Internet of Things Data Acquisition System (webDAQ)

Project Members:

Fausto Tommasi

Allen Burcham (facilitator)

Chris Lucas

Taylor Watson

Under Guidance from:

Dr. Gary Bledsoe and Dr. Kyle Mitchell

ECE 4800/BME 4960/BME 5950: Senior Design I

Saint Louis University

Parks College of Engineering, Aviation, and Technology

Thursday October 20, 2016

Keywords: Internet of Things (IoT), Data Acquisition System (DAQ), Amazon Web Services (AWS), do it yourself (DIY), single board computer, analog-to-digital converter (ADC), backend, session, amplifier, digital potentiometer, general purpose diode

Table of Contents

Executive Summary

This report is a description of the work done on an Internet of Things data acquisition system that continuously samples analog data and uploads the data to the internet. Data acquisition systems (DAQs) are used in many industries from infrastructure systems testing to education. Typical data acquisition systems gather an analog signal and output this data in a digital form that can be analyzed and used by an application. The problems with current systems are that the data can only be seen by one user at a time and the user needs to be physically close to the DAQ. The presented design will resolve these issues allowing for the data being captured by the DAQ to be sent to multiple, remote users that have access to the secured IoT backend.

Intro/Background

Data acquisition systems have myriad uses, including infrastructure systems testing and electrical hardware design, as well as educational purposes. One of the limitations of the current standards of data acquisition technology is the speed and availability of acquired data. Presently, such devices must be handled and maintained in person; they are only useful in the physical presence of a user. Any data gathered must then be handled manually and, if it must be distributed in any way, must be sent out manually as well. These limitations emerge from the necessitated connection of DAQ devices to on-site computer systems by physical media. Here presented is a design for a device which solves these problems through the use of Internet of Things (IoT) technology.

The data acquisition device designed for this project acquires data through a single channel and makes that data available to an Internet of Things platform that is accessible to any authorized user. This method eliminates the setbacks of DAQ systems by allowing any number of users access to real-time, continuous data from the device via the internet. It samples signals at voltages in the range of +/-5V (10V peak to peak) and at frequencies up to 50 kHz, then sends these signals to its IoT backend, which store the data and manage its distribution to subscribed users.

The presented design is a system that not only acquires data, but fully connects the acquisition device to the internet and thus to as many users as have submitted queries with proper authorization. This involves an IoT platform hosted by Amazon Web Services, which must be utilized to manage connections and device states, as well as data storage and distribution. Applications on both the DAQ and client ends must be able to communicate appropriately with each other over this IoT platform.

Market/Social/Ethical

There currently exists a market for Internet of Things devices due to the progress made in advancements of computer technology, however there is no DAQ product that allows for custom IoT applications. Current markets do not allow users to capture data and then create applications for it.

The IoT DAQ is a commercial product targeted toward an audience interested in building their own in a “Do It Yourself” approach. Consumers of the IoT DAQ will be able to connect it to any device that agrees with its input specifications. As a result, it is important that we as

designers exercise care with how data is transmitted across the web. Additionally, the consumer and user of this product is morally and ethically responsible for the actions that are performed with this device. They should be aware of the risks of transmitting their data. As the designers, we must guarantee that their data is secured during transmission as well.

The intention of this product is to facilitate its users with the transmission that is to occur after the data acquisition. In doing so, users will be able to spend their time creating applications that make use of the data rather than spending development time on work that is related to data transmission. This will allow fast and efficient development of data acquisition applications using an IoT framework that guarantees a data format and throughput.

The potential for this product would allow users of any ability, from hobbyist to professional, to quickly iterate over a design that requires some remote data acquisition that can be solved by IoT and build reliable applications without having the overhead of network communication and storage that is a key concern in IoT applications

Design Parameters

- The device will sample external, analog data and transmit the data to an endpoint through the internet.
- The application will download or stream the data for local use and will be able to command the device by setting its properties.
- The device will function properly as long as it …
    - has sufficient power
    - is connected to the internet

    - is used indoors in normal conditions
- The system will target classrooms and home enthusiasts (DIY market)
- The cost of the device will be fixed to the cost of the hardware, there will be no recurring cost for licensing.
- The cost of the IoT service will be dependent on the number of messages sent.
- The application will be open-source and free.
- The device and application will function as long as their requirements are met and the backend IoT service is available.
- The user will be expected to modify the voltage ranges of their signals to be within the operating specification of the device.

Technical Analysis

Our functional decomposition is outlined below in Figure 1. This diagram demonstrates the functionality of our product as a flow of connected functions, and divides these functions between the team members.

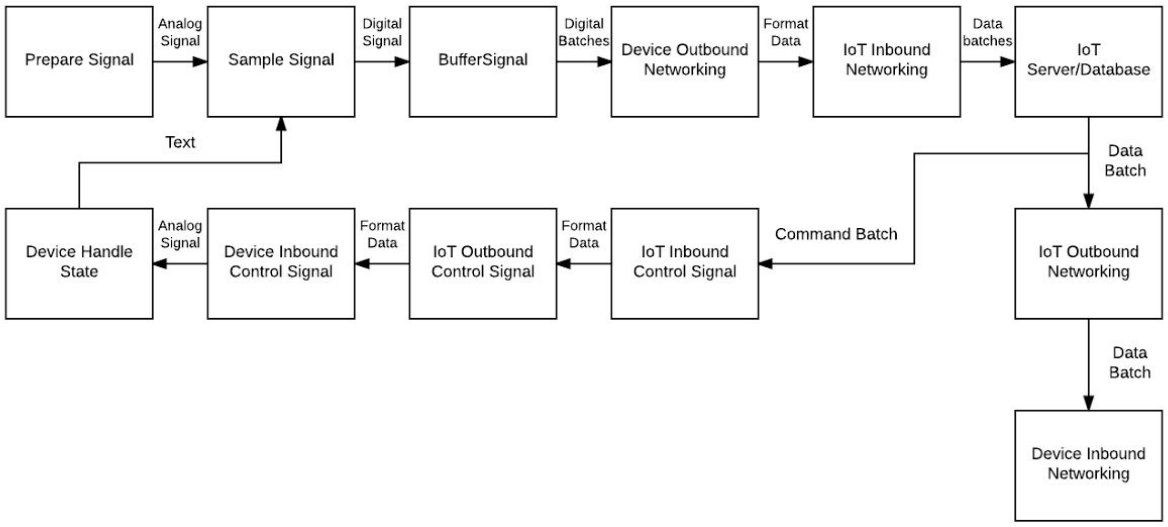

Figure 1

The design of the WebDAQ system is comprised of three basic elements: the IoT backend, a device to communicate with the backend, and an application to retrieve the data from the backend. There were many options considered when the IoT backend was chosen. This backend was then linked to the hardware device that actually acquires the data. This communication device samples analog data and transmits the data over the internet to our backend. The user is then able to retrieve this data via a user end application. Users are able to choose which sections of data they want to download and view.

The analog signal conditioning circuit is comprised of five stages. The first three stages are comprised of op-amps, digital potentiometers, and an array of different resistors. The fourth stage is for ADC protection purposes and the fifth stage is an inverting and DC offset stage.

The first stage is a high impedance rail to rail output unity gain amplification stage. The purpose of this first stage is to ensure that impedance matching will not be an issue based upon what types of sensors and devices the user connects to our device.

The second stage of this circuit is tunable gain amplification. The purpose of this stage is to attenuate a large signal or amplify a small signal before being sent into the ADC. Here another op-amp that is identical to stage one is being used along with two digital potentiometers. The output here will be attenuated appropriately, based on what the user enters in the client application, by adjusting the digital potentiometers programmatically and the output will be 180 degrees out of phase with the original user input.

The third stage is a first order low pass filter that obeys Nyquist filtering criteria for our ADC. The cutoff frequency is determined by what the user enters for their sampling frequency in the client application and a digital potentiometer will be set accordingly.

The fourth stage is a clipping stage to ensure the ADC will not be damaged. Two general purpose diodes are used here to clip the output voltage at 0.2V and 1.6V. This is enough to ensure the circuit will not be damaged before the ADC.

The fifth stage is the DC shifting and inverting stage and is comprised of resistors and the same op-amp as used in the first three stages. A DC shift of 0.9V is needed as a new reference for 0V since our ADC can only read from 0V to 1.8V and we are assuming zero mean input signals. This will allow for a full sine wave to be seen within this voltage range. Since stage two

inverted the signal to be 180 degrees out of phase with the original input, the signal needs to be inverted to be back in phase with the original input. The output of stage 5 will be the input to the ADC.

The single board computer is compromised of two main functions. The first will be a sampling function that reads raw data and sets the parameters of the conditioning circuit based on user input. The second will be a networking portion that will transmit the data over the internet to our backend as well as listen for user commands.

The sampling is done from the output of our signal conditioning circuit which provides a raw voltage between the range of 0V - 1.8V to the ADC. The ADC is the one that is built into the Beaglebone Black. This ADC is rated for 200k samples per second and can read positive voltages that are less than 1.8V. The ADC samples this raw voltage at a rate of 50kHz, or a rate specified by the user, and relates a time sample to this reading. Each voltage read is communicated to the software that is in charge of transmitting the data over the network. This networking software sends the raw ADC values, along with some other necessary information that will be used by the IoT backend. The networking component exists in the same project space that the back end software exists, thus facilitating the communication and object structure between the two. The purpose of this code is to perform the appropriate handshakes, as specified by the backend, and to transmit the data that is relevant to the user. In addition to this communication from the device to the user, the user is able to specify a sample rate and tweak some of the sampling parameters.

