on. IEEE standards and prevalent industry solutions impose several stringent requirements on the linearity of input signals, on exact coherency in sampling, and on tolerable jitter in the clock signal. All of these requirements make accurate spectral testing expensive and time consuming, and make the test setup difficult to maintain and calibrate.

The goal of this project is to develop a prototype test system for Extremely Cost-Effective Spectral Test. Recently published spectral test algorithms will be implemented for dramatically relaxing the stringent requirements. The concrete objective is to demonstrate a PCB test system for very low-cost, high-accuracy full spectrum test for high performance ADCs from Texas Instruments. The PCB will include a low cost sine wave generator, a clock generator, a low-order RC filter block, an ADC input driver, a socket for an ADC under test, and ADC output collection. The collected data will be transferred to a computer for analysis and display. Spectral test results will be compared with results obtained from Audio Precision instruments.

System Level Design

The system level block diagram can be seen below in Figure 1. For this test board, we will be using a DAC to generate an impure sine wave. This sine wave will be passed through a buffer for isolation and will be exposed to one of two filters. One of the filters will be an RC Filter while the other will be an RR Filter or voltage divider. Only one filter will be selected at a time, and our board's logic will switch between the two filters as needed. This output will be buffered and sent to the ADC under test. In real time, the digital output data of the ADC will be sent back to our FPGA in real time. From here, we will use spectral test algorithms on the PC to interpret the results of the ADC under test.

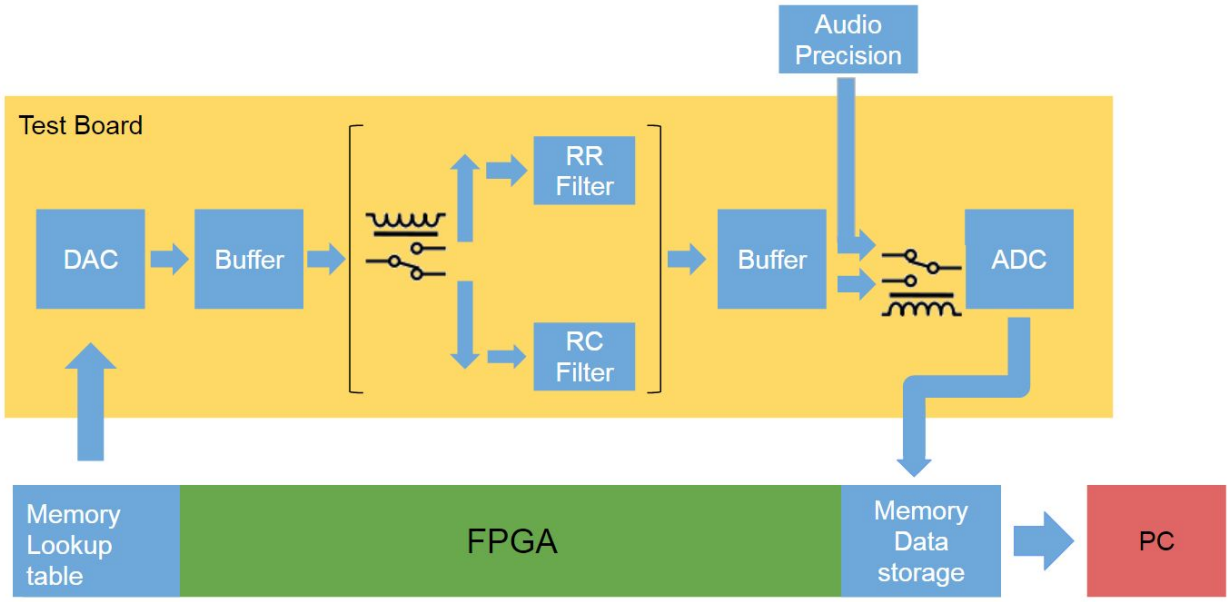

Figure 1: Block Diagram of the Low-Cost High Accuracy Spectral Test System

Detailed Description/Implementation

Overall System

A more detailed view of our ADC Test Board can be seen below in Figure 2. Here we see that the test board and the FPGA interface with one another by the male header pins J1. Digital Logic, SPI, and Clock signals enter the test board via J1 and proceed to get buffered in HB5. Next, the appropriate signals go to the DAC, Filter, Audio Precision Switch, and ADC.

As described above, these digital signals interface with our DAC in HB1 and generate a ±2.5V sine wave. This sine wave gets filtered by our RC and RR filters in HB2, and due to gain calculations results in a ±4.3V sine wave by design (described in a later section). Next, we go to HB4, which contains a relay as a switch, which we use digital signals to determine whether we wish our ADC to read signals from the DAC or the Audio Precision Equipment. Finally, our sine wave travels to HB3, which contains our ADC under test. Lastly, the digital output of our ADC gets sent back to a header pin on J1, which causes data to reach our FPGA. These pin assignments can be seen below in Table 1.

Pin	Signal
1	SCLK
2	nCS_DAC (Inverted Chip Select DAC)
3	DIN (Digital IN [DAC Data Input])

4	RC1
5	nRC1
6	RC2
7	nRC2
8	RC3
9	nRC3
10	RC4
11	nRC4
12	RC5
13	nRC5
14	RC6
15	nRC6
16	RC7
17	nRC7
18	RR
19	nRR
20	PURE_ON (relay switch to AP)
21	DAC_ON (relay switch to DAC)
22	nCS_ADC (inverted Chip Select ADC)
23	DO (Digital Out [ADC Data Out])
24	Digital GND

Table 1: The Pin Assignments for J1 with Pin 1 being the leftmost pin.

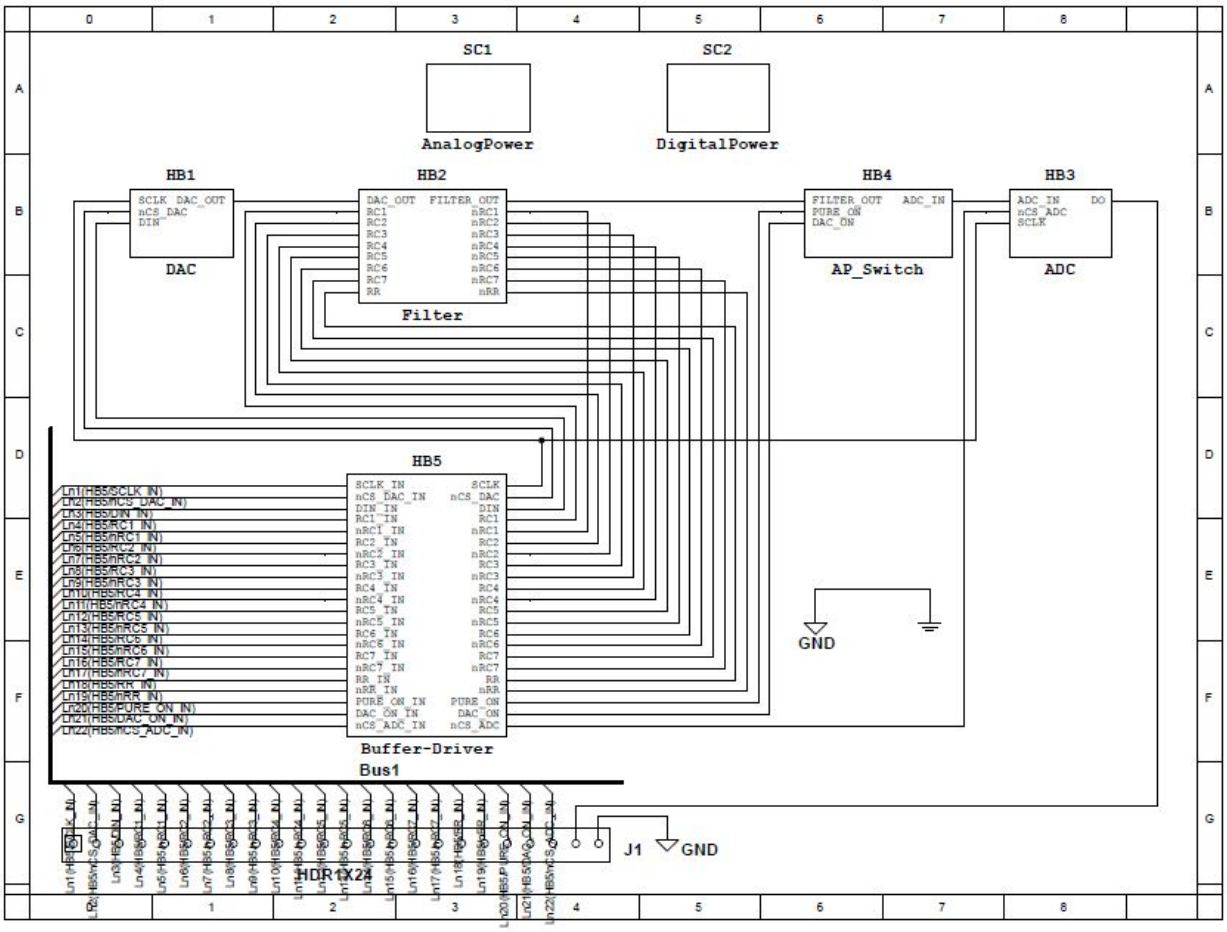

Figure 2: A hierarchical block view of our ADC Test Board in Multisim.

Power Supplies

In order to provide power to the test board, we will be using multiple 9V batteries to supply power to the board. Obviously, not all parts run at the same voltage, so voltage regulations will be used to obtain ±5V and 3.3V where appropriate in the system. For the analog portion of the circuit, we will be using LDOs in the form of the parts uA7805 and uA7905, seen below in Figure 3 (shown as LM7805CT and LM7905CT, for they are equivalent) to obtain ±5V. The maximum output current for this Regulated Dual Supply is 1.5A. The outputs of these LDOs will be considered clean, which is important for the precision of the analog portions of the schematic. In addition, analog and digital supply voltages must be separated from one another for noise purposes, therefore, we will need to use separate voltage regulators for the digital/analog power supplies on the board.

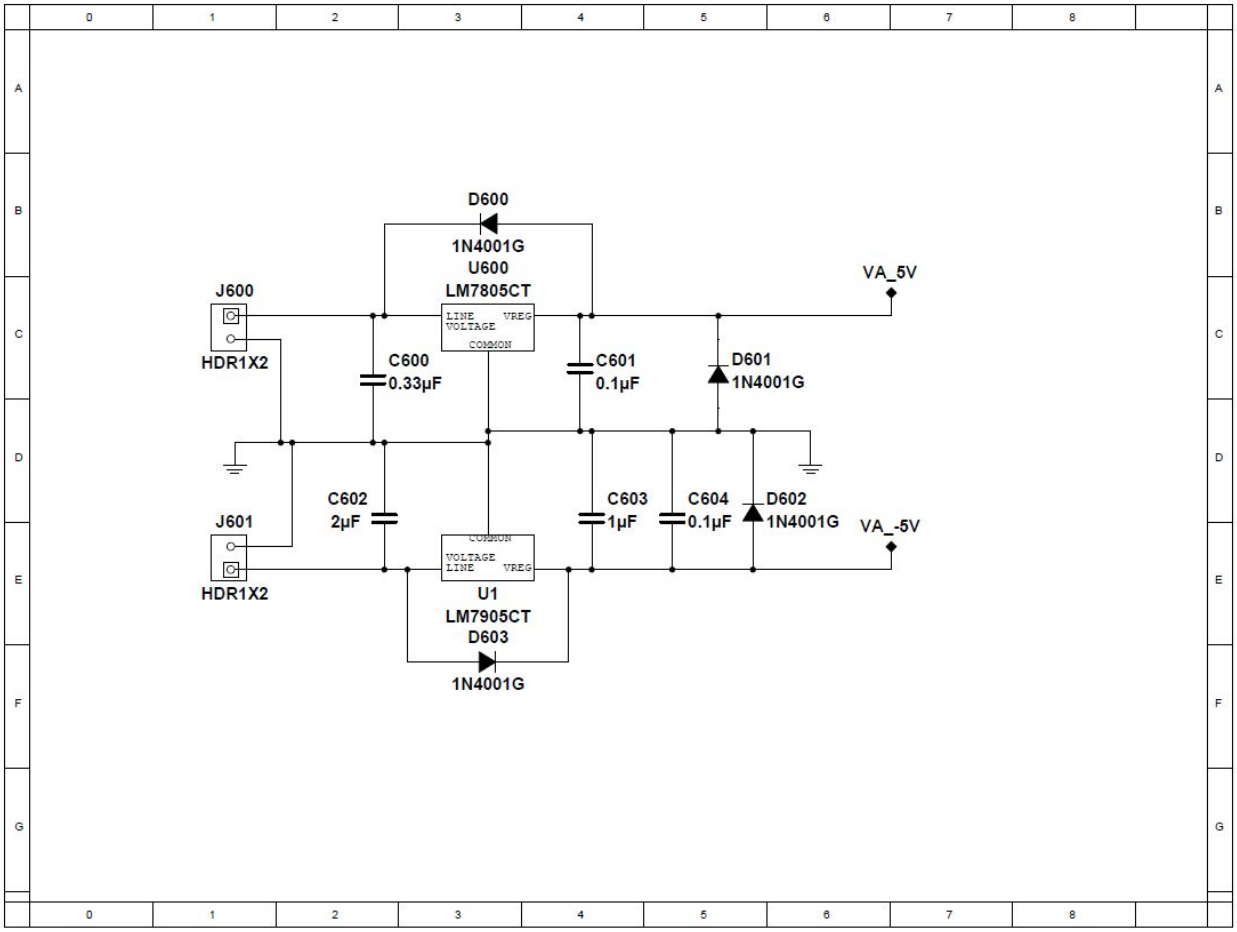

Figure 3: The Analog Regulated Dual Power Supply providing ±5V to our board.

For the digital power supply of the circuit, we designed a synchronous buck converter takes a 9V battery as an input and outputs 3.3V and a maximum current of 3A. This was important because we are powering a lot of digital inverters, relays, and relay drivers, and need this high amperage to satisfy all of our power supply needs. Also, since this buck converter is serving as a digital power supply, it does not need to be as clean as our analog power supply LDOs. Below in Figure 4 is out designed buck converter.

