Electrical-Impedance Biofeedback Instrument for Swallowing Rehabilitation

Thesis submitted in partial fulfilment of the requirements for the Degree of Masters of Engineering in Electronic and Electrical Engineering

By

Ryan McGurk

09.2022

University of Canterbury

Electrical and Electronic Engineering

Christchurch, New Zealand

Abstract

Dysphagia is the difficulty or abnormality of swallowing. It is usually a consequence of another health condition and may be present in any of the phases of swallowing. The cause may be structural or neurological. The effects of dysphagia range in severity; milder symptoms range from discomfort to difficulty swallowing. More severe effects include expelling food or liquid out through the mouth and nose or aspiration of material into the lungs, resulting in heavy bouts of coughing and increasing the risk of developing pneumonia.

Treatment for dysphagia, if applicable, therefore depends on the type and cause of dysphagia. Evaluation charts, barium swallow, pharyngeal manometry, and endoscopy are some of the available tools in the diagnosis and classification of dysphagia.

Patients diagnosed with neurogenic dysphagia may undergo motor-training exercises for swallowing rehabilitation. However, literature suggests that motor-training exercises are more effective when the patient is presented with some form of interpretable feedback of their motor activity – known as biofeedback. Patients themselves have expressed the view that biofeedback gives something to aim for. To this end various techniques for biofeedback have been formally reviewed, including surface electromyography, acoustic, endoscopy and ultrasound amongst others.

Pharyngeal manometry is an invasive procedure which measures pressure along specific sections of an endoscopic-style probe and provides measurements that identify pharyngeal muscle activity sequencing. These measurements have been used as biofeedback information in rehabilitative treatment. However, such techniques are typically performed in a clinical setting and only administered by suitably trained staff.

Bio-impedance has been researched and demonstrated in other literature to be a suitable tool for assessing swallowing function, comparisons have been made to manometry with mixed results.

Previous Masters students in the University of Canterbury have investigated the suitability of a bio-impedance sensing device to perform as an easy-to-use, non-invasive alternative to intrusive pharyngeal sequence measuring devices. Though impedance measurement changes were obtained from a human subject, the research concluded that the number of measuring channels must be expanded. The device concept was named GULPS (Guided Utility for Latency in Pharyngeal Swallowing).

This project aims to expand the number of channels of the GULPS prototype whilst retaining the sensitivity and signal gain of the original.

The previous implementations of GULPS took a multi-frequency approach as a means to create channels with 40 kHz and 70 kHz as the selected nominal current injection frequencies. This approach makes adding further channels very challenging as each channel is in effect a complete impedance measurement device, each with their own signal injection circuit, amplifiers, filters and detectors. Each signal injection circuit must have its power supply isolated from the rest posing further design considerations.

Channel multiplexing was determined to be the most efficient method by which to add additional channels to the bio-impedance module. This presents its own set of challenges, specifically the amplitude detector board must settle on a steady output considerably faster than the channel multiplexing rate, to allow sufficient time for multiple samples of the channel value to be taken for averaging.

A tetrapolar electrode measuring scheme was selected as this approach has proven successful in previous implementations of GULPS, as well as other bio-impedance projects. The number of channels was expanded to sixteen and comparable sensitivity was demonstrated on simulated loads of 120 Ω subjected to a 10% drop in nominal impedance. A test chamber was constructed with channel electrodes spaced vertically along a column filled with saline solution to simulate conditions comparable to that of a human neck. A narrow conductive cylinder attached to an insulating rod was lowered and raised through the path of the current injection and voltage measurement electrodes. This resulted in sufficiently