Using the same protocols, the backend communicates some commands to the single board computer. The networking software processes these commands and appropriately modifies

the resistances of particular elements of the conditioning circuit or modify the software sample rate when applicable. This way the user is able to customize the stream of data that is being recorded by the sampling circuit.

The AWS IoT service is able to work with AWS Lambda and DynamoDB by default. This functionality is important so that the low-frequency samples that pass through the IoT gateway are entered into the DynamoDB database. This is done with a built in IoT rule to insert data to the database.

Since there is one database and the potential for multiple simultaneous clients accessing the data, the client will have its own interface for managing configuration settings. Essentially, the client will ask the user which session they want to view, and if they want live data streamed. The client will then have the option to fetch data from a time window in the database and keep listening on the IoT topic for new data if streaming is selected. With this system the user has the ability to organize their data into named sessions and recall it either using the client or even AWS's developer console.

Time window data can be written to a file specified by the user. There is a built in option to view written data as a Plotly .html file, but the file can also be accessed by other programs on the system, like Matlab, which a user can utilize for any data processing they find necessary. The resulting system will retain device state between client and DAQ without complicating the cloud data storage model.

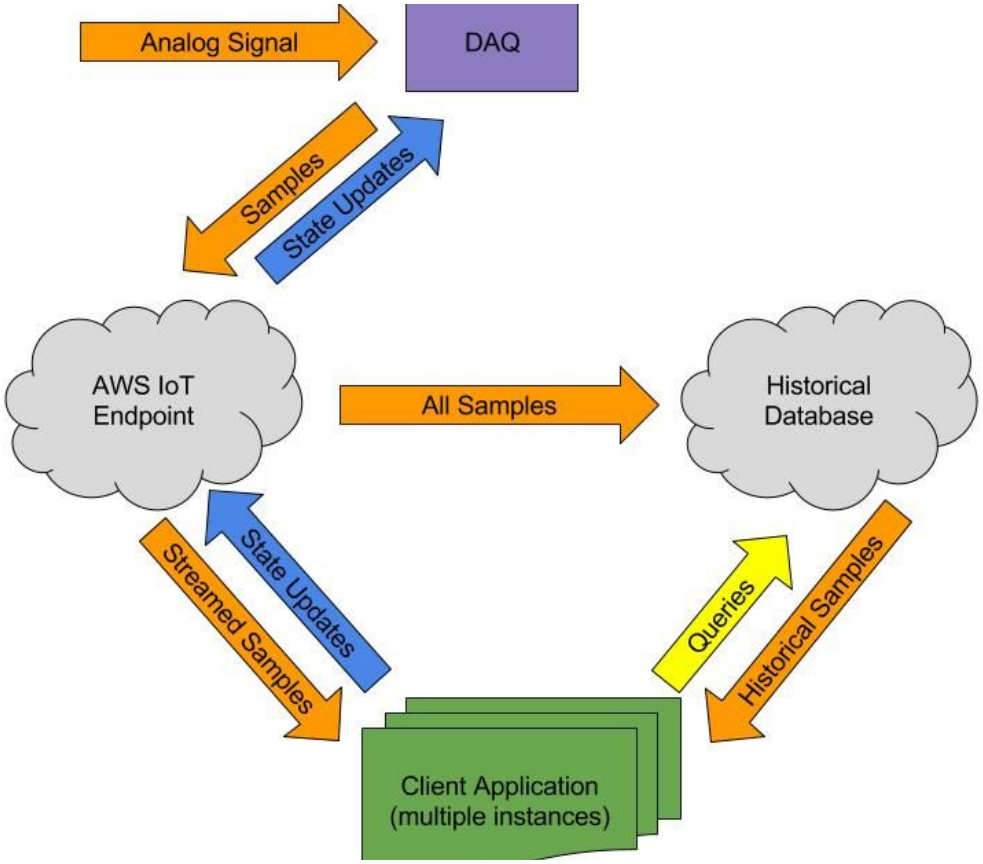

Figure 2

Figure 2 shows a simple system flowchart for IoT DAQ.

The analog data comes into the DAQ, is sampled, and is sent to the IoT endpoint. The IoT endpoint forwards the data to a DynamoDB table and creates a row for every batch of data. The data is also forwarded to any clients that are subscribed to the topic that the sample is published to, this is for real-time streaming. The client application can query the database to get a set of the historical data. The application can also update the state of the DAQ. The state change will allow for changing the power state, controlling different sampling sessions, and choosing the sampling frequency for a particular session.

Implementation Plan

Our design implementation consisted of three relatively concurrent sub-projects that were integrated into our final design. The first is the analog signal conditioning circuit which is primarily hardware and will be comprised of five stages. The first is be a high impedance rail to rail output unity gain amplification stage. In order to achieve this an op-amp was chosen that has a high slew rate, high input impedance, unity gain stable, rail to rail output, and proper voltage supply range. The non-inverting terminal is tied to the user input voltage and the output of the op-amp is tied back to the inverting terminal with a $1\,k\Omega$ resistor to prevent instability behavior (see Figure 3).

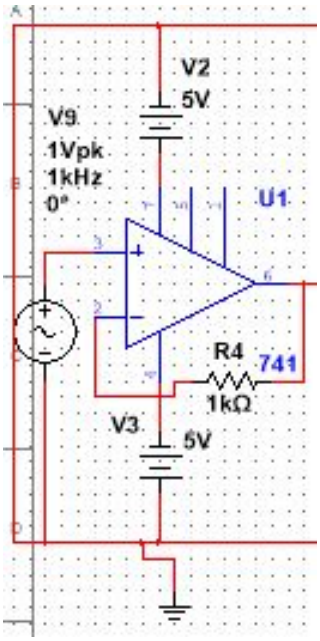

Figure 3: Stage 1

The second stage of this circuit is tunable gain amplification. Here another op-amp that is identical to stage one is being used along with two digital potentiometers. The non-inverting terminal is tied to ground and the inverting terminal is tied to two digital potentiometers, one digital potentiometer ($10\,k\Omega$ resistor for testing purposes) is coming from the output of stage one and going into the inverting terminal and the other digital potentiometer ($1\,k\Omega$ resistor for testing purposes) is used as a feedback from the output back to the inverting terminal (see Figure 4). The digital potentiometers used are each $10\,k\Omega$ on the Maxim DS3903 chip.

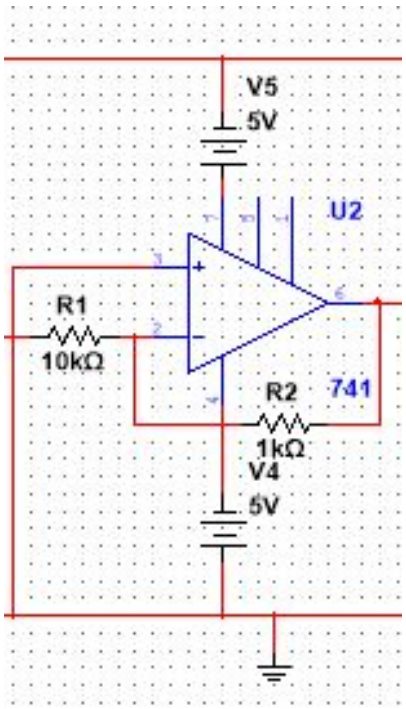

Figure 4: Stage 2

The third stage is a first order low pass filter that obeys Nyquist filtering criteria for our ADC. Another identical op-amp from stage one and stage two is used. The output from the second stage goes into another digital potentiometer ($3182\,\Omega$ resistor for testing purposes), with a 1nF capacitor in parallel connected to ground, and goes into the non-inverting terminal of the

op-amp. The digital potentiometer is the $90\,k\Omega$ potentiometer present on the Maxim DS3903 chip. The inverting terminal is again connected in a feedback manner exactly like stage one to ensure unity gain and free of instability behavior (see Figure 5).

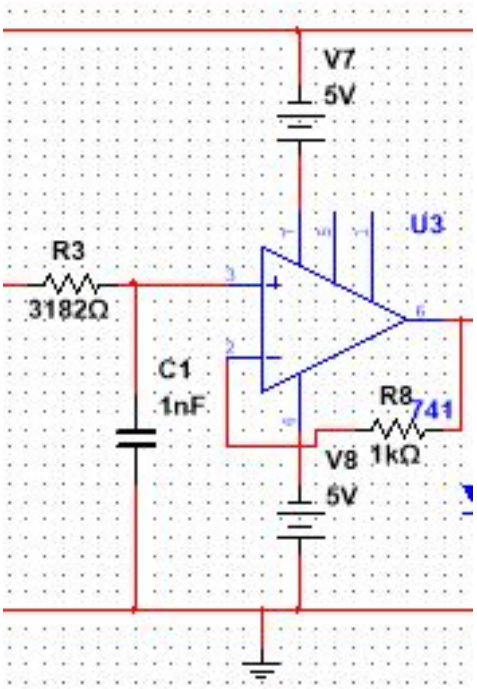

Figure 5: Stage 3

The fourth stage for the analog signal conditioning before we allow the output to go into the ADC is a clipping protection stage. This is comprised of two general purpose 1N4001 diodes (see Figure 6). This will clip the output at -0.7V and 0.7V. This will then ensure the signal is clamped slightly above 0V, at about 100-200mV for safety of the ADC, and at about 1.5-1.6V after the offset from stage 5.