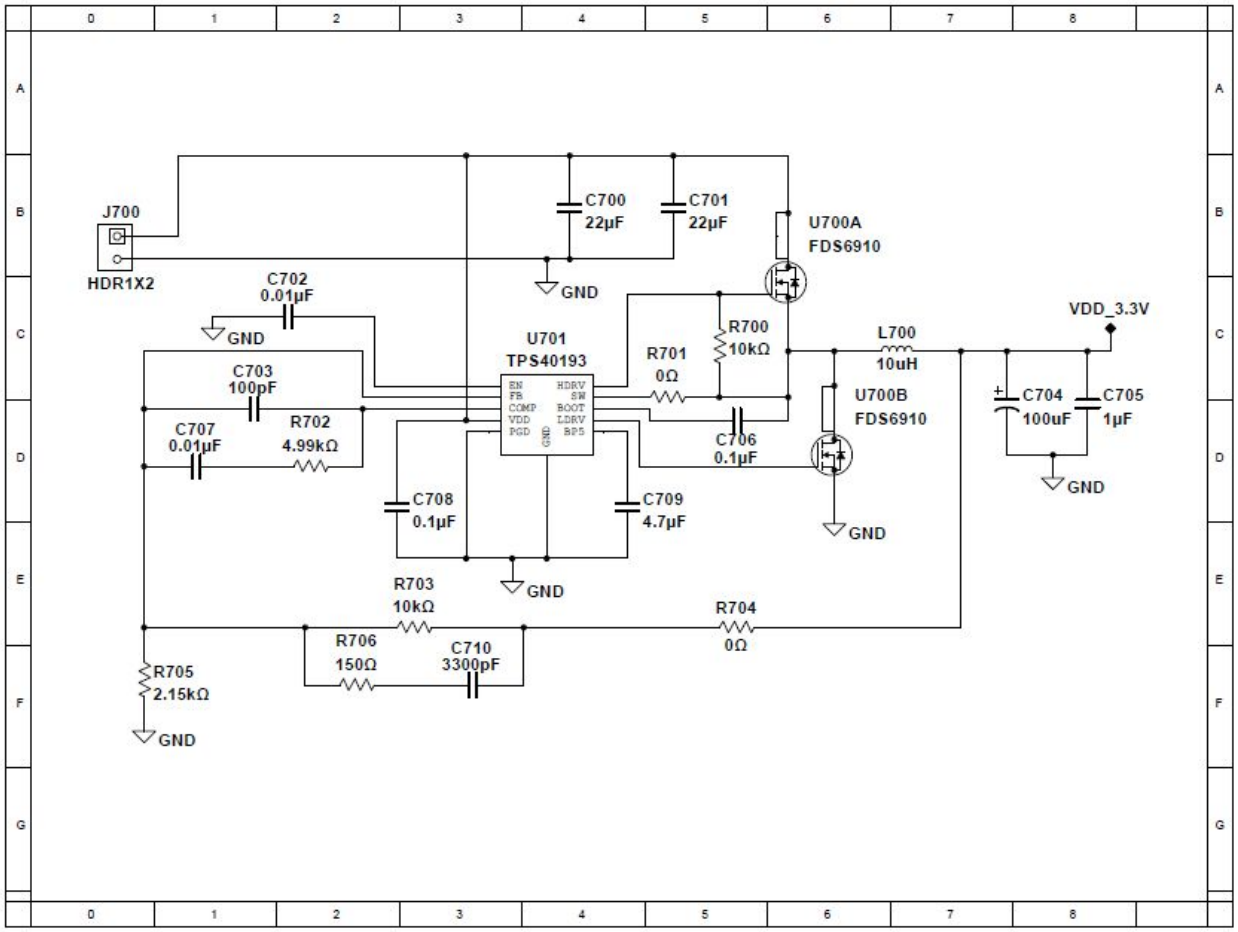

Figure 4: The Synchronous Buck Converter which provides 3.3V Digital Supply to the board.

Digital Buffers/Driver

As mentioned earlier, our digital signals from the FPGA must be run through a buffer in order to clean up the signal and remove jitter. We utilized two inverters to do this seen below in Figure 5 (74AC11004). Also, the digital logic which was responsible for the digital switching of our relays required relay drivers in order to ensure sufficient performance. We selected DRV777 to serve as our relay drivers for the appropriate signals. Since all of these components were digital, they received power from our 3.3V Buck Converter.

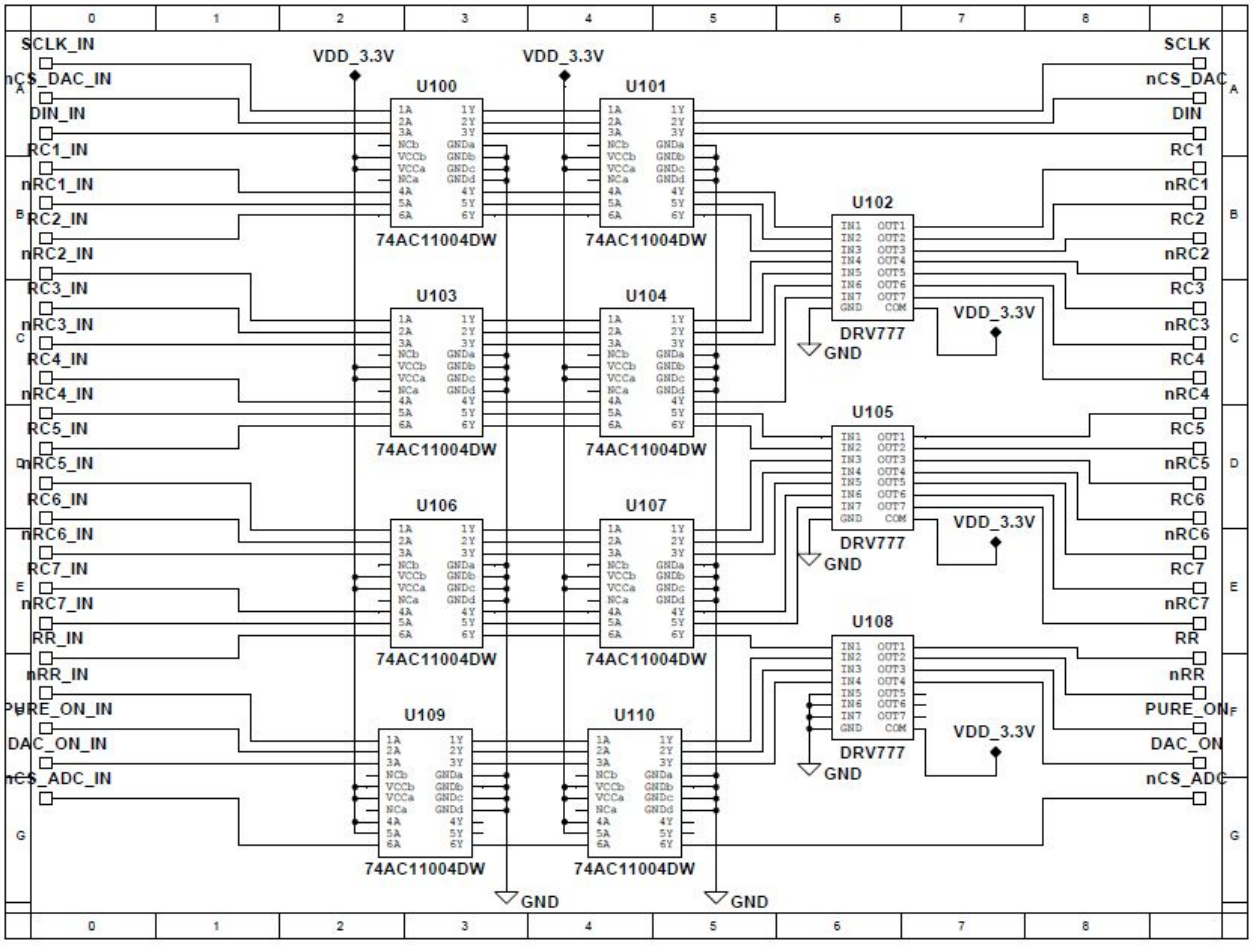

Figure 5: The Buffers and Drivers for our Digital Signals from the FPGA.

DAC

The Digital to Analog Converter (DAC) that we selected for the board is the DAC8831 from Texas Instruments. This DAC is a 16-bit, low power, voltage output DAC whose architecture is of the R-2R structure. This DAC was chosen due to its sample rate (2MSPS) being larger than the sampling rate of the ADC (1MSPS). This is important for precision of our spectral tests.

We are using the DAC to generate an impure sine wave. The design that we will be using will generate a sine wave with an amplitude of ±2.5V. In order to do this, we needed to ensure that our DAC was wired to bipolar configuration. To make our project different and innovative, we decided to generate multiple sine waves with varying frequencies, so that we can observe how varying frequency affects our test results. We chose seven frequencies that were pretty evenly spaced between 1kHz and 20kHz. These frequencies were 1.0332kHz, 3.3827kHz, 6.888kHz, 10.332kHz, 13.776kHz, 16.665kHz, and 20.259kHz. We chose these based off the resistor and capacitor values available to us (described in more detail below in the Filter/Buffer Section).

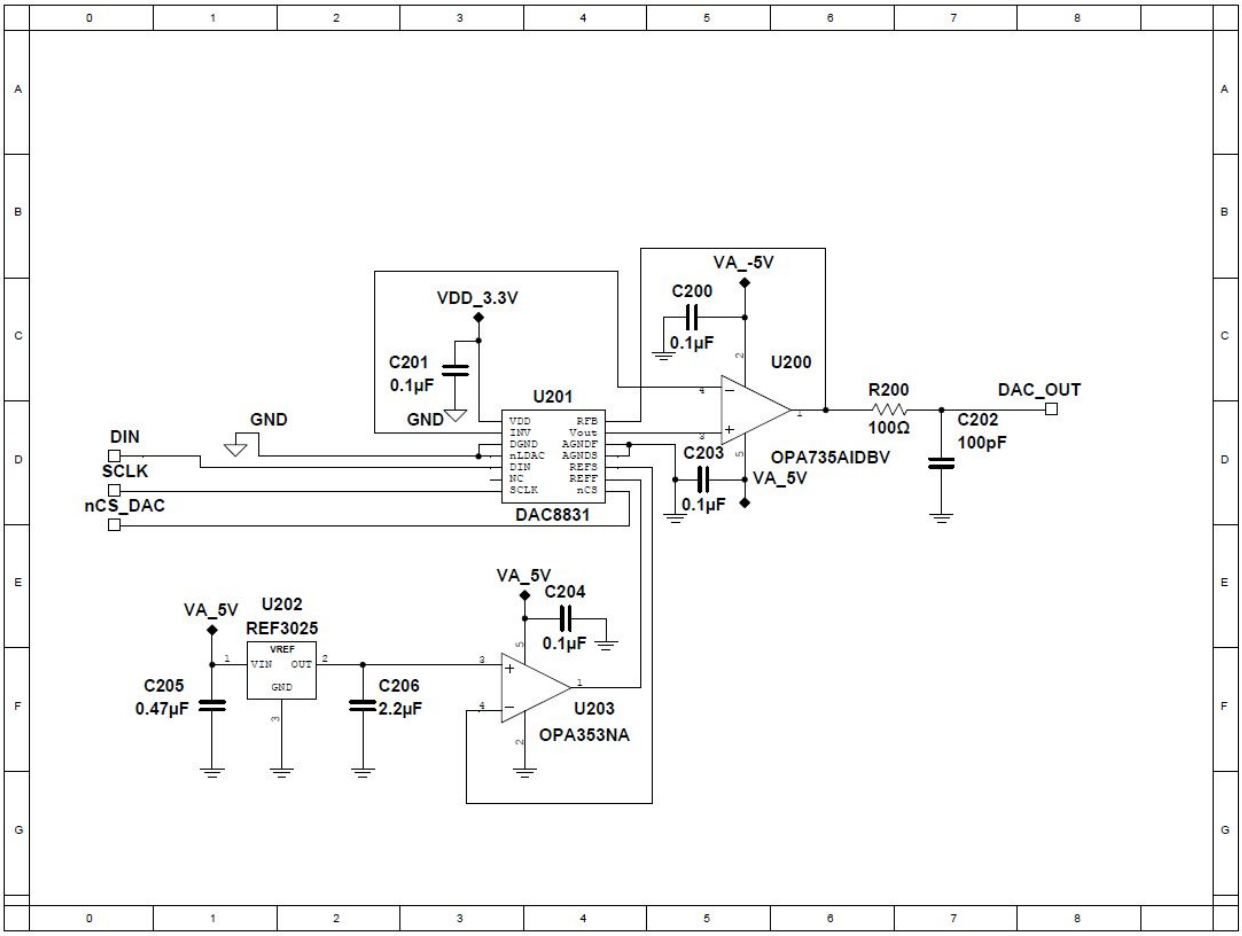

Figure 6: The DAC and its supporting circuitry.

As seen above in Figure 6, our DAC (U201) is powered with 3.3V from our Buck Converter. Also, 5V analog power comes in to REF3025 (U202), which converts the 5V power supply to a 2.5V power supply. After this, it is buffered by OPA353 (U203) and sent to the voltage reference input pins REFS and REFF. These reference pins dictate that the maximum output amplitude of the DAC will be 2.5V. The output ±2.5V sine wave of our DAC gets sent to U200, which serves as an isolation buffer, and then gets noise filtered by R200 and C202. After this, our sine wave gets sent to our filter design.

Filter Design

For the test board, we will have two filters present. One is low pass RC Filter while the other is a RR Filter or voltage divider. Only one filter will be selected at a time. This selection will be handled digitally with the use of low signal relays (RY300-307) which will be controlled by the FPGA. The RR Filter will be designed so that it contains a constant attenuation of -3dB. The RC Filter will be designed so that the corner frequency will be at -3dB. The reason that we are doing this is because our algorithms for low-cost high-accuracy spectral test require both a phase shifted output (RC Filter) as well as a matched amplitude output (RR Filter). From here using the differences in the signals, we are able to obtain similar spectral test results to the traditional method. Due to our design requiring multiple frequencies, we included a

capacitor bank of various values and used the relays to obtain a wide range of RC Filters with the corner frequencies matching our input frequencies. These frequencies are 1.0332kHz, 3.3827kHz, 6.888kHz, 10.332kHz, 13.776kHz, 16.665kHz, and 20.259kHz. We obtained component values for these filters by setting R303 and R304 to chosen values, and using the following equation to obtain R305:

R305 = (R303*R304)/((sqrt(2)-1)*R303-R304)

Similarly, we used these resistor values to find the capacitance that we needed in order to obtain the frequencies that we desired (ω = 2*pi*f):

C30X = sqrt((R304^2-R303^2-2*R303*R304)/(R303^2*R304^2*ω^2))

In order to provide isolation between the Filter and the ADC/DAC, we must include buffers around it. For these, we will design one inverting amplifier circuit at the filter input and one at the filter output. Since our design calls for our DAC to output a ±2.5V sine wave and our ADC to receive a ±4.3V sine wave, the overall filter/buffer segment must provide a gain of 1.72. However, we must also account for the 1/sqrt(2) attenuation given by the filters, therefore our adjusted gain for this system is 1.72*sqrt(2) = 2.43. Below in Figure 7 is a representation of what we desire.

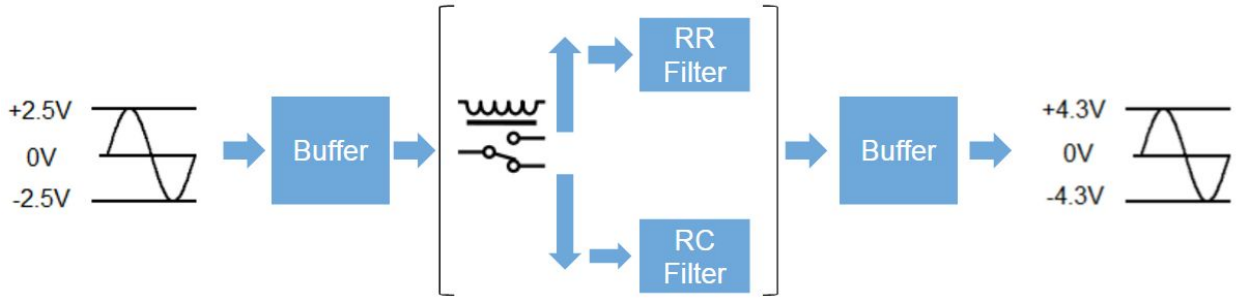

Figure 7: An overall block diagram of our filter design.

In order to ensure that the values which we settled on are correct, we ran a simulation in TINA-TI, which is Texas Instrument's SPICE Based Analog Simulation Program. We set up the test circuit for the filter below and simulated the output when we fed a ±2.5V sine wave at the seven frequencies that we chose. Below is our TINA-TI simulation and results for a 10.332kHz sine wave seen in Figures 8-10. As we see below, the green ±2.5V sine wave represents the input sine wave, and the red sine wave is our output sine wave. We can clearly see for both the RC and RR Filters, that our output sine wave is of magnitude ±4.3V, therefore the simulation checks out and is a success.

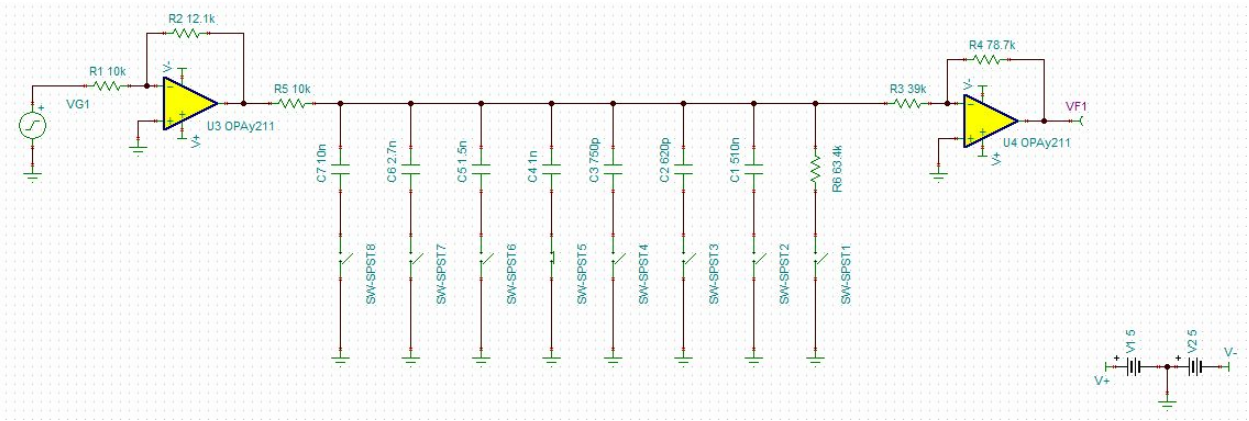

Figure 8: Our TINA-TI Filter Design Simulation with the 10.332kHz capacitor selected.

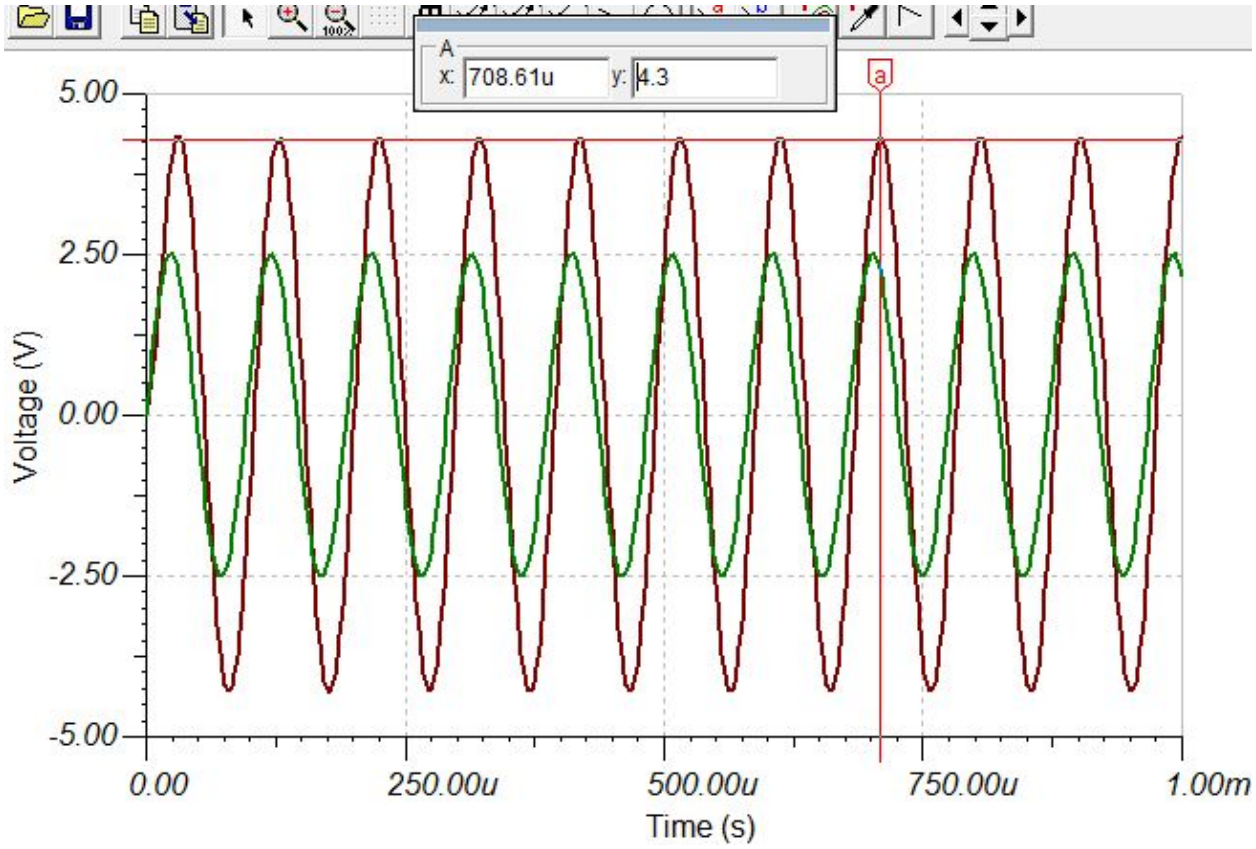

Figure 9: The RC Filter Simulation Results ran at 10.332kHz, where the green is the input and the red is the output.

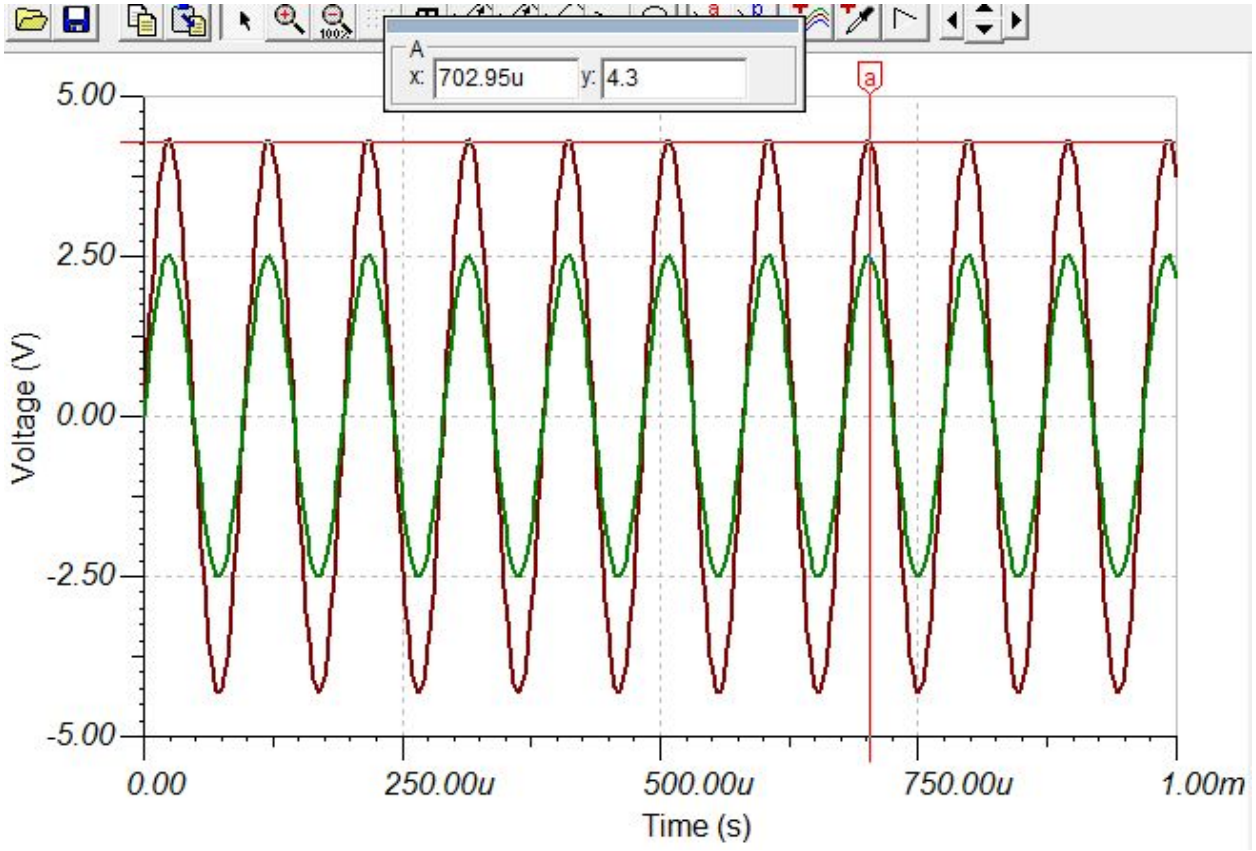

Figure 10: The RC Filter Simulation Results ran at 10.332kHz, where the green is the input and the red is the output.

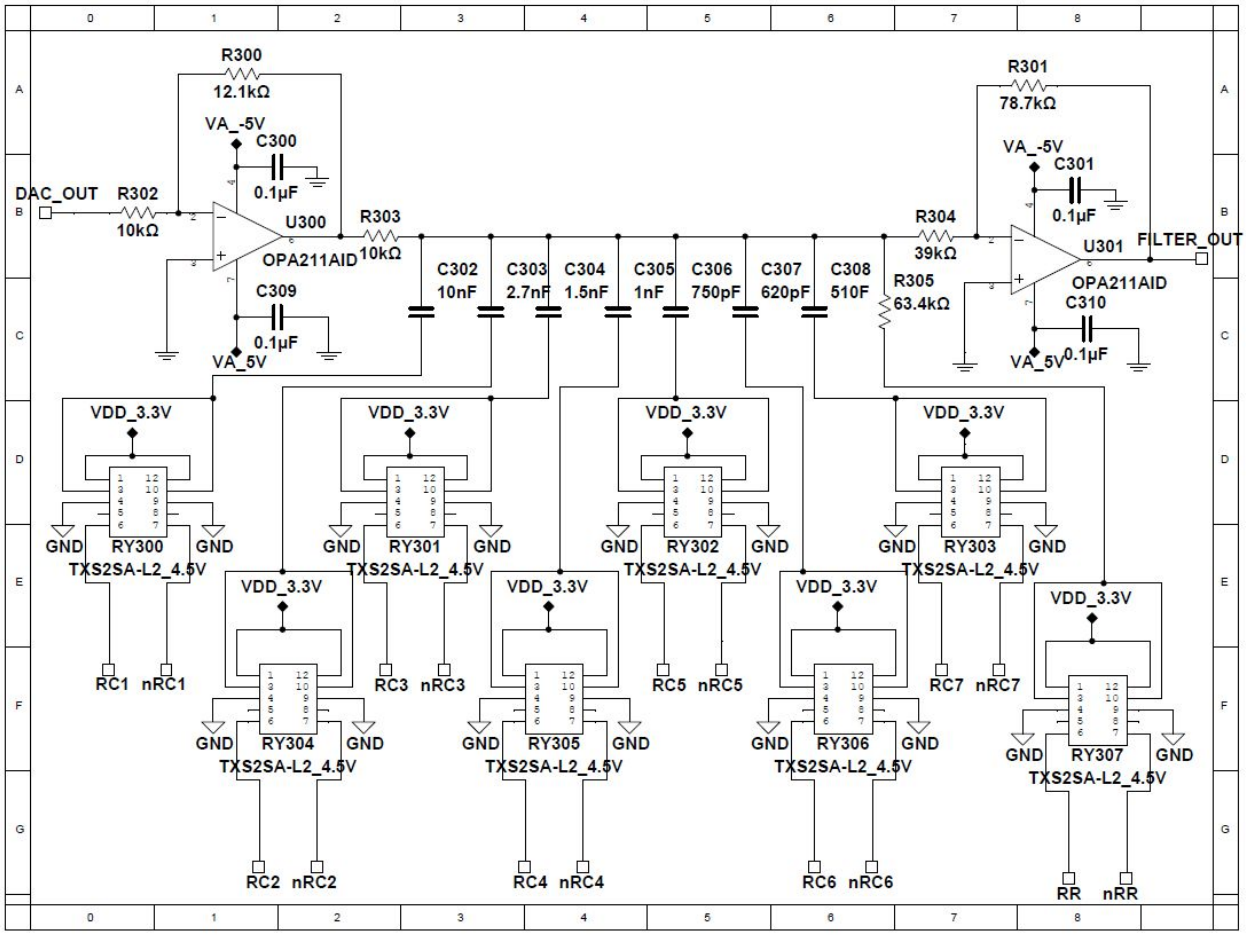

Figure 11: The Buffers and Filters for our low-cost high-accuracy spectral test.