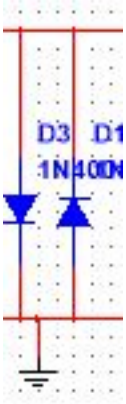

Figure 6: Stage 4

The fifth and final stage is the DC shifting and inverting stage. A DC shift of 0.9V is needed as a new reference for 0V since our ADC can only read from 0V to 1.8V and we are assuming zero mean input signals. This allows for a full sine wave to be seen within this voltage range. Since stage two inverted the signal to be 180 degrees out of phase with the original input, the signal needs to be inverted to be back in phase with the original input. This is achieved with another identical op-amp from the first three stages along with 3 $1\,k\Omega$ resistors and a $10\,k\Omega$ resistor. These resistors are used to maintain a gain of one and to divide down the supply voltage to achieve the DC offset of 0.9V. A 1nF capacitor is placed across the output to correct instability oscillation that occurs (see Figure 7). The output of this stage is the input to the ADC.

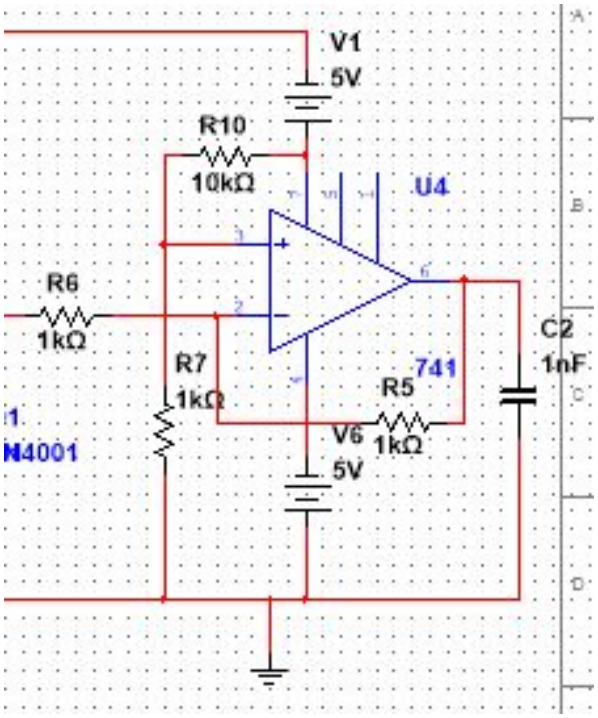

Figure 7: Stage 5

The overall circuit can be seen in Figure 8 below.

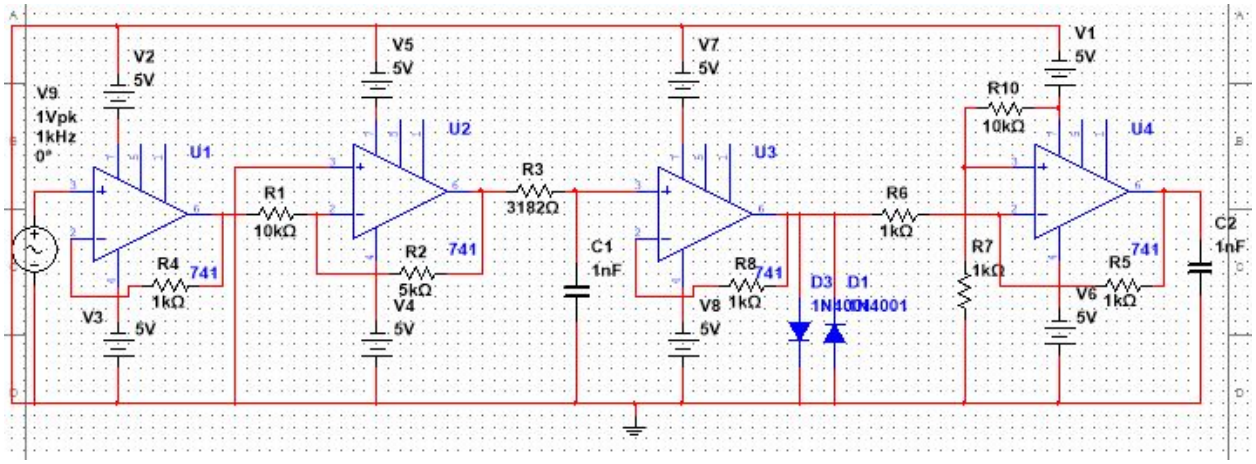

Figure 8: Overall Circuit

The single board computer implementation is comprised of two major pieces of software on the Beaglebone. The first is the ADC controller, which is C code that is in charge of pairing a

sample and a time stamp. The Beaglebone OS provides functionality that allows users to access the data readings from the integrated ADC. The ADC controller will make use of this functionality to take this raw reading, associate a timestamp, and pass this recorded data to the networking software. The choice to use C for this software implementation was mostly based on speed. The Beaglebone OS does not provide a software API to control the functionality with respect to the header pins. Since there is a constraint on 50kHz sample speed, it was important to select a framework that would be able to sample at a speed that was more than fast enough. To communicate with the networking software the ADC controller writes its results to a stream that is read later on by the networking code. The C code takes a command line argument for the frequency and then is able to control its sampling frequency based on this input.

The DaqPublisher is the second major piece of software that is in charge of taking the ADC reading and time stamps, and formatting them properly in order for them to be stored in the backend database. This publisher reads values from the previously mentioned file and transforms the data into a string that contains the correct information. It formats this data along with session and device information and passes it on to the AWS library code that is in charge of the distribution of the data. Additionally, functionality was implemented to both of these pieces of software to handle user commands. The DaqPublisher handles incoming requests and communicate with the ADC software using interprocess communication in order to specify sample speeds, cut off frequencies and input gain.

The AWS backend is configured to fulfill the requirements of the project. Two AWS services were required. First, the IoT service handles the communication of all data streams. Second, a DynamoDB (noSQL) database stores low-frequency sample data.

Upon initialization, the client and DAQ establish connections to the IoT gateway in the cloud. The devices actually use 5 concurrent connections for sending/receiving data because of AWS IoT connection limitations. They also use a dedicated channel for state updates. Both the client and the DAQ get the persisted state, which includes topic name, gain, and frequency information, from the cloud device shadows. Because of this initial sync, the DAQ can restart and pick up it's previous state to resume sampling, and the client can know the state of the DAQ.

State updates are initiated in the client and propagated to the DAQ. When the DAQ receives a state update, it kills the currently running threads handling the sampling and publishing and starts new ones with the desired parameters. Since the gain needs to be approximated, the approximated value is sent back to the client in the state update. It would be nice to keep an exponentially-weighted-moving-average of the sample frequency and update the client with the current frequency, since some variation is expected.

The DAQ batches samples to include 1500 samples. This value is currently hardcoded, but should be a function of frequency to minimize latency for varying frequencies. The value 1500 along with 5 concurrent connections allows us to achieve the throughput constraints for AWS IoT when sampling at 50 kHz.

When batches are sent through the IoT gateway, the gateway forwards the data to any subscribed client applications. It also runs a rule that inserts the data, which is a JSON object, into the DynamoDB table. Since it would be too expensive to reserve enough database throughput capacity to insert high-frequency data as individual samples, a filter is applied so that only batches that were samples at less than 1 kHz are entered into the database. To get around this limitation, the format of the data would have to change. One implementation would be to

convert the batch into a csv file and save the file in an S3 bucket. This would require cloud configuration changes and addition of S3 access to the client.

An AWS IAM user has been created with query access to the DynamoDB batch database. The client uses this user information to query the database for batches. The database used a composite primary key made up of the sessionID (hash key) and timeStamp (range key). In each row, the JSON representation of the batch is stored as a blob.

Authorization of clients and encryption of data is handled by the IoT service. Certificates were generated for both the DAQ and client. To avoid hardcoding the certificates, they were placed in configuration files that can be read by both the client and DAQ. The configuration files have been left in our GitHub repository for the project, but the certificates have been disabled so that they are not used by the public. The certificates and configuration files were left there so that their format can be seen as an example of what the program expects. Connections to the IoT service are authenticated with the certificates, and the IoT data is encrypted with them.

The authorized client machines may access the data being sampled through a client application of our design, from which currently streaming data may be verified and samples from the beginning of the session may be written to a file. The client is responsible for converting the raw ADC values to real voltage values as they are written to the file.

There are three “modes” in which the client functions: historical, streaming, and hybrid. This means that the client application can request older samples from the database that were previously stored, it can stream the data currently being samples, or it can do both of these in order to create a moving snapshot of the full history of a session. Thus that the user can view any

portion of the DAQ's data, old or new. The client is portable to any capable operating system and useable given the correct authorization keys from AWS.

References

"Beagleboard:Beagleboneblack - Elinux.Org". *Elinux.org*. N.p., 2016. Web. 18 Oct. 2016.