Audio Precision Switch

For this design, we will want to compare the non traditional method of spectral test which we are implementing with the traditional method (using Audio Precision Equipment). Therefore, we used a SMA connector (J400), an isolation buffer in OPA211 (U400), and a relay to digitally switch between the DAC's impure sine wave and the Audio Precision's pure sine wave. This can all be seen below in Figure 12.

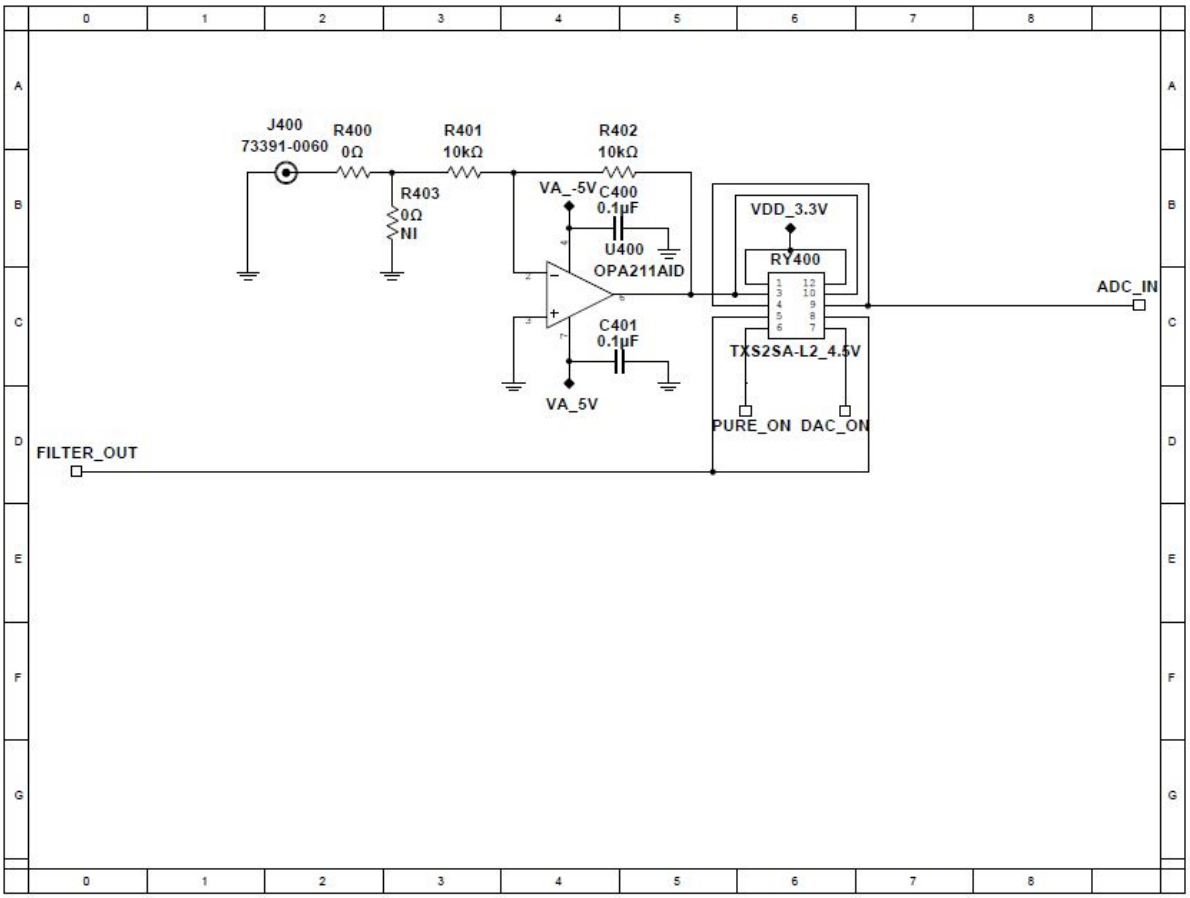

Figure 12: The Switching Relay between the Impure DAC sine wave and the pure AP sine wave.

ADC

The Analog to Digital Converter (ADC) under test will be the ADS8881 from Texas Instruments. This ADC is an 18-bit, 1-MSPS, SAR (successive approximation), precision ADC that operates on very low power. The ADC is also takes in a dual-differential input, which we have to account for in our input driver, which is described below. On the actual test board, a socket (U503) will be in place so that we can run the spectral test on a large quantity of ADCs. The ADC contains some supporting circuitry around it, including a reference driver and an input driver which can be seen below in Figure 13.

It is also important to note that the ADC will be operating in 3-wire mode for SPI communication between the Test Board and the FPGA. The entire test board is designed around this ADC, so its selection was the first step in our design process.

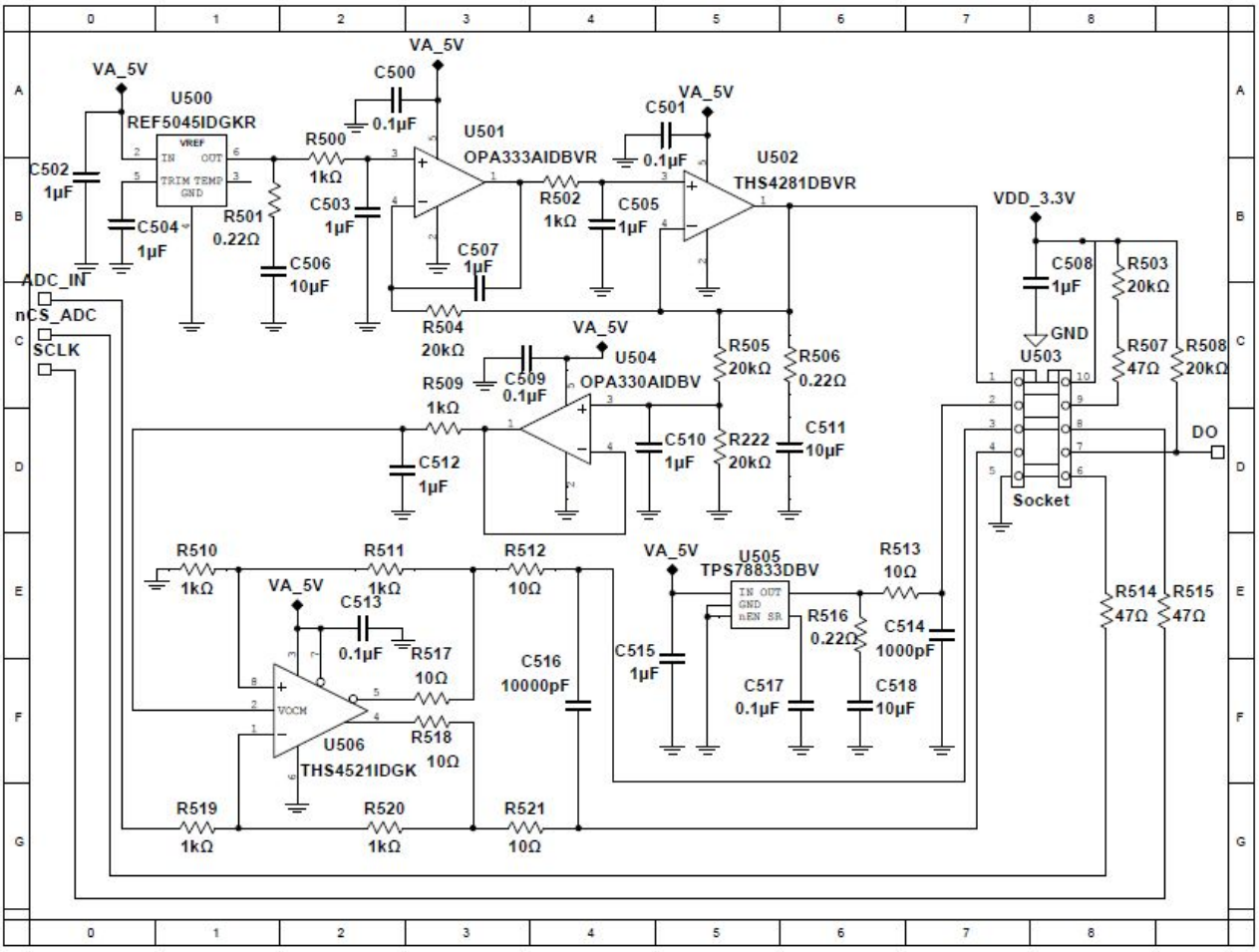

Figure 13: The ADC under test along with its biasing circuitry.

In Figure 13 above, 5V analog supply is provided from our LDO to REF5045 (U500). This regulator brings the voltage level down from 5V to 4.5V. Next, OPA333 and THS428 (U501 and U502) serve as both buffers, filters, and feedback to isolate the components from one another, filter out any noise, and provide a clean 4.5V to pin 1 of our ADC. Pin 1 of our ADC (U503 above) was defined as Analog Reference Input, and since we are providing 4.5V to it, this means in our results the maximum the ADC will convert from Analog to Digital is a 4.5V signal (to 18 bits worth of 1s). The entirety of this circuit makes up our Reference Driver for the ADC.

Next, the 4.5V at pin 1 goes through a voltage divider seen by R505 and R222, buffered by OPA330 (U504), and noise filtered by R509 and C512. This results in a voltage of 2.5V. This voltage supplies the common mode reference to our differential input amplifier at U506 (THS4521). This differential input amplifier is set up in Single Ended Signal Configuration, which means that we only need to provide it one signal instead of two (for the other is permanently tied to ground). The signal that gets provided is the ±4.3V that is the output of our Filter Design. The differential input amplifier has the effect that pins 3 and 4 of the ADC (the analog inputs) will always sum up to the reference voltage. Therefore, if we apply 4.5V to ADC_IN, we will see that pin 3 (the noninverting pin) is at 4.5V and 0V at pin 4 (the inverting pin).

Additionally, if we apply 2V at ADC_IN, we will obtain 2.0V at the noninverting pin and 2.5V at the inverting pin.

To provide analog power to the ADC at pin 2, our LDOs supply 5V analog supply to TPS78833 (U505), which brings it down to 3.3V. Then, this 3.3V gets noise filtered and sent to pin 2, the analog power supply pin. Additionally, we tie pin 5 to ground, which is the device ground, and tie pin 10 to the 3.3V digital supply from the Buck Converter (digital power supply pin). We supply clock signal, chip select, and receive ADC_digital out (our test results), which gets sent to the FPGA via J1. This concludes the in depth schematic design of our ADC Test Board.

Clock

Ideally, we would like to use the fastest clock signal possible. We choose to have the clock signal for the ADC and DAC be shared, since we are using 3-wire SPI to communicate with both of them. This would also simplify things, for they would always be synchronized with each other in the digital timing. After looking at the datasheets, we discover that 50MHz is the ideal, fastest clock frequency that satisfies both chip requirements.

We decided to use the clock drive from FPGA board. FPGA provide maximum of 100 MHz clock frequency, so it satisfies our specifications since this is enough for both ADC and DAC operate clock frequency as mentioned in the previous paragraph. In order to get 50MHz from the FPGA main clock's frequency, we can use Johnson Ring Counter, this type of shift register can divide a digital signal by an even integer multiple, which fits our case perfectly with the integer being 2.

FPGA/Memory

We will choose FPGA - Cyclone V from ALTERA for our design because to generate sine wave, we need write a lookup table with 16-bits worth of sine wave value into a ROM. It is easy to achieve from FPGA. Also, we listed above in the Clock section that this FPGA is easy for use to design a clock divider, and feed the desired clock frequency into DAC and ADC. FPGA can reduce the difficult level of our project, and since they have spare ones at our parts shop and in our labs, this decision makes a lot of sense for us.

After we obtain the ADC digital output with our test results, we will store this data in the FPGA's on-board memory. Then, we can transfer the data via USB drive to MATLAB, where we will run our algorithms.

Serial Peripheral Interface

The chip to chip communication protocol we are going to use is called *Serial Peripheral Interface.* Basically, this protocol uses master to slave architecture to communicate between chips. There are essentially four pins we concerned about, SCLK (serial clock), MOSI (master output slave input, output from master), MISO (master input slave output, output from slave), SS (slave select, output from master).

In some situations there might be only three pins, for example, DAC8831. We use 3 pins to control it: SCLK (serial clock), SDI (serial data input, same as MOSI, digital codes are going to be sent into this pin sequentially), and nCS (inverse chip select, same as SS). Below in Figure 14, we see our ModelSim Simulation for our timing diagrams for our ADC and DAC (ADC and DAC share the same clock).

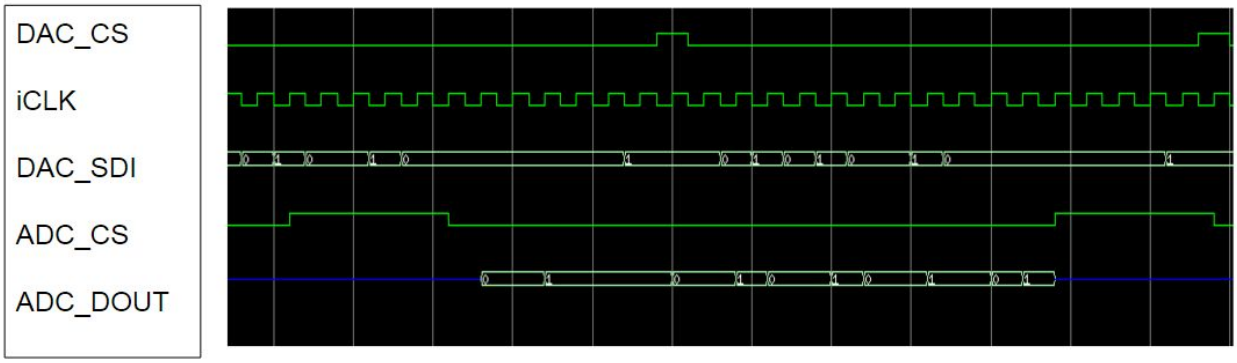

Figure 14: The ModelSim Simulation of our SPI.

PC Interface

There are two types of software that are used to interface between the PCB and FPGA, Quartus and MATLAB. Quartus is used to control the FPGA, and MATLAB is used to generate the input wave information for the DAC. For our case in generating the sine wave lookup table, MATLAB will be used to generate a file which contains the information of sine wave, like data points, frequency and amplitude. Then, this file is used on the FPGA with the assistance of Quartus to control the DAC. When working with Quartus, we work heavily in Verilog and Modelsim when it comes to memory mapping the data, as well as generating the SPI communication.

Algorithms

The algorithms were provided by Dr. Chen as MATLAB code for the nontraditional spectral test. We will use this new algorithms to manipulate our collected data, and obtain comparable results to the traditional spectral test using Audio Precision Equipment. Please refer to Benjamin Magstadt's Master Thesis "Relaxing the requirements for accurate testing of data converters" for more information.

PCB Strategy and Design

We decided that when we created our printed circuit board, we would make a four layer board. The top layer would consist of the majority of the components and routing. The second layer would be a ground plane, with analog and digital ground separated, except for a slot in the middle. This would serve as a system star ground, where this slot connecting the two ground planes consisted of about 5% of the width of the board. The third layer would consist of power planes, which would be separated much like the ground planes. Digital power of +3.3V would lie directly above the digital ground plane, and the +5V and -5V analog planes would lie above the analog ground plane, but be shaped as obscure polygons so that all

pins that required ±5V would receive them in the appropriate spot. The bottom layer would consist of any additional components and routing that could not fit on the top layer efficiently.

When placing our components, our strategy separate analog and digital circuits to isolate the noise of the digital from interfering with the analog (much like our power and ground planes.) However, two of our components are mixed signal components (our DAC and ADC). Therefore, our strategy was to place these components at the intersection where the analog and digital ground/power planes meet, as shown below in Figure 15.

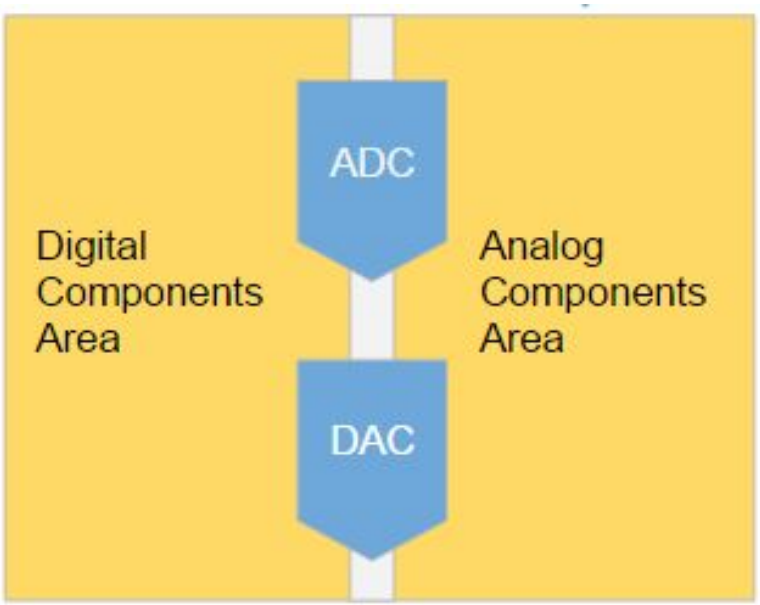

Figure 15: A visual representation of our component placement strategy for our PCB.

Below in Figure 16, we can see that we implemented the strategies listed above into our design. We separated our analog and digital ground planes, and merged them together with a system star ground where the two planes met. We also separated our analog and digital components with our digital components being on the bottom of Figure 16 and our analog components being on the top. In addition, you can see that our power planes were strategically separated, where our -5V analog plane being a unique polygon shape that looks like a backwards “F” embedded in the +5V analog plane.

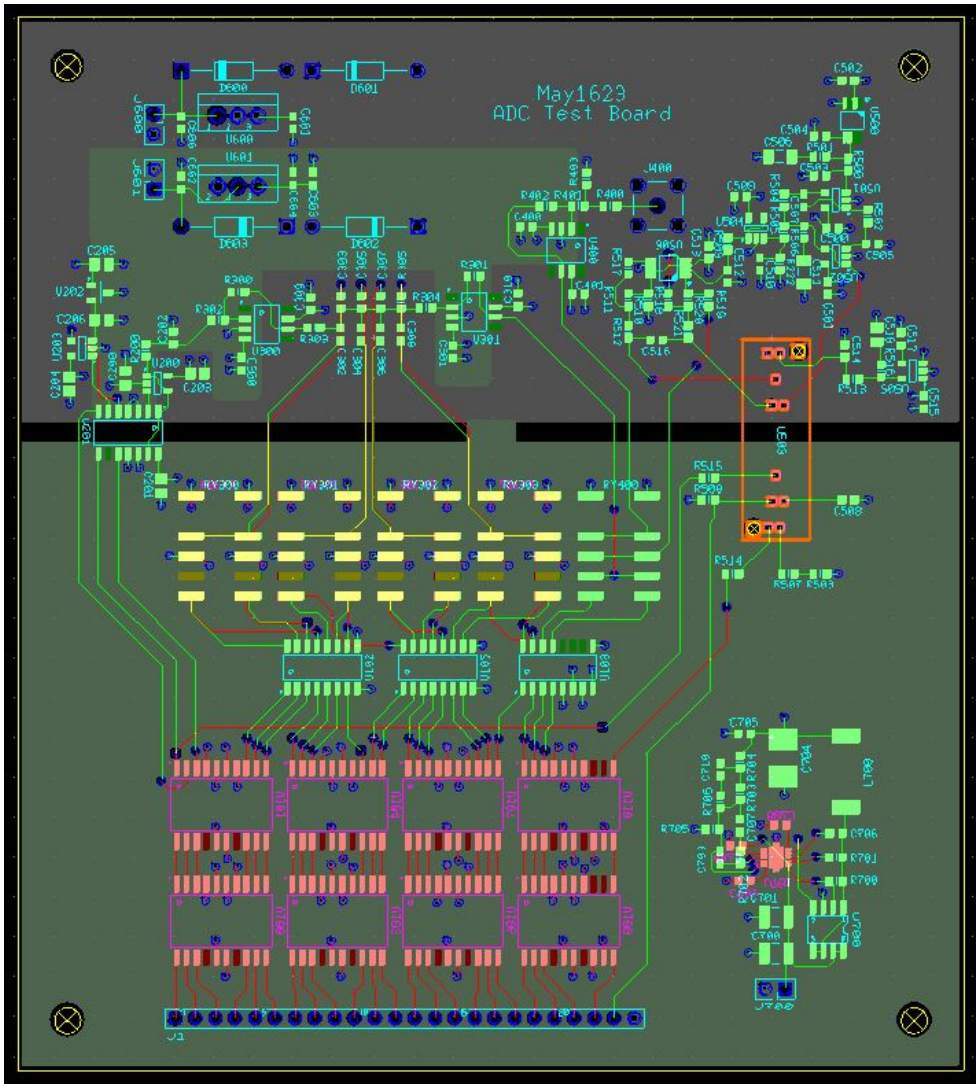

Figure 16: The actual PCB layout of our ADC Test Board.

Test Procedures and Results

Hardware Test Plan

Below is our test plan for testing the hardware of the ADC Test Board:

Step 1
Power supply: LDOs and Buck converter

1.1 Connect LDO (Analog) with 9V supply, check the output voltage, ±5V.

1.2 Connect Buck Converter (Digital) with 9V supply, check the output voltage, + 3.3V.

Step 2
DAC

2.1 Check ref voltage: 2.5V.

2.2 Give a input code and check the output voltage.
For input code 1111 1111 1111 1111, the output voltage should equal to +Vref, 2.5V.
For input code 1000 0000 0000 0000, the output voltage should equal to 0V.
For input code 0000 0000 0000 0000, the output voltage should equal to -Vref, -2.5V.

Step 3
Digital Buffer-Drivers

3.1 Measure the signal integrity after the two inverters (buffer), ensure it has the same logic value that we sent.

3.2 Measure the signal integrity after the relay driver.

3.3 Ensure that the Relay Switches are working as intended.

Step 4
Filter

4.1 Extend the input pin on the R303, and connect the wire with function generator, and apply an AC signal with the function generator. (Input +/- 2.5Vac)

4.2. Configure relays to the intended corner frequencies, setup filters.

4.3 Measure the output signal. Should expect a sine wave of +/- 4.3V, and right corner frequency from RR/ RC.

Step 5
ADC

5.1 Check ref voltage (pin 1 of ADC), 4.5V.

5.2 Give 1V input voltage, Check the input driver gain equal to 1.

5.3 Check the input drive common mode voltage be Vref/2, 2.25V.

5.4 Check DVDD (pin 10 of the ADC), 3.3V Digital Voltage.

5.5 Check AVDD (pin 2 of the ADC), 3.3V Analog Voltage.

5.6 Feed DC signal to the ADC, and check the output digital code.

Use input constant voltage 4.5V, check the output digital code, should be 1FFFF, which is 01 1111 1111 1111 1111 in binary.

Use input constant voltage 0V, check the output digital code, should be 00000, which is 00 0000 0000 0000 0000 in binary or 3FFFF, 11 1111 1111 1111 1111.

Use input constant voltage -4.5V, check the output digital code, should be 20001, which is 10 0000 0000 0000 0000 in binary.

All of the following test plan was carried out and worked, except for steps 2.2 and 5.6. For step 2.2 we received very strange results seen below in Table 2.

Input Code	Measured Output Voltage	Expected Output Voltage
1111 1111 1111 1111	34mV	2.5V (+Vref)
1000 0000 0000 0000	17mV	0
0000 0000 0000 0000	0mV	-2.5V (-Vref)

Table 2: Our Test Results for Step 2.2 (DAC Testing)

We believe that the source of this error is that our DAC chip is fried, for all of the pins of the DAC are at the appropriate voltage levels and receiving the appropriate digital signals. At the time of writing this report, we have a new DAC8831 ordered, but it has not arrived yet.

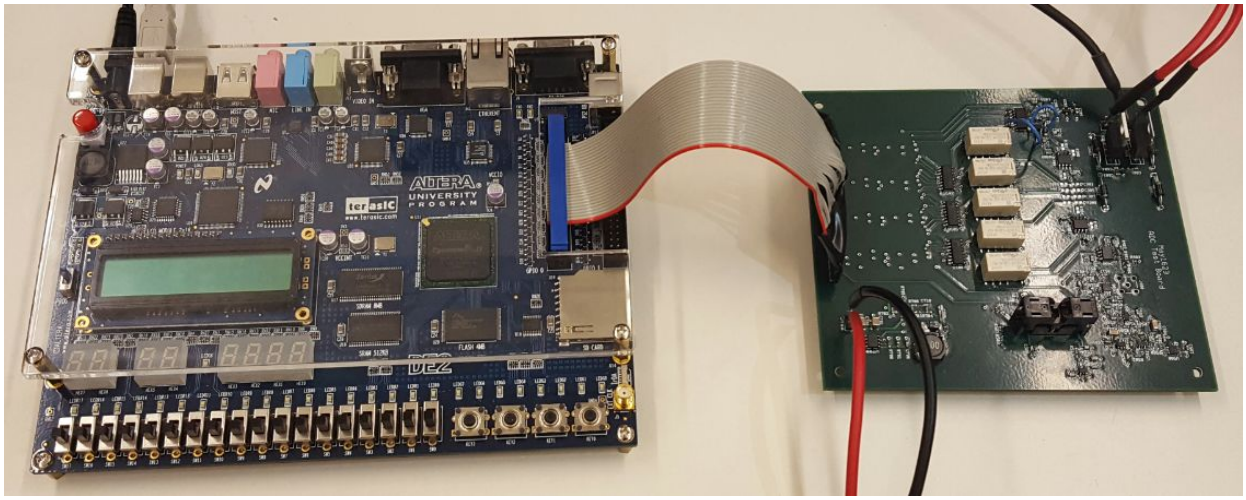

Figure 17: The Final Setup with our FPGA on the left and our ADC Test Board on the right.

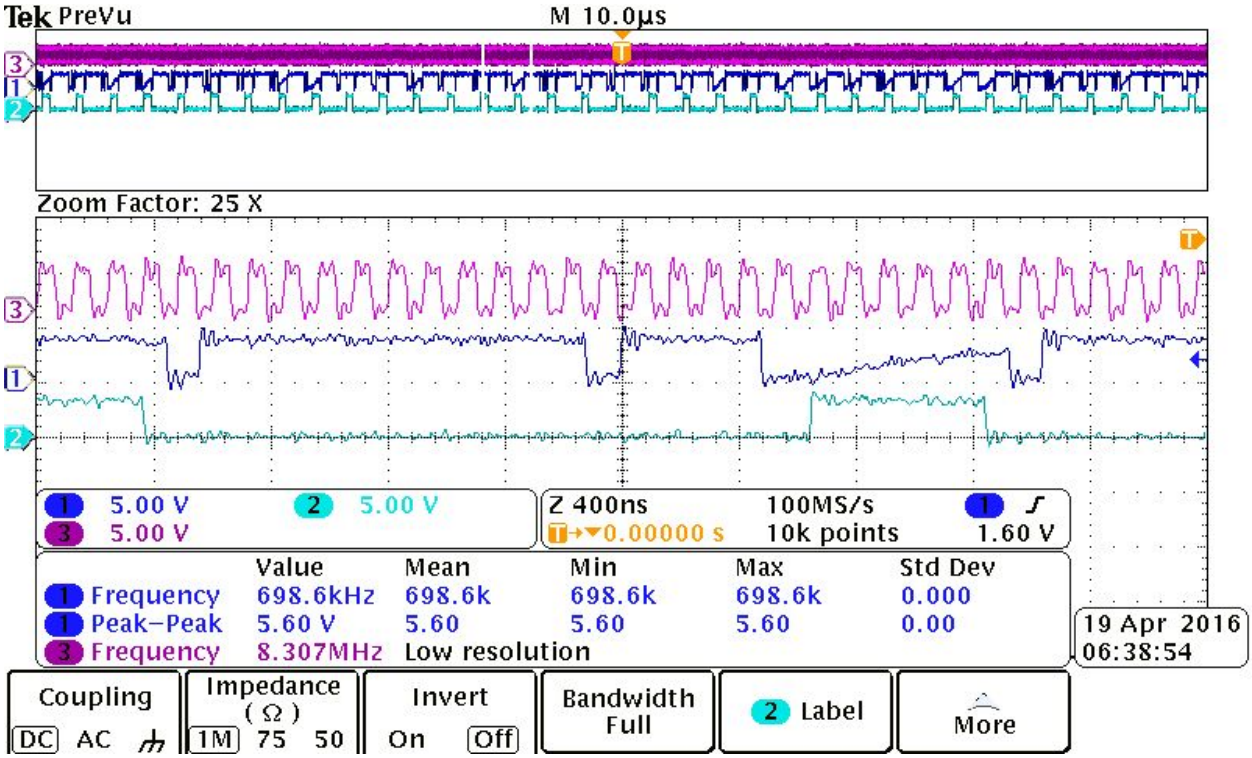

Figure 18: ADC output code for 0V input.