Appendices:

Specifications

- The device samples +/-5V (10V peak to peak) range electrical signals at a variable sample speed up to 50 kHz through a single sampling port, with at least 8 bit resolution per sample.
- The device functions when provided with power from a 5-Volt, 1-Amp DC power source, and runs an implementation of linux.
- It uses an ethernet or wifi connection to transmit sampled data over the internet, with minimum data loss, at a rate that allows real-time streaming of the data; at least 100 kB/s at any given time.
- An IoT backend, AWS IoT, handles the stream from the device and appropriately distributes the data to authorized clients. The total cost of this handling will be at most $10 per million messages.
- The backend is also capable of changing the device's state remotely, such as changing sample frequency and turning sampling on/off.
- A client application interfaces with the backend such that users are able to control and monitor a device or devices remotely.

Resources

- Tools & Applications
    - C
        - Standard libraries
        - LibPRUIO library
    - AWS services
        - IoT
        - DynamoDB
    - Java
        - AWS IoT API
        - DynamoDB API
        - jSwing API
        - GSON library
        - LGoodDatePicker library
    - Python
        - Numpy
        - Pandas
        - Plotly
- Hardware
    - Single Board Computer (BeagleBone Black)
    - Acquisition Circuit

- 2 general purpose 1N4001 diodes
- 1 Analog Devices OP484 4-circuit op-amp chip
- 1 Maxim DS3903 3-circuit digital potentiometer chip
- 2 1000pF 5% tolerance capacitors
- 5 $1\,k\Omega$, 1 $10\,k\Omega$ 0.1% tolerance resistors
- 1 TSSOP adaptable breakout board for prototyping
- Power supply, multimeter, function generator, breadboard

Testing

The system can be broken into several independently testable components. The following are the unit-testable systems: analog signal processing, sampling, data transmission, back-end data processing, back-end state, and client application.

Analog System Tests

The analog signal conditioning circuit has been simulated. All stages have been simulated independently and all stages have been simulated together as part of the overall design. Below are figures of each stage simulated individually and a brief description of their behavior is described.

For stage one it can be seen on the oscilloscope that there is indeed a unity gain being achieved with no attenuation of the input seen on the output (see Figure 10). The output is 7.909V peak to peak and the input is 8V peak to peak, the voltage gain can be calculated and comes out to be 0.9886.

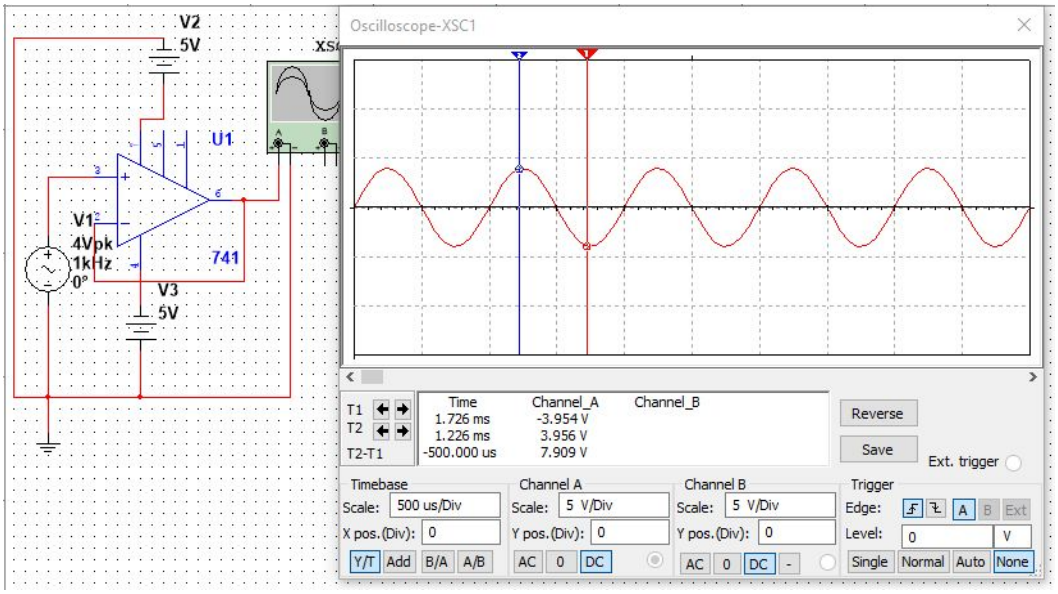

Figure 10

For stage two there are resistors in place of the digital potentiometers, the gain here is designed to be -0.5. In simulation one can clearly see that the gain of -0.5 is achieved as the output is 3.941V peak to peak and the input is 8V peak to peak (see Figure 11).

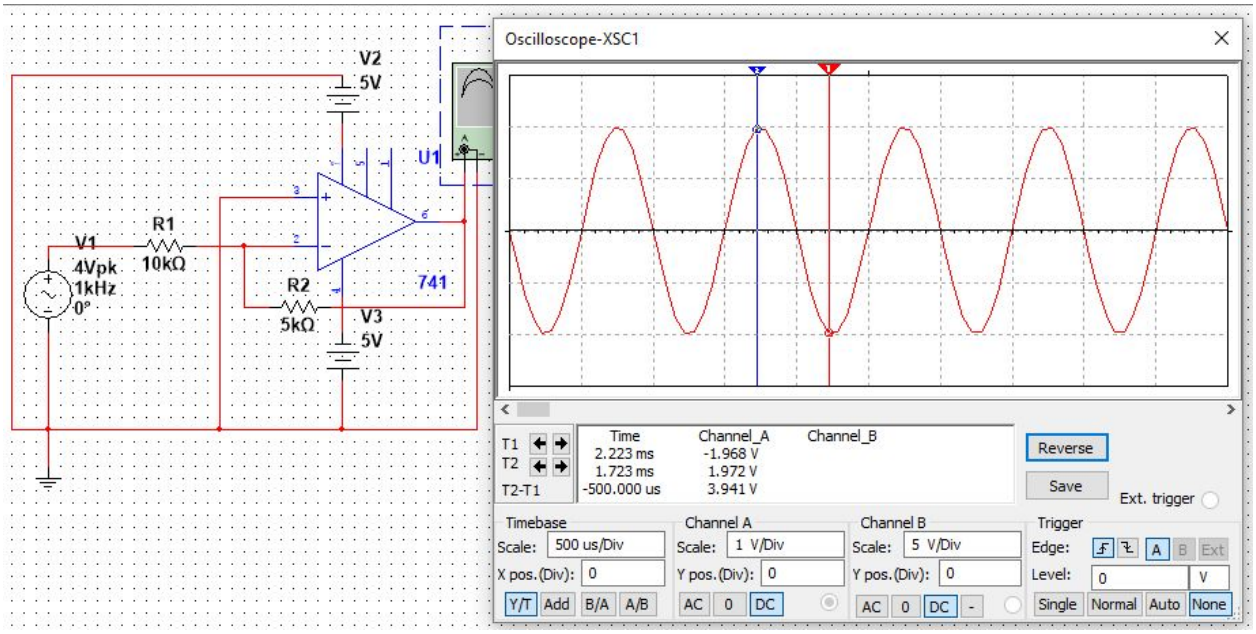

Figure 11

For stage three there is one resistor in place of the digital potentiometer. The designed sampling frequency was chosen to be 100kHz. Therefore the desired cutoff frequency should be 50kHz by Nyquist's filtering criteria. Calculating the cutoff frequency to be 50kHz resulted in a resistor value seen in Figure 12. One can see there is unity gain achieved in Figure 12 and in Figure 13 the desired cutoff frequency of 50kHz was achieved.