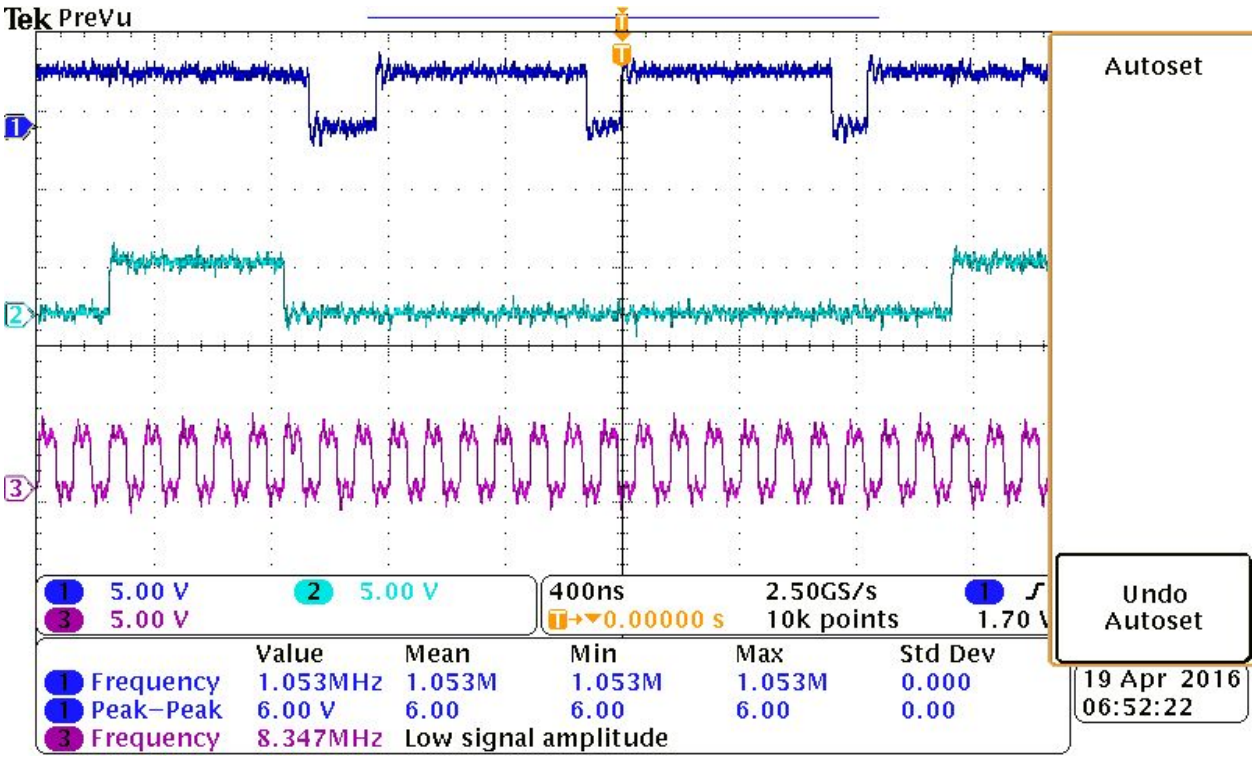

Figure 19: ADC output code for 4.5V input.

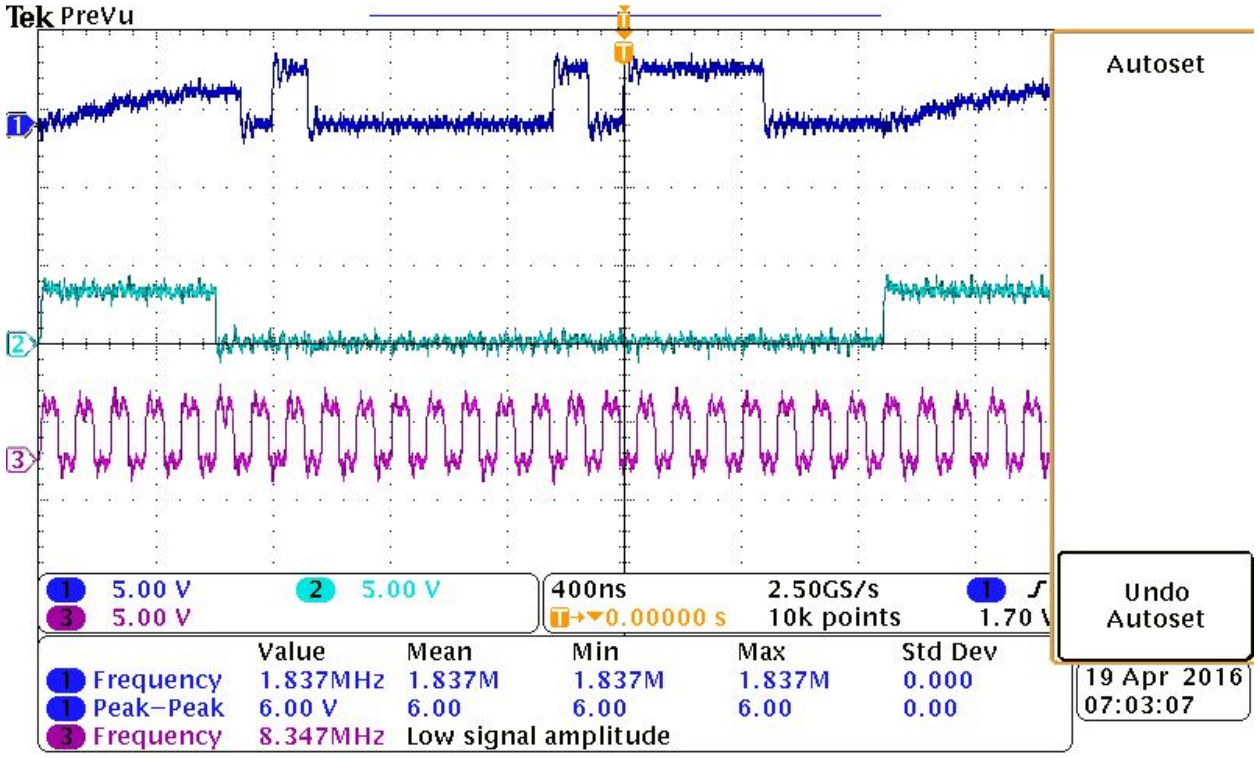

Figure 20: ADC output code for -4.5V Intput.

Input Voltage	Measured Output Code	Expected Code
0	11 1111 1111 1011 1100	11 1111 1111 1111 1111
4.5V	01 1111 1011 1111 0111	01 1111 1111 1111 1111
-4.5V	10 0000 0010 1111 0000	10 0000 0000 0000 0000

Table 3: Table ADC measured output codes v.s. expected codes.

With regards to Step 5.6 in our test plan, we did not fully achieve our expected results. This can be seen above in Figures 18-20 as well as Table 3. From Figure 18-20, the dark blue wave represents the ADC output code, the cyan wave represents the ADC chip select, and the purple wave represents the clock signal. From the datasheet, the MSB of the output code supposed to be carried out from the falling edge of the chip select, and the rest bits are carried out subsequently on the falling edge of the clock. But from the oscilloscope we can see there is one clock delay on the output code wave. However, if we account for this we get the suspected code, but it really isn't realistic in an automatic way when we need high speed sampling. We suspect that this is a software issue in our code with the FPGA.

Simulated Test Results

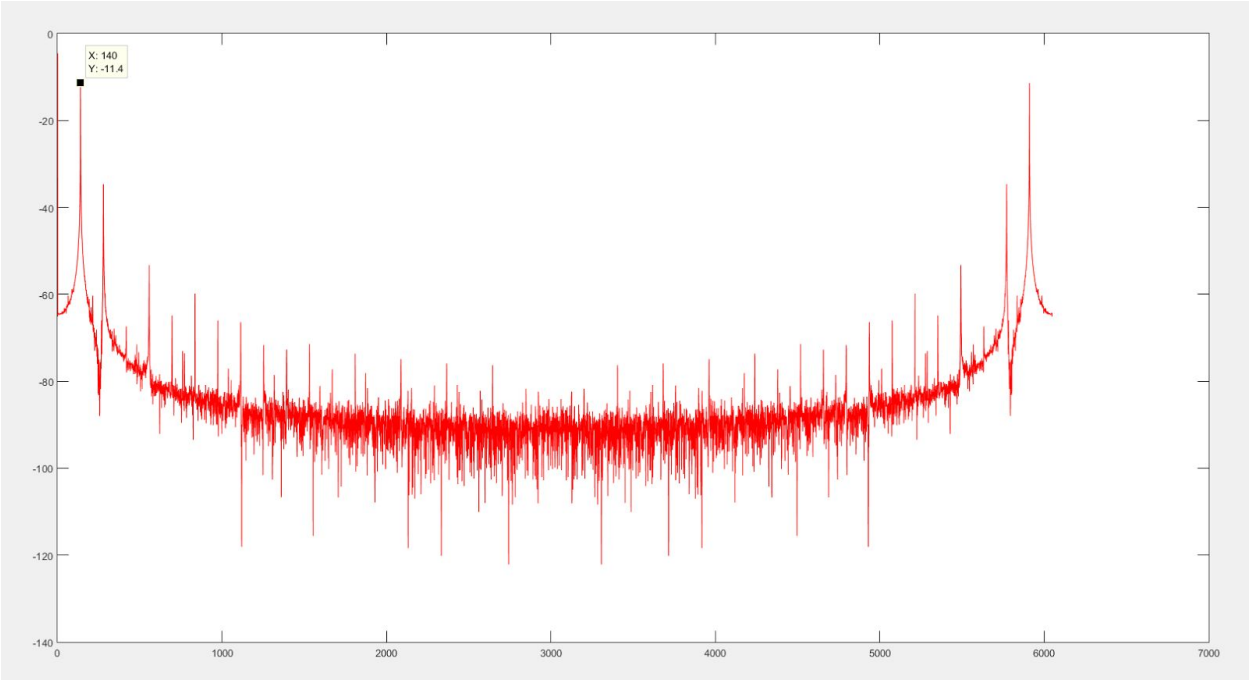

Figure 21: Arduino Due Spectral test with Frequency = 127Hz

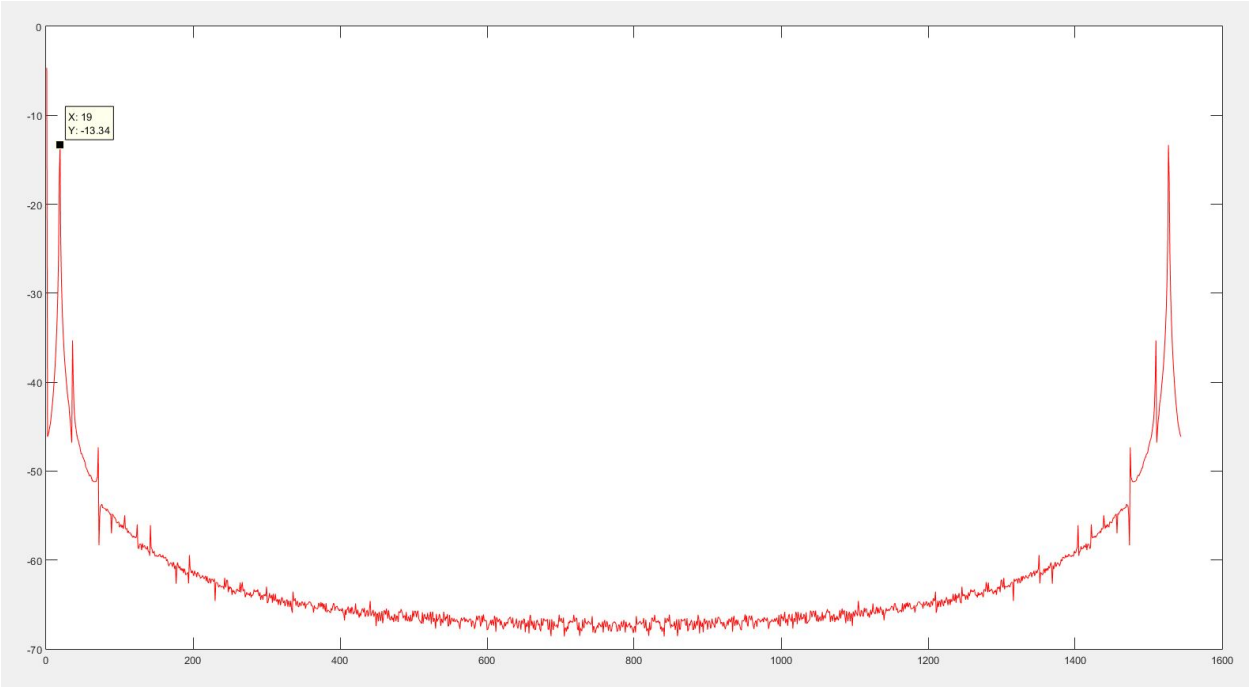

Figure 22: Arduino Due Spectral test with Frequency = 63Hz

The data for the Figure 21 and 22 were not captured on our PCB, however, it is a overview of what the spectral test of an ADC looks like. The spectrums were captured on the Arduino Due using a signal

generator that we had in lab. The Arduino Due contains an internal ADC in its microprocessor with the sampling frequency of 5.5kHz. These Figures really do not have anything to do with our project, they are simply example for what our spectral test results should look like. We labeled the peak signal frequency in each plot, and it matched the signals that we fed it, and observed the harmonics in each graph.

Future Work

Unfortunately, we were unable to achieve successful spectral test results at this point in time with the project. Therefore, next semester a new senior design team will take over what we had started this semester. Reflecting back, our hardware is nearly functional, there are just some minor bugs with our ADC and DAC, for the rest of the hardware checks out (Filter/Relays/Inverters). This new senior design team will continue where we left off, review our hardware and FPGA code (making improvements where they see fit), and continue to work towards getting successful spectral test results. They could also implement a GUI, so that the entire process is much more friendly to the average user. This new senior design group will have a much greater advantage than we had in being successful, due to all of this valuable data that we are leaving behind for them. This concludes our Design Document.

Modified date: April 24th, 2016

List of Figures and Tables

Appendix I:
Operation Manual

1. Connect 9V batteries to PCB (ADC Test Board).
2. Connect all GPIO pins from FPGA to the PCB.
3. Upload the Verilog code into the FPGA.
4. Set the frequencies for the filters by setting the logic on the relay pins.
5. First use RR filter on the PCB, start DAC, by cycling the lookup table for the DAC, and also start the ADC.
6. When the ADC collect the 2^16 points of data from DAC, stop the operation on the FPGA.
7. The values captured from the ADC will be store in the FPGA memory, and use the altera control panel software to write these values into a txt file.
8. Repeat step 5 to 7 by using RC filters.
9. Upload the ADC captured values for RR and RC filters into the MATLAB and plot the frequency spectrum.
10. Using the Audio precision instrument, and repeat step 4 to 6 without using the filters, and make sure to set the Audio Precision Relay Switch to its appropriate logic value.
11. Upload the captured file into MATLAB and plot the spectrum.
12. Observe Results.

Appendix II:
Alternative Design

The original design plan was to use the arduino to control the PCB, but we changed to the FPGA later on for a couple of reasons:

1. Arduino doesn't support accurate timing control, for all delay functions are in microsecond delay. Since we want 50MHz delay, the clock period is 20ns for each clock. So by using FPGA, we will be able to have the accurate timing control.
2. The SPI function on the Arduino doesn't output the correct clock, which will cause the malfunction of the DAC and ADC. For example, the output clock from the arduino is only 8 clock cycles, however, DAC needs 16 clock cycles and ADC needs 18 clock cycles in order to get the correct results.
3. The Arduino doesn't have enough memory on the board. It has only 512KB memory, but the FPGA has up to 8MB memory that we can access, which is far superior.

Appendix III:
Other Considerations

One important thing that happened during the debugging process of the DAC, we discovered that we had accidently routed together two pins that should not have been routed together, which initially lead us to believe that this was our issue for our DAC not working. Since the PCB is semi-permanent, to correct this we needed to take an exacto knife to wire traces and slice them a bunch of times until they were no longer

connected. Then, we took wire from our lab kits and jerry rigged the connections in their proper orientation. Although this was an important step, this did not change our test results, which lead us to believe that our DAC was burnt.

All in all, senior design was an adventure, full of panic when things did not work and satisfaction when we successfully discovered a bug and cleared it. I know we will not be forgetting our experiences in this class anytime soon. Thank you for reading.

Appendix IV: Project Plan

*Listed below are sections of our project plan that were not already listed above in the design document.

Concept Sketch/Mockup

Design requirement

The DUT(device under test) we are going to use is ADS8881 from Texas Instrument. The standard method to test the ADC is using the audio precision instruments to generate the pure sine wave and feed this sine wave into the ADC, but the cost of generating a pure sine wave is very high, because the audio precision is very expensive. So the alternative approach to test the ADC is using the DAC to generate the relative low purity sine wave, and use the algorithm to correct the distortions that comes from the DAC's sine wave.

Assessment of Proposed solution

Our proposed solutions is using the DAC to generate the sine wave and test the ADC. By doing this, the advantage is that it is low cost, relaxes the testing requirements, and in the end is high accuracy. Cost is a main issue in terms of project budget, an audio precision instrument usually worth 20000 dollars, but a DAC only cost 10 to 30 dollars. Even though the audio precision instrument will generate a much purer sine wave than DAC, the inaccuracies caused by DAC will be fixed by algorithm. In terms of testing requirement, the inaccuracies are introduced by DAC, but we will still get the accurate testing results, so the testing requirements are related. Based on our design references (Dr. Chen's master student Ben Magstadt's Thesis) the test results made by DAC will be very close to the results made by audio precision instruments.

Validation and Acceptance Test

Most of the functional blocks will be tested individually in the software simulation, and after all of them pass the design requirements, the PCB will be made, and the further testing will be done on the PCB. The software will be modeled in ModelSim and measured on an Oscilloscope.

System Description

Content

In this project, we are asking to design a circuit board to test the accurate spectral. The typical accurate spectral testing is very choosy. It requires a very pure sine wave which can be very difficult to generate when the under-test ADC is accurate. Our project is to use the non-standard way to test the ADC. This

means we can feed the nonlinear (impure) sine wave to the ADC. This method will significantly reduce the cost of ADC testing.

The ADC we choose in this project is the ADS8881 from Texas Instruments, it's a 18-bit differential input ADC, with maximum sample rate at 1MSPS. Our goal is to design a circuit board which can support the ADS8881, and allow the user to plug different ADS8881 chips and test the quality of them.

Technical approach

In order to be able to test the ADC with nonlinear input, we need to feed the sine wave to two different filters separately. This will allowed us to distinguish the noise from input signal and the noise from the ADC.

Process details

We need to design a circuit to support our ADS8881, with the power to drive the ADC. We will do this using a set of batteries, which as a result will help us reduce the background noise. For the input of the ADC, we will use the DAC to generate the sine wave. The sine wave THD (total harmonic distortion) from this method is typically around -50dB to -80dB, depends on which DAC we use. Nevertheless, it is larger than the pure sine wave THD required in standard test. The digital input to DAC is from the memory located on the FPGA. We also want to share the clock signal, generated from the FPGA) between the DAC and ADC to sure our signal is coherent.

Operating Environment

The overall system is broken into three different environments, the PCB which we are going to fabricate, the FPGA which we will utilize to communicate with the system, and the PC GUI which will contain the software spectral test algorithms. The PCB will contain the everything between the ADC and the DAC, which can be seen in gold in Figure 1. The FPGA will contain the memory to which we program the SPI communication, which can be seen in green in Figure 1. We will be using Verilog with the FPGA's interface software to do this. From here, the test results will be extracted from the FPGA and be sent to our PC, which will pipe into MATLAB to implement the algorithms. After this happens the project goal is met for the test board.

Functional Requirements

The main functional requirement of this test board is to deliver accurate spectral test using the algorithms devised in Ben's Thesis. These spectral test results must be comparable to spectral test results from Audio Precision equipment for this alternate method to be considered valid.

An alternate functional requirement is the speed of the test. The test should take be high speed and completely controllable by the user through the PC. No physical switching should be needed on the board. All switching between filters must be done digitally.

Non-Functional Requirements

The project is titled Low Cost High Accuracy Spectral Test System, with emphasis on the low cost,

therefore this is one of the non-functional requirements. Below is our cost analysis:

Budget: $500
Anticipated Costs:
PCB Manufacture: $66
ADC Socket: $11
FPGA: Free from ISU Parts Shop (rental)
Electrical Components: $247.80
Total: $324.80

Below is our Bill of Materials for our Electrical Components for ADC Test Board:

Quantity	Part Description	Digikey Part Number	Unit Cost	Total Cost
				$ -
	Resistors, 1% unless specified			$ -
3	0 Res 0603	RMCF0603ZT0R00CT-ND	$ 0.10	$ 0.30
3	0.22 Res 0603	P17462CT-ND	$ 0.40	$ 1.20
5	10 Res 0603	RMCF0603FT10R0CT-ND	$ 0.10	$ 0.50
3	47 Res 0603	RMCF0603FT47R0CT-ND	$ 0.10	$ 0.30
1	100 Res 0603	RMCF0603FT100RCT-ND	$ 0.10	$ 0.10
1	150 Res 0603	RMCF0603FT150RCT-ND	$ 0.10	$ 0.10
4	1k Res 0603 0.1%	A110108CT-ND	$ 0.25	$ 1.00
3	1k Res 0603	RMCF0603FG1K00CT-ND	$ 0.10	$ 0.30
1	2.15k Res 0603	RMCF0603FT2K15CT-ND	$ 0.10	$ 0.10
1	4.99k Res 0603	RMCF0603FT4K99CT-ND	$ 0.10	$ 0.10
6	10k Res 0603	RMCF0603FT10K0CT-ND	$ 0.10	$ 0.60
1	12.1k Res 0603	RMCF0603FT12K1CT-ND	$ 0.10	$ 0.10
5	20k Res 0603	RMCF0603FT20K0CT-ND	$ 0.10	$ 0.50
1	39k Res 0603	RMCF0603FT39K0CT-ND	$ 0.10	$ 0.10

1	63.4k Res 0603	RMCF0603FT63K4CT-ND	$ 0.10	$ 0.10
1	78.7k Res 0603	RMCF0603FT78K7CT-ND	$ 0.10	$ 0.10
				$ -
	Signal Capacitors 5%			$ -
1	100pF C0G Cap 0603	490-1427-1-ND	$ 0.10	$ 0.10
1	510pF Cap 0603	490-1444-1-ND	$ 0.15	$ 0.15
1	620pF Cap 0603	490-1446-1-ND	$ 0.15	$ 0.15
1	750pF Cap 0603	490-1448-1-ND	$ 0.16	$ 0.16
2	1000p C0G Cap 0603	490-1451-1-ND	$ 0.10	$ 0.20
1	1.5nF Cap 0603	490-1455-1-ND	$ 0.11	$ 0.11
1	2.7nF Cap 0603	490-3283-1-ND	$ 0.15	$ 0.15
2	10000pF NP0 Cap 0603	490-9666-1-ND	$ 0.18	$ 0.36
				$ -
	Power Capacitors 10%			$ -
1	100pF X7R Cap 0603	1276-1909-1-ND	$ 0.10	$ 0.10
1	3300pF X7R Cap 0603	490-1503-1-ND	$ 0.10	$ 0.10
2	0.01uF X7R Cap 0603	490-1512-1-ND	$ 0.10	$ 0.20
15	0.1uF Cap 0603	490-1532-1-ND	$ 0.10	$ 1.50
4	0.1uF X7R Cap 0805	490-1673-1-ND	$ 0.10	$ 0.40
1	0.33uF Cap 0603	490-3294-1-ND	$ 0.13	$ 0.13
1	0.47uF X7R Cap 0805	490-3328-1-ND	$ 0.30	$ 0.30
1	2.2uF X5R Cap 0805	490-3334-1-ND	$ 0.24	$ 0.24
11	1uF X7R Cap 0603	490-3900-1-ND	$ 0.10	$ 1.10
1	2.2uF Cap 0603	490-12324-1-ND	$ 0.17	$ 0.17
1	4.7uF X5R Cap 0603	490-7203-1-ND	$ 0.36	$ 0.36
1	10uF X7R Cap 0805	490-6477-1-ND	$ 0.10	$ 0.10
2	10uF X7R Cap 1206	490-6518-1-ND	$ 0.83	$ 1.66
2	22uF X7R Cap 1210	490-4524-1-ND	$ 1.01	$ 2.02
1	100uF Capacitor 0.138 x 0.110 in	P16248CT-ND	$ 1.59	$ 1.59
				$ -
	Inductors			$ -
1	10uH Ind 0.402 x 0.394 in	445-3578-1-ND	$ 1.60	$ 1.60
				$ -
	ICs			$ -
2	IC ADS8881	296-37253-5-ND	$ 38.13	$ 76.26
1	IC REF5045	296-24504-1-ND	$ 3.81	$ 3.81
1	IC OPA333	296-26269-1-ND	$ 2.60	$ 2.60
1	IC THS4281	296-39224-1-ND	$ 3.02	$ 3.02

1	IC OPA330	296-37699-1-ND	$ 1.81	$ 1.81
1	IC TPS78833	296-12380-1-ND	$ 1.16	$ 1.16
1	IC THS4521	296-35973-1-ND	$ 2.97	$ 2.97
1	IC OPA353	296-26278-1-ND	$ 2.89	$ 2.89
1	IC REF3025	296-26322-1-ND	$ 1.64	$ 1.64
1	IC DAC8831	296-18030-5-ND	$ 14.18	$ 14.18
1	IC OPA735	296-41480-1-ND	$ 3.25	$ 3.25
8	IC 74AC11004DW	296-4152-1-ND	$ 2.28	$ 18.24
3	IC DRV777	296-35584-1-ND	$ 0.58	$ 1.74
1	IC UA7805	296-39515-5-ND	$ 0.56	$ 0.56
1	IC UA7905	296-17077-ND	$ 0.79	$ 0.79
1	IC MOSFET FDS6910	FDS6910CT-ND	$ 1.15	$ 1.15
1	IC TPS40193	296-21677-1-ND	$ 2.74	$ 2.74
3	IC OPA211	296-22634-1-ND	$ 7.61	$ 22.83
9	IC 4.5V DPDT Relay	255-2348-5-ND	$ 5.56	$ 50.04
				$ -
	Misc.			$ -
4	1N4001 Diode	1N4001-TPMSCT-ND	$ 0.11	$ 0.44
1	SMA Connector	WM5543-ND	$ 2.99	$ 2.99
4	Nylon M3 Standoff 20mm	36-25514-ND	$ 0.79	$ 3.16
4	Nylon M3 Screw	36-29341-ND	$ 0.20	$ 0.80
3	1x2 Male Hdr Pins	S1011EC-02-ND	$ 0.16	$ 0.48
1	1x24 Male Hdr Pins	S1011EC-24-ND	$ 0.53	$ 0.53
3	9V Battery Strap	36-84-4-ND	$ 0.71	$ 2.13
3	9V Battery	N145-ND	$ 2.38	$ 7.14
	Total			$ 247.80

Market/Literature Survey

As stated earlier, the Iowa State Graduate Student Benjamin Magstadt based his graduate thesis around this idea of relaxing the specifications needed for spectral testing. We are using his research and implementing it into a test board for one specific kind of ADC. The alternative, which we will be comparing this to, is the traditional spectral testing which involves Audio Precision equipment. The technology for spectral testing exists, we are just trying to implement an alternative, and compare our results to the traditional standard.

Deliverables

The deliverables we are going to give:

Semester 1: simulation results, design schematics, functional PCB
Semester 2: verilog codes, proof of FPGA interfacing with PCB, output data from ADC, data analysis results from MATLAB

Work Breakdown Structure
Project Schedule

9/2015 - 10/2015 - Research/Component Selection/Methods
11/2015 - 12/2015 - First Draft Schematic/Simulations
1/2016 - 3/2016 - Hardware Redesign/Correct Semester 1 Errors/Begin Software
3/2016 - 4/2016 - PCB Manufacture/Testing/Documentation

Risks/Feasibility Assessment
In order to replace the traditional Spectral Test System, we are utilizing the research of Benjamin Magstadt who proposed that three algorithms with the right system would provide comparable results to the very expensive Audio Precision Instruments. In his thesis, he achieved a proof of concept, so if we follow his algorithms the project should be feasible. We will be using his design and approach as a strong reference as we incorporate our low cost high-accuracy spectral test system.