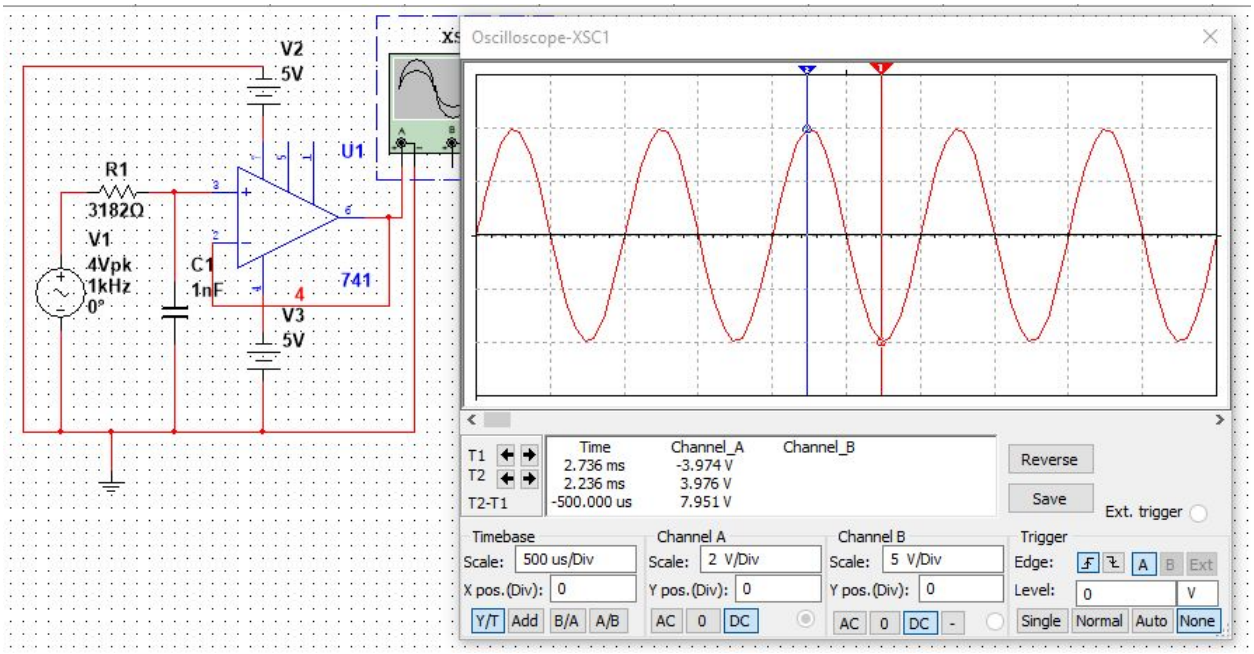

Figure 12

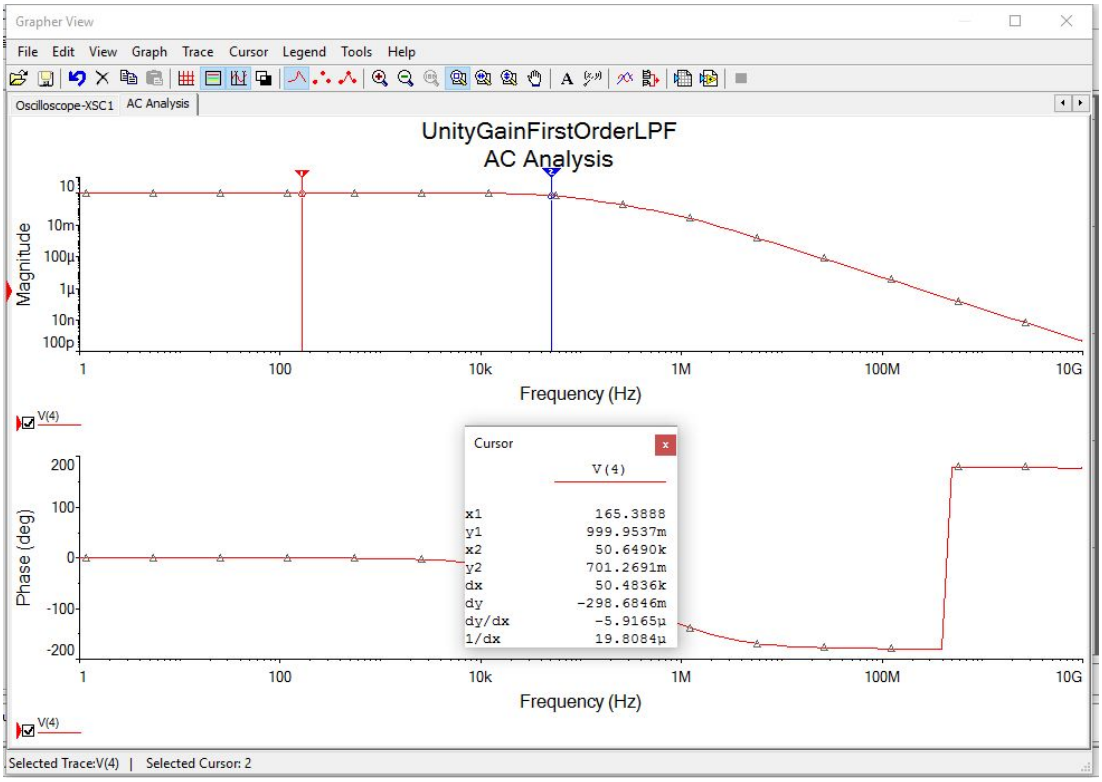

Figure 13

When stage five was simulated the desired DC offset of 0.9V was achieved and a gain of -1 was also achieved for inverting the input signal (see Figure 14)

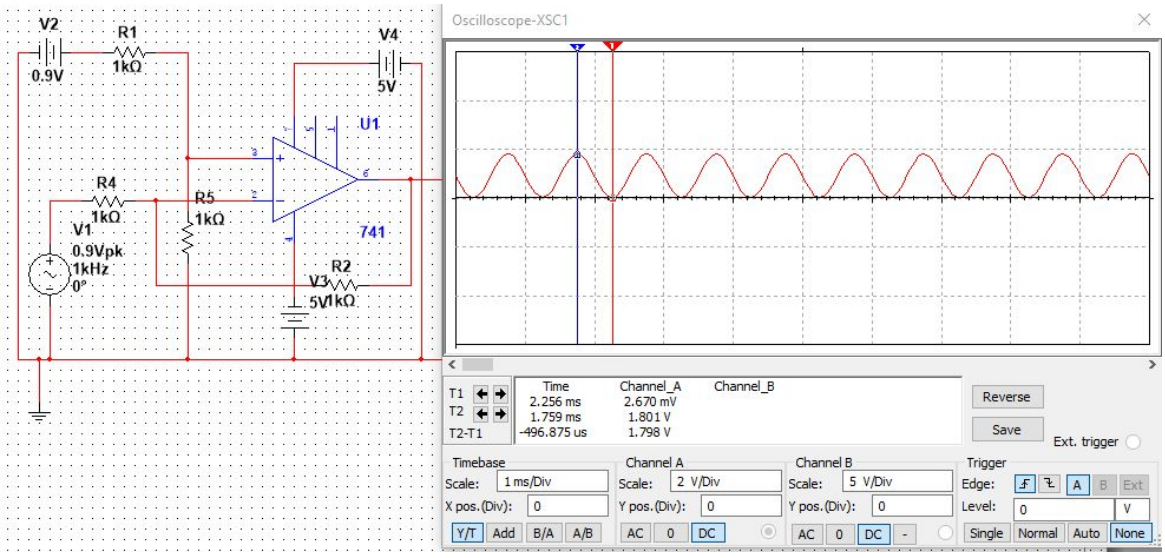

Figure 14

Finally, when implementing the original version of the circuit with stage five being the clipping stage with a zener diode and resistors the clipping was not taking place. This was resolved by switching the order of stages four and five. Stage four is the clipping stage and it is composed of just two general purpose diodes rather than using a zener diode with resistors. No individual simulation was done for stage four, merely just the overall design was simulated after this change. Present in Figures 15 and 16 are simulation results for the overall design after changes. Figure 15 is where there is no clipping taking place and Figure 16 is where the clipping (protection) mechanism is taking place. Figure 15 the input voltage is set to 1V peak, it can be seen that the DC offset of 0.9V is working and that the gain of 0.5 is applied appropriately and the output is 1V peak to peak or 0.5V peak. Figure 16 has an input voltage of 2V peak and clipping occurs at 0.2V and 1.6V which falls within the 0V and 1.8V limits of the ADC.

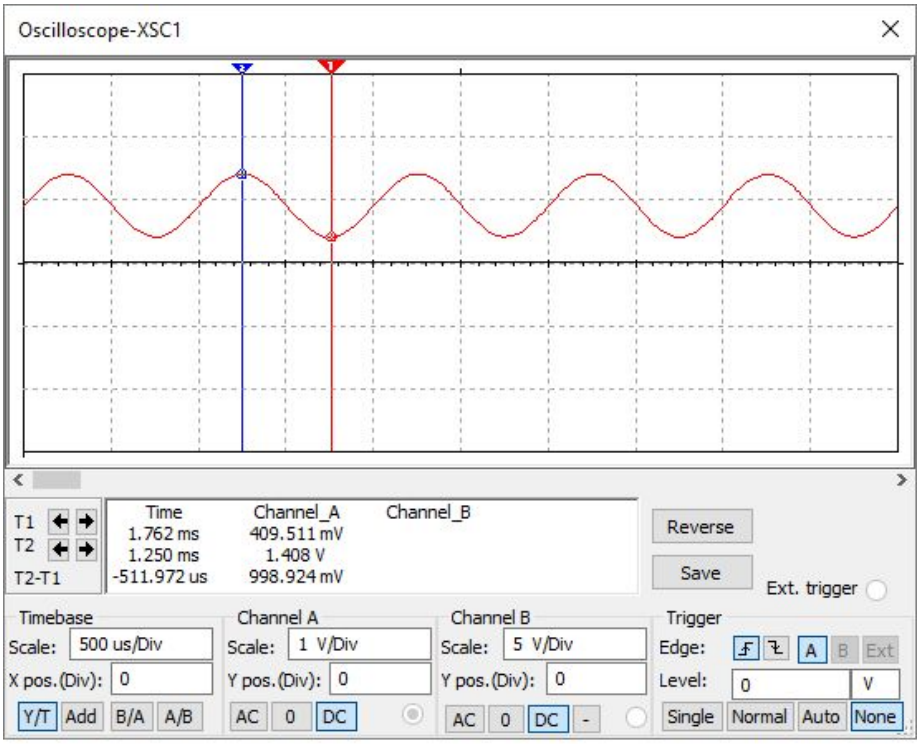

Figure 15

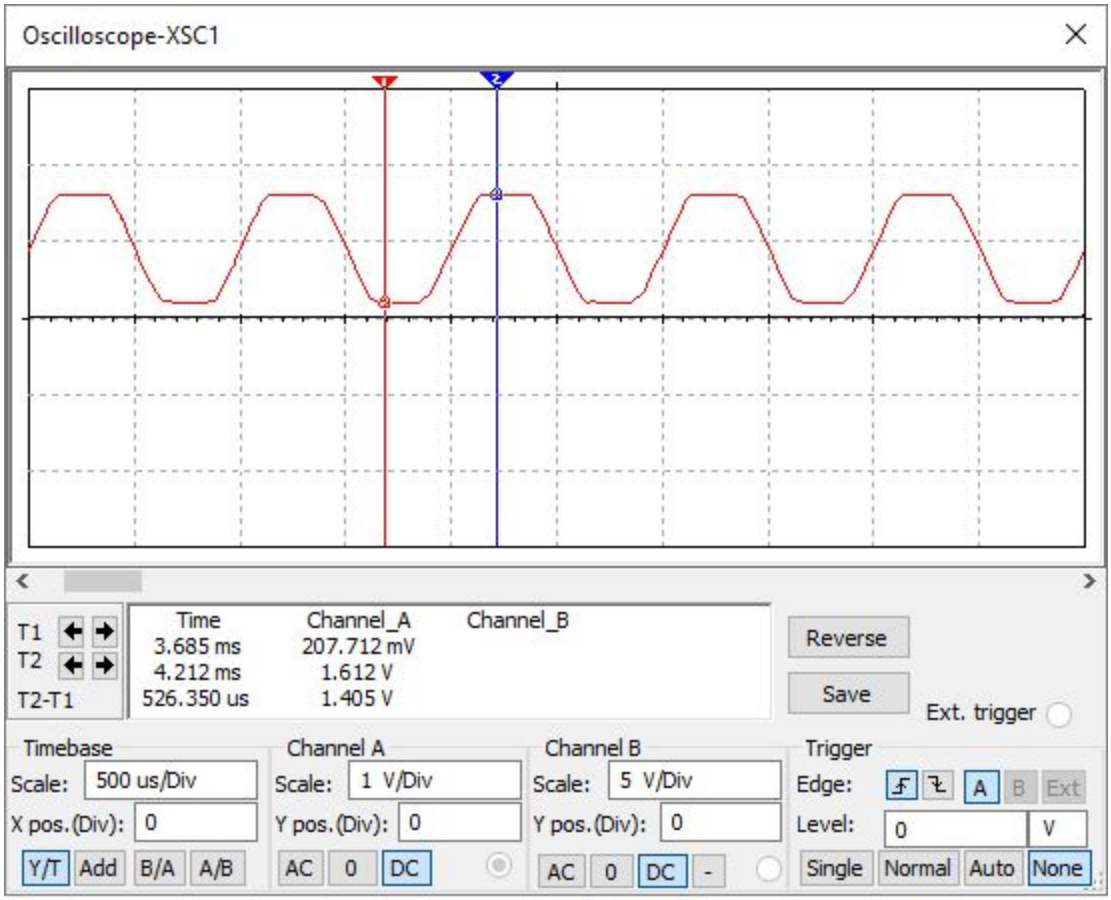

Figure 16

Each stage was built separately from one another and tested independently. The stages that require digital potentiometers (stage 2 and 3) were tested with fixed resistor values at first just to verify behavior before testing of the digital potentiometer control took place. Tests were initially performed on a breadboard and lots of instability oscillation was seen. The circuit was then moved to a perf board and all the parts were soldered as a whole circuit where each stage was tested independently. The noise was still seen and a 1nF capacitor was placed across the output of the last stage and got rid of the noise from the instability oscillation. For the first stage several input voltages were applied in the range of 0.1V peak to peak to 10V peak to peak, the

gain of one was verified by dividing the output over the input voltage. The input resistance was calculated on the breadboard by placing a resistor in series with the input voltage, measuring the voltage drop across the resistor and dividing by the current through the resistor, this yielded an input resistance of $203\,k\Omega$ ($1\,k\Omega$ was used: Rin = Vin/Iin = 1.024V/5.0414uA = $203\,k\Omega$). This verified that stage one was working as expected. Stage two was tested with fixed resistors where the gain was set up to be -0.5. The resistor across the output and inverting terminal was $500\,\Omega$ and the resistor in series with the input voltage and inverting terminal was $1\,k\Omega$. The same method to test stage one was done, the output voltage was divided by the input voltage and a gain of about -0.5 was seen. For example the input voltage was set to 6V peak to peak and the actual input voltage was 5.8V peak to peak and the actual output voltage was -3.03V, the gain is -3.03/5.8= -0.5224. This was repeated for several sets of input and output voltages. Stage three was tested with a fixed resistor of $3182\,\Omega$ to achieve a cutoff frequency of 50kHz (sample frequency of 100kHz which was beyond our requirements). The input voltage was set to 1V peak to peak and the frequency was turned up until the output voltage was 0.707 times the input voltage. When the output voltage was 0.710V the frequency was 50.1kHz which was what was expected. This was also tested at different input voltages and verified across the whole range. The last two stages were tested with the whole circuit integrated together on the perf board. Below in Figures 17 and 18 the oscilloscope traces show that the DC offset of 0.9V does indeed work along with the gain of 0.5 and clipping occurring as well as the signal passing through if it is below the clipping threshold. Figures 19 and 20 show the minimum clipping voltage (0.17V) along with the maximum clipping voltage (1.65V).