Conclusion
The goal of the project is to develop, test, and document a prototype for the Low Cost High Accuracy Spectral Test System for the TI ADS8881. Here, we will created hardware in the form of a PCB for the system, and will create and incorporate functional Matlab programs of Ben's algorithms to perform the Spectral test. Our data will be visible on a PC which is directly controlling the PCB board. The device under test will simply be inserted into a socket, for easy insertion and removal for a large number of ADS8881s.

Appendix V:
Code
FPGA Code

```
`timescale 1ns/100ps;
module dac_adc_testing(iCLK, oDAC_CS_N, oDAC_SDI, oClock, oADC_CS_N, iADC_DOUT);

	input iCLK;		//This is 50MHz clock from FPGA
	output  oClock;			//FPGA output clock to DAC and ADC, same for Clock.
	output [1:0]oDAC_SDI;
	output oDAC_CS_N;
```

```
        output oADC_CS_N;
        input  [1:0]iADC_DOUT;

        reg Clock;
        reg [4:0]clockcounter;
//DAC variables
        reg DAC_CS_N;
        reg [1:0] SDI;
        reg [4:0] DAC_counter;          //bit position of the 16 bits DAC
        reg [15:0] waveforms [0:2];     //[0:2] is just for testing purpose, it will be changed to
2^16-1 after the memory reading is implemented.  ***
        reg [15:0] counter;                     //total elements in the lookup table (for DAC use),
will be changed to accomodate 2^16 elements. ***
        reg DAC_done;
        reg [9:0] i;
        reg dac_ready;          // This flag is used for wait for a certain amount of time

//ADC variables
        reg [17:0] ADCcode [0:2];       // This is code read from ADC, the size will be changed to
2^18-1 later ***
        reg ADC_CS_N,ADC_done;
        reg [3:0] indexcounter;         // Will be changed to count 2^16 codes ***
        reg [4:0] ADC_counter;          // bit position counter for ADC generated codes.
        reg first_N;
        reg converting_done;                    // This flag is rised when ADC is done with
converting. About wait for 600ns.
        reg [2:0]j;   // temp counter for ADC_CS_N initial waitting time
        reg ADC_init_wait;
        reg [4:0]k;

        integer DAC_code_temp[0:65535];

        assign oClock = Clock;
        assign oDAC_CS_N = DAC_CS_N;
        assign oDAC_SDI = SDI;
        assign oADC_CS_N = ADC_CS_N;

//initial conditions
        initial begin
        Clock =0;
        clockcounter =0;
```

```
        waveforms[0] = 16'b1010100100000001;
//      waveforms[1] = 16'b1000010000100000;
//      waveforms[2] = 16'b1110111101111110;
        i = 0;
        dac_ready =0;

//set DAC initial conditions
        DAC_CS_N = 1;
        DAC_done = 0;
        SDI = 0;
        DAC_counter = 0;
        counter = 0;
//set ADC initial conditions
        first_N = 0;
        ADC_counter = 0;
        ADC_done = 0;
        indexcounter = 0;
        ADC_CS_N = 0;
        j = 0;
        ADC_init_wait =0;
        k = 0;
        converting_done = 0;
        end

/*
* This block is generating the output clock for DAC and ADC.
*/
        always@(posedge iCLK)begin
   if(clockcounter >= 2)begin
         clockcounter <= 0;
      Clock <= ~Clock;
   end
   else begin
         clockcounter <= clockcounter + 1;
   end

        end

/*
* In this block, the DAC is generating sine wave.
*/
        always@(negedge Clock)begin
   if(i != 2)begin
```

```
        i <= i +1;
    end
    else begin
        dac_ready <= 1;
    end
        if(DAC_counter <= 16 && DAC_done == 0 && dac_ready)begin  // change the
~DAC_done to DAC_CS_N, match the timing
                case(DAC_counter)
                0:begin
                DAC_CS_N <= 0;
                SDI <= waveforms[counter][15];
                $write("start%b",waveforms[counter][15]);
                end

                1:begin
                SDI <= waveforms[counter][14];
                $write("%b",waveforms[counter][14]);
                end

                2:begin
                SDI <= waveforms[counter][13];
                $write("%b",waveforms[counter][13]);
                end

                3:begin
                SDI <= waveforms[counter][12];
                $write("%b",waveforms[counter][12]);
                end

                4:begin
                SDI <= waveforms[counter][11];
                $write("%b",waveforms[counter][11]);
                end

                5:begin
                SDI <= waveforms[counter][10];
                $write("%b",waveforms[counter][10]);
                end

                6:begin
                SDI <= waveforms[counter][9];
                $write("%b",waveforms[counter][9]);
                end
```

```
7:begin
SDI <= waveforms[counter][8];
$write("%b",waveforms[counter][8]);
end

8:begin
SDI <= waveforms[counter][7];
$write("%b",waveforms[counter][7]);
end

9:begin
SDI <= waveforms[counter][6];
$write("%b",waveforms[counter][6]);
end

10:begin
SDI <= waveforms[counter][5];
$write("%b",waveforms[counter][5]);
end

11:begin
SDI <= waveforms[counter][4];
$write("%b",waveforms[counter][4]);
end

12:begin
SDI <= waveforms[counter][3];
$write("%b",waveforms[counter][3]);
end

13:begin
SDI <= waveforms[counter][2];
$write("%b",waveforms[counter][2]);
end

14:begin
SDI <= waveforms[counter][1];
$write("%b",waveforms[counter][1]);
end

15:begin
SDI <= waveforms[counter][0];
```

```
                $write("%bend",waveforms[counter][0]);
                end

                16:begin
                DAC_CS_N <= 1;
                DAC_done <= 1;
                end

                default: begin

                end
                endcase


                DAC_counter <= DAC_counter +1;

        end
        if(DAC_counter == 16 ) begin
                DAC_counter <= 0;
         DAC_done <=0;
        end
//      if(counter < 65535 && DAC_done)begin
//       counter <= counter +1;
//       DAC_done <= 0;
//      end

        end

/*
* In this block, the ADC is converting voltages into codes.
*/
        always@(posedge Clock)begin
   //////////////////////////
   if(j !=1)begin                   // This block is used for wait 50ns befor set the ADC_CS_N high.
One time use.
         j <= j +1;
   end
   else begin
         ADC_init_wait <= 1;
   end
   //////////////////////////

        if(first_N == 0 && ADC_init_wait ==1)begin
```

```
        ADC_CS_N <= 1;
        end

    ////////////////////////////
    if(ADC_CS_N ==1 && k != 4)begin          //This block is used for ADC_CS_N waits 600ns
after the ADC_CS_N has a rising eduge.
      k <= k+1;
    end
    if(ADC_CS_N ==1 && k == 4) begin
      converting_done<= 1;
      k <= 0;
    end
    ////////////////////////////

        if(ADC_CS_N == 1 && converting_done)begin
        ADC_CS_N <= 0;
        first_N <= 1;
    converting_done <= 0;
        end

        if(ADC_counter <= 18 && ADC_CS_N == 0 && first_N ==1 )begin
                case(ADC_counter)
                0:begin
                ADCcode[indexcounter][17] <=iADC_DOUT;
                end

                1:begin
                ADCcode[indexcounter][16] <=iADC_DOUT;
                end

                2:begin
                ADCcode[indexcounter][15] <=iADC_DOUT;
                end

                3:begin
                ADCcode[indexcounter][14] <=iADC_DOUT;
                end

                4:begin
                ADCcode[indexcounter][13] <=iADC_DOUT;
                end

                5:begin
```

```
ADCcode[indexcounter][12] <=iADC_DOUT;
end

6:begin
ADCcode[indexcounter][11] <=iADC_DOUT;
end

7:begin
ADCcode[indexcounter][10] <=iADC_DOUT;
end

8:begin
ADCcode[indexcounter][9] <=iADC_DOUT;
end

9:begin
ADCcode[indexcounter][8] <=iADC_DOUT;
end

10:begin
ADCcode[indexcounter][7] <=iADC_DOUT;
end

11:begin
ADCcode[indexcounter][6] <=iADC_DOUT;
end

12:begin
ADCcode[indexcounter][5] <=iADC_DOUT;
end

13:begin
ADCcode[indexcounter][4] <=iADC_DOUT;
end

14:begin
ADCcode[indexcounter][3] <=iADC_DOUT;
end

15:begin
ADCcode[indexcounter][2] <=iADC_DOUT;
end
```

```
                16:begin
                ADCcode[indexcounter][1] <=iADC_DOUT;
                end

                17:begin
                ADCcode[indexcounter][0] <=iADC_DOUT;
                end

                18:begin
                ADC_done <= 1;
                ADC_CS_N <= 1;
                end

                default: begin

                end
                endcase


                ADC_counter <= ADC_counter +1;

        end
        if(ADC_counter == 18 ) begin
                ADC_counter <= 0;
        end
        if(indexcounter < 2 && ADC_done)begin            // 2 will be changed to 2^18-1 ***
                indexcounter <= indexcounter +1;
                ADC_done <= 0;
        end
        end

/*
* The memory on the FPGA is being used in this block
*/



endmodule
```

MATLAB Code:

Sine wave lookup table with desired frequency

```
%%Equation for calculate J:(f_target)/(f_sampling)=(J_target)/(M);
%%M=2^16. In this case, J= ((20.3*10^3)/(2*10^6))*(2^12) = 41;

% The V_out = (V_ref*(D-2^15))/(2^15); D(i) is in the range of 328 - 65208; given the output voltage from
% V_out(min) = 2.5*(328-2^15)/(2^15)= -2.4749
% V_out(max) = 2.5*(65208-2^15)/(2^15) = 2.4749
% V_out is within 98.9990% of V_ref
%25 - 700hz
%81 -2k


k=0;
i=1;
for i=1:4096
   k=k+1;
   D(i)= round((1+2*0.99*sin(2*(249/4096)*k))*1024+1024);

end
plot(D)
csvwrite('16khz.csv',D);




function [Amp, Phase, Jmeasured, JinitMeasured] =
nonCoherentCorrection_V2(signal, Fsamp, t, numberOfHarmonics)

   signal = signal';
   iterations = 10;
   m = length(signal);
```

```
%FIRE METHOD BEGIN

[ftdataCoh cohSig nprd P A efft h1Hat2N Jint Jd] =
ncfft_7v_nNumHarmCorr(signal,length(signal),0,0,numberOfHarmonics);

JinitMeasured = Jint + Jd;

%FIRE METHOD END

%subtracting fundamental
r = signal - (A * cos(2 * pi * Fsamp * (Jint + Jd) / m * t + P));

%running closed form solution of least square for harmonics
D0 = zeros(m,2*(numberOfHarmonics)+1);
X = transpose(r);

for z = 1:(numberOfHarmonics)
   for y = 1:m
      D0(y,(z-1)*2+1) = cos(2*pi*Fsamp*(z)*(Jint+Jd)/m*t(y));
      D0(y,(z-1)*2+2) = sin(2*pi*Fsamp*(z)*(Jint+Jd)/m*t(y));
   end
end
D0(:, 2*(numberOfHarmonics)+1) = 1;

s0 = (transpose(D0) * D0) ^ -1 * (transpose(D0) * X);

for z = 1:(numberOfHarmonics)
   ah(z) = sqrt(s0((z-1)*2+1)^2 + s0((z-1)*2+2)^2);
   ph(z) = -atan(s0((z-1)*2+2) / s0((z-1)*2+1));
   if(s0((z-1)*2+1) < 0 && s0((z-1)*2+2) > 0 && ph(z) > 0)
      ph(z) = ph(z) - pi;
   elseif(s0((z-1)*2+1) < 0 && s0((z-1)*2+2) < 0 && ph(z) < 0)
      ph(z) = ph(z) + pi;
```

```
    end
  end

  %running least squares for the fundamental frequency with harmonics
  %subtracted
  r = signal;
  for b = 2:(numberOfHarmonics)
    r = r - ah(b) * cos(2*pi*Fsamp*(b)*(Jint+Jd)/m*t+ph(b));
  end

  %debug
%   r123 = r - (A * cos(2 * pi * Fsamp * (Jint + Jd) / m * t + P));;
%   Y = fft(r123, m) / m;
%   Ymag = 2 * abs(Y);
%   figure;
%   plot(20 * log10(Ymag),'r');
%
  %FIRE METHOD BEGIN

  [ftdataCoh cohSig nprd P A efft h1Hat2N Jint Jd] =
ncfft_7v_nNumHarmCorr(r,length(r),0,0,numberOfHarmonics);

  Jmeasured = Jint + Jd;

  %FIRE METHOD END

  %subtracting the updated fundamental
  r = signal - (A * cos(2 * pi * Fsamp * (Jint + Jd) / m * t + P));

  %running closed form solution of least square for harmonics
  D0 = zeros(m,2*(numberOfHarmonics)+1);
  X = transpose(r);
```

```
    for z = 1:(numberOfHarmonics)
        for y = 1:m
            D0(y,(z-1)*2+1) = cos(2*pi*Fsamp*(z)*(Jint+Jd)/m*t(y));
            D0(y,(z-1)*2+2) = sin(2*pi*Fsamp*(z)*(Jint+Jd)/m*t(y));
        end
    end
    D0(:, 2*(numberOfHarmonics)+1) = 1;

    s0 = (transpose(D0) * D0) ^ -1 * (transpose(D0) * X);

    for z = 1:(numberOfHarmonics)
        ah(z) = sqrt(s0((z-1)*2+1)^2 + s0((z-1)*2+2)^2);
        ph(z) = -atan(s0((z-1)*2+2) / s0((z-1)*2+1));
        if(s0((z-1)*2+1) < 0 && s0((z-1)*2+2) > 0 && ph(z) > 0)
            ph(z) = ph(z) - pi;
        elseif(s0((z-1)*2+1) < 0 && s0((z-1)*2+2) < 0 && ph(z) < 0)
            ph(z) = ph(z) + pi;
        end
    end

    Amp = [A, ah(2:numberOfHarmonics)];
    Phase = [P, ph(2:numberOfHarmonics)];

    Jmeasured = Jint + Jd;

end
```