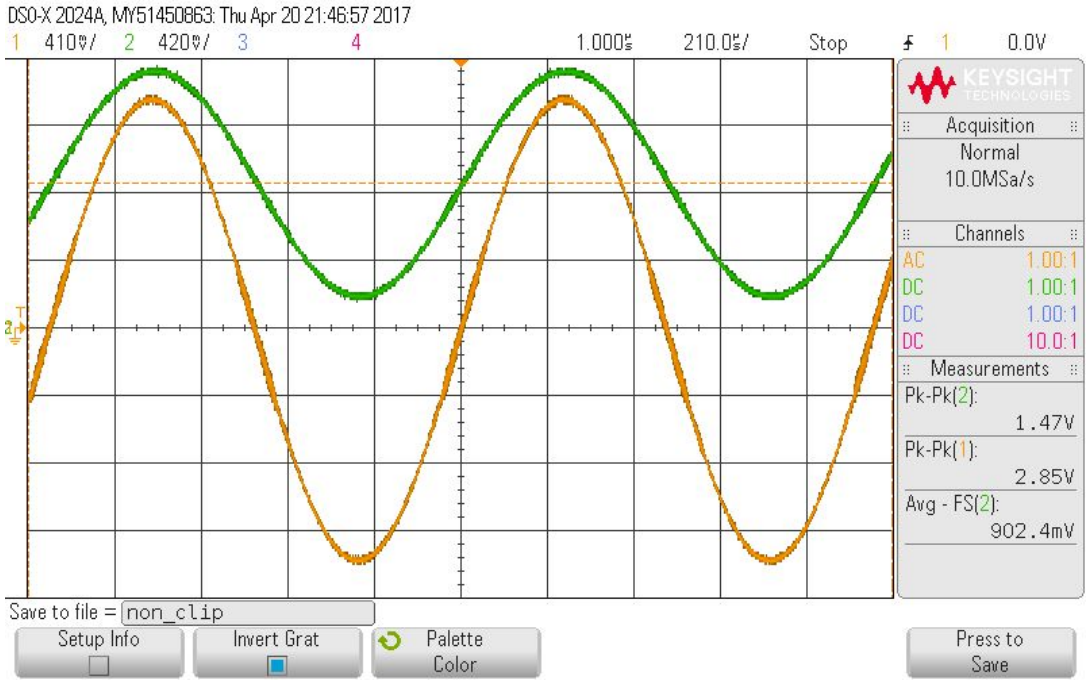

Figure 17

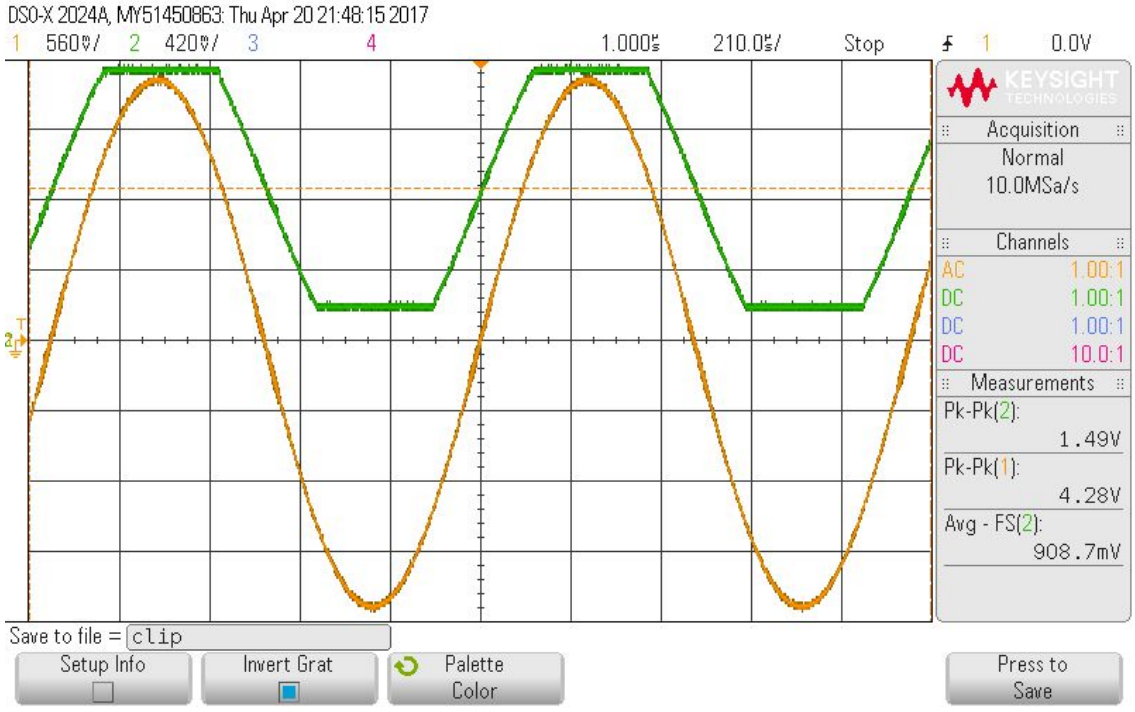

Figure 18

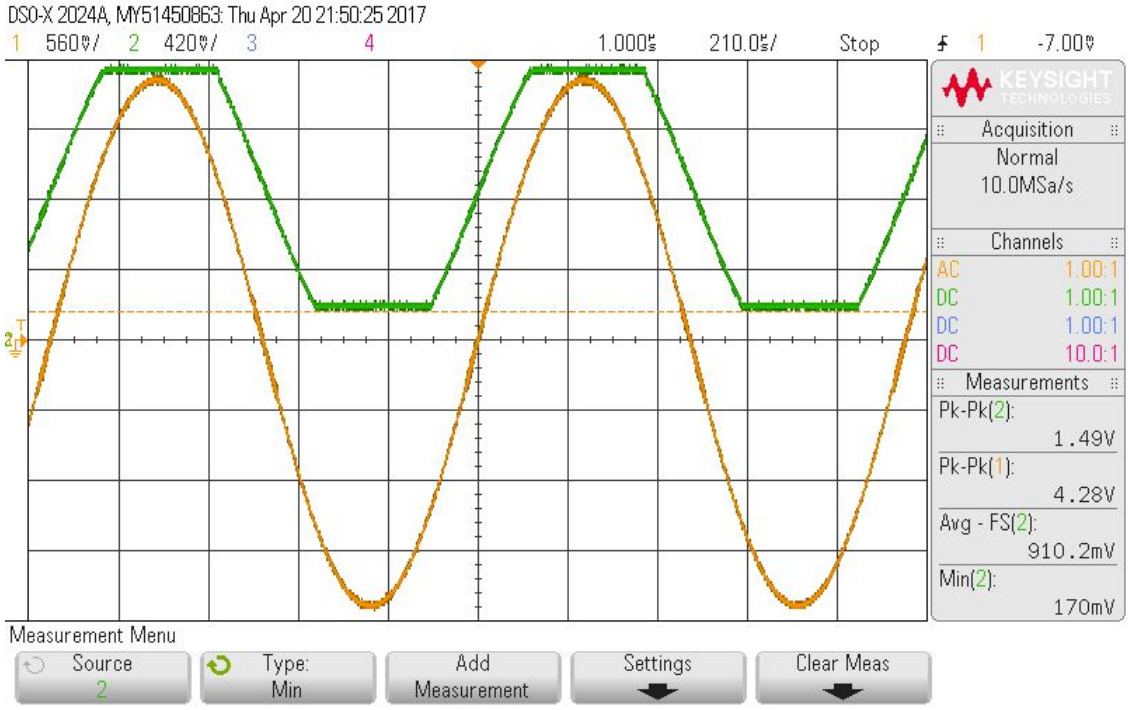

Figure 19

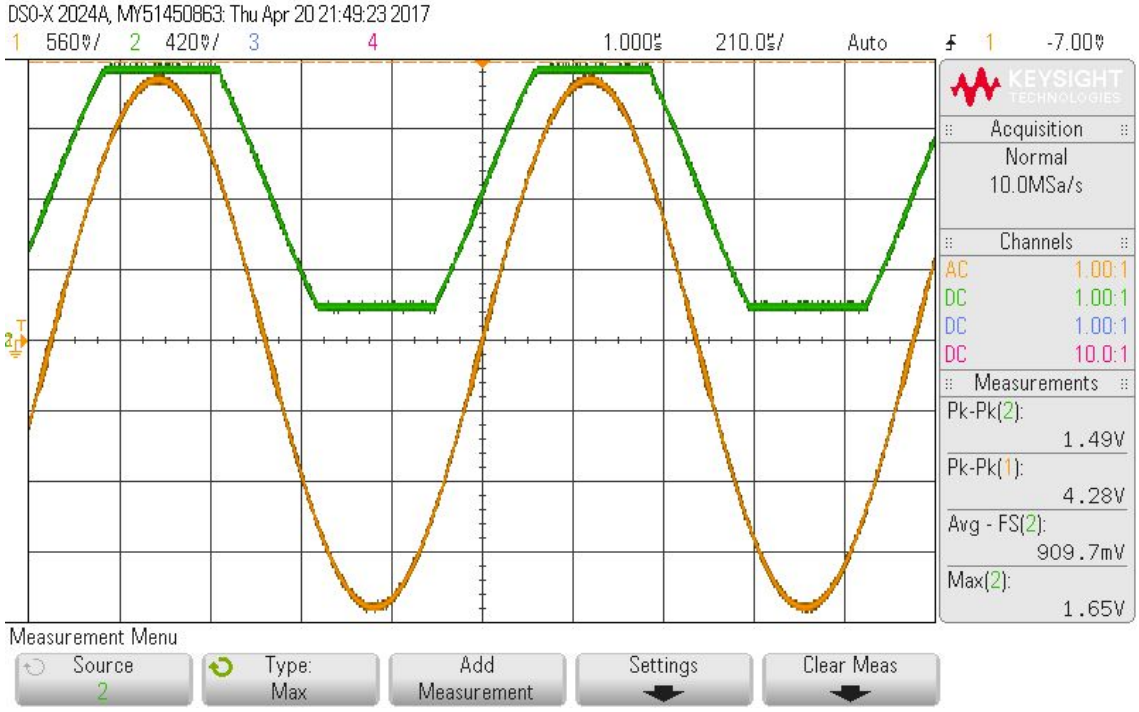

Figure 20

Single-Board Computer Tests

The ADC was tested by using a function generator to create a sine wave with varying frequencies, up to frequencies of 50kHz, and the recorded data points were been graphed to model the wave output. The accuracy of the results were within 3 decimal points of the known value from the function generator. This test was run without the analog conditioning circuit.

The Digital Potentiometers have been tested independently from the rest of the analog circuit by sending I2C commands to them from the SBC header. Additionally the software tool developed for the I2C communication was tested in the same way.

The test for network communication was to send data from the SBC to the client and verify that the client had received the samples that the SBC had sent out.

The entire SBC functionality was also tested in integration where all the pieces were expected to behave correctly together.

Backend Tests

The IoT communication has been tested by designing a mock DAQ and a mock client. These virtual devices can communicate with each other using the designed messaging protocol (the classes that define the protocol are the same as the ones used by the actual DAQ and client). The tests are all run manually by starting the publisher and subscriber using the same IoT topic on different computers. The publisher prints its messages to its console and the subscriber prints the messages it receives to its console. Using this test, we have verified that the AWS IoT backend works as expected and that we are using the IoT API correctly.

The back-end database has every message with low-frequency data that passes through the IoT gateway added to it. It has been tested that values get added in batches and that they are queryable from the client.

Client Tests

The client application needs to query the backend database. Mock data was published to the backend database via our Java publisher tests and retrieved into the client application. Then the application could run independently from dummy data. This test demonstrated that data could be received by the client, and further integration tests showed its functionality while connected to the actual device.

The Plotly grapher, was used to view the data coming in and verify timing and data integrity. Since the client was designed with portability in mind, our testing was done on two machines with different operating systems - Windows 10 and MacOS 10.12 - to prove that the application was portable.

Integration Tests

A full sampling integration test involves the system working including the conditioning circuit, ADC sampling, SBC networking, AWS networking, Client application, and the back propagation of user commands. So, the included components are the DAQ, backend, and client.

The test involved passing in several waves (sine, square, triangle) at different frequencies at different voltages. The range of frequencies tested include frequencies from 100Hz - 50kHz. The range of valid voltages include 0.1V peak to peak - 10V peak to peak. The observed results show that the input voltage is offset and clipped where necessary in the range of 0.1V peak to peak to 0.9V peak to peak. As we increased the input voltage up from 0.1V peak to peak we

were seeing correct results up until 0.9V peak to peak. Once we passed this input voltage unexpected behavior started to occur, we had a weird shaped sine wave that appeared to have harmonic issues. Upon further research it was discovered in the digital potentiometer (DS3903) datasheet that the input voltage range wasn't large enough for our application. The input voltage range for these digital potentiometers is -0.5V to +6.0V. This will not allow the full swing of the sine wave (negative voltage) to be visible and thus a different digital potentiometer should have been chosen. Further research shows that a good alternative for a digital potentiometer is an Intersil X9258TS24IZ-2.7. This part would work as the supply voltage can be both +5V and -5V allowing for negative voltage to come through the digital potentiometers.

The DAQ publishing was tested for several frequencies that covered the whole supported range. It was noticed that lots of data was missing when received by the client. This showed up as gaps when the data was graphed. Research showed that we might be hitting the publish and throughput limitations of the AWS IoT service. This discovery led to the implementation of sample batching and concurrent connections to the gateway. Batching improved performance so that the system was usable at low frequencies. Parallel connections seemed to improve performance a little, but this was not quantifiable. The resulting system has a throughput capable of around sampling around 5 kHz without problems. After some analysis, the problem is thought to be the *Java* code on the DAQ that publishes the samples generated by the *C* code. The publishing code cannot keep up with high-frequency publishing, even though we worked on optimizing it to the extent Java would allow.

To make the system operable all the way to 50 kHz, the Java code could be re-written in a language like C. Alternatively, the project could pivot to a more powerful SBC with much higher CPU frequency and more RAM to run the DAQ.

Personnel

Allen Burcham
648 Summer Winds Lane | Saint Peters, MO 63376 | (636) 288-0935 | burchama@slu.edu

Education

Saint Louis University, Parks College of Engineering, Saint Louis, MO — May 2017
Bachelor of Science in Computer Engineering — GPA: 3.6/4.0
- Dean's List (Spring 2016)

Saint Charles Community College, Cottleville, MO — May 2014
Associates of Arts in Computer Science — GPA: 3.8/4.0
- Dean's List (6 times)

Skills

Programming Languages:
- Python, C/C++, VHDL, AVR assembly, Java, and MATLAB

Software:
- Xilinx 13.2.1 ISE with Spartan3E NEXYS 2 development board
- Multisim circuit simulator
- AVR Studio 5.1 and STK500 development board for the Atmega32a microcontroller
- National Instruments TestStand
- Perforce

Technical Skills:
- Circuit analysis tools including: Digital Multi Meters, Power Supplies, and Oscilloscopes
- Update circuit boards embedded boot code and application firmware

Work Experience

Emerson Remote Automation Solutions, Marshalltown, Iowa
Test Engineer Intern — May 2016 – August 2016
- Write testing procedures and update documentation database of testing results
- Update devices embedded boot code and application firmware
- Unit and sanity testing of software and firmware
- Debugging of existing software and test scripts

St. Charles Community College, Cottleville, Missouri
Educational Aide — August 2013 – Present
- Assist students with math questions to increase their knowledge
- Identify students' difficulties and devise methods to ensure success in classes
- Help students develop basic study skills to become independent

OfficeMax, Cottleville, Missouri
Store Sales Consultant — June 2012 – December 2014
- Provided customer service and current product training including computers, printers, networking devices, and office supplies.

Saint Charles School District, Saint Charles, Missouri
IT Intern — January 2011 – May 2011
- Provided knowledge and assistance including major technologies, hardware systems, and support methods used to run a large, real-world network.

Relevant Coursework
- Electronic Circuit Design (with Lab)
- Analog Integrated Circuit Design
- Linear Systems (with Lab)
- Semiconductors Devices
- Advanced Digital Design
- Microprocessors (with Lab)
- Computer Systems Design (with Lab)
- Computer Architecture and Organization

Activities
- **Institute of Electrical and Electronic Engineers**, Saint Louis University — 2015 – Present

Christopher William Lucas
chris@chriswlucas.com | (269) 910-8203 | www.chriswlucas.com

EDUCATION

Saint Louis University: August 2013 – May 2017
Parks College of Engineering, Aviation and Technology: St. Louis, MO

- Bachelor of Science: Computer Engineering, Computer Science
- Minor: Engineering Mathematics
- Cumulative GPA: 3.92/4.0

WORK EXPERIENCE

Software Development Engineering Intern: May 2016 – August 2016
Amazon.com: Seattle, WA

- Developed a metric gathering and aggregating solution for users of Apache Spark
- Learned about how software is developed in a large company
- Became comfortable with many internal tools and used them to automate workflow

Software Development Intern: May 2014 – August 2015
Qvolve, LLC: Kalamazoo, MI

- Worked during school breaks designing database solutions with FileMaker Pro
- Wrote FileMaker Pro scripts, normalized data, designed user interfaces, and created web widgets
- Attended FileMaker DevCon 2014 in San Antonio, TX

PROJECTS

Context-Aware Synonym Suggester (CASS): January 2016 – December 2016
Capstone Project: Computer Science

- Developing a standalone Word Sense Disambiguation (WSD) Java package
- Writing WSD test suite to verify accuracy of implemented algorithms
- Creating an extension to LibreOffice Writer that uses the WSD package to suggest synonyms

TECHNICAL SKILLS

Working knowledge of Java, C++, Python, Scala, MatLab, and VHDL
Basic knowledge of Haskell, Swift, and x86 Assembly
Familiar with using the Unix command line, Git, and Eclipse

HONORS

Honor Societies: ΤΒΠ (Engineering), ΑΣΝ (Jesuit), ΟΔΚ (Leadership)
The Computer Science Award: Department of Mathematics and Computer Science – 2014
The Calculus Award: Department of Mathematics and Computer Science – 2014
Dean's List: listed five times

ACTIVITIES

Parks Racing, Formula SAE Team: Treasurer 2015 – 2016, Member 2013 – present

Habitat for Humanity: Assistant Vice President of Builds 2013 – 2014
Boy Scouts of America: Eagle Scout 2013

Fausto Tommasi
17791 Old Jamestown Rd
Florissant, MO 63034
(314) 724-7477
ftommasi@slu.edu
https://www.linkedin.com/in/faustotommasi
https://github.com/ftommasi

Education

Saint Louis University – Comp. Engineering/Comp. Science **August 2013 – *Current**
Est. Graduation Date: December 2017 **GPA 3.41**

Courses

Capstone – Computer Science
Computational Problem Solving
Computer Architecture
Computer System Design
Data Structures
Differential Equations I
Discrete Mathematics
High Performance Computing
Junior System Design
Linear Algebra
Linear Systems
Linear Systems Lab
Microprocessors
Microprocessors Lab
Object Oriented Design
Software Engineering

Achievements

Boeing BOLD Scholarship
ICPC SLU team member
SIM – St. Louis Scholarship
SLU Dean's List
SLU Ignatian Scholarship
ICPC Semi-Finalist 2015

Skills

Proficient with computer software/hardware use and troubleshooting
Proficient with Hardware Description Language VHDL
Experienced in various programming languages including Python, JavaScript, Java, C#, C/C++, MatLab and Assembly
Experienced with embedded systems development in devices like RaspberryPi and Pumpkin.
Experience with development in Windows and Linux environments.
Experience with Agile development
Experience using Git/Github/VSO and version control
Proficient in the Spanish Language and Culture/Completely Bilingual
Proficient in teaching/tutoring at several levels

Related Experience

Software Developer Intern - Premium Retail Services **June 2016 – August 2016**

- Experience with Scrum Agile Manifesto, stand-ups, sprints, and project-planning
- Experience Developing Object Oriented Systems using Architecture patterns
- Experience with Relational Databases and SQL databases
- Experience with the Software Life Cycle from a technical, operational, and business perspective
- Experience with .NET and Entity Frameworks
- Experience using and creating REST Web APIs
- Experience with modern JS/HTML technologies like Bootstrap, JQuery, Ajax and Angular

Command, Data Handling, and Instructions - SSRL at SLU **August 2015 – December 2015**

- Responsible for software development relevant to navigation and operation of the CubeSat
- Experience with embedded systems, hardware, and software
- Experience with team meetings, action items, deadlines, and general organizational skills
- Experience with Imaging software using libraries like OpenCV

Taylor Watson

watsont@slu.edu
(972) 998-0597
3701 Lindell Blvd. Apt. 9A
St. Louis, MO 63108

Education

Saint Louis University, St. Louis, MO — Cumulative GPA 3.72 / 4.00

Bachelor of Science in Computer Engineering and Computer Science — May 2017

Relevant Coursework

Operating Systems, Computer Systems Design, Network Design, Algorithms, Computer Security, Linear Systems, Microprocessors, Semiconductors, High-Performance Computing, Object-Oriented Software Design

Skills

Programming/Description Languages: C++, Java, C#, Python, SQL, MATLAB, VHDL

Applications/APIs: Windows API, Visual Studio, IBM Clearcase, Xilinx, Modelsim, Eclipse, Unix tools

Work Experience

The Boeing Company, St. Louis, MO — May – August 2016

- Programmed as a software engineering intern in Flight Simulation, helping to develop tools for the EA-18G Growler.

Saint Louis University Math and Computer Science Department, St. Louis, MO — March 2015 –January 2016

- Engaged in a computer science/natural language processing research project headed by Dr. Kevin Scannell.

Brookhaven Country Club, Dallas, TX — June 2013 – Aug 2015

- Performed lifeguarding duties, maintained pool areas, and supervised other employees.

Cistercian Preparatory School Summer Camp, Irving, TX — June 2014 – June 2015

- Supervised and helped to teach children under 14 as a camp counselor at my high school, and organized various activities.

Honors/Clearances/Certificates

Department of Defense Security Clearance: Secret	June 2016 - June 2026
James G. Costigan Scholarship	Scholastic Years 2015 and 2016
Boeing BOLD Scholar	Scholastic Year 2014
Parks College Dean's List	Fall 2013/14/15 + Spring 2015/16
Saint Louis University Provost Scholarship	August 2013 - present

Organizations

IEEE	Sept 2014 – present
Sigma Tau Gamma Fraternity	Jan 2014 – present
Greek Standards Board	Jan 2015 – Jan 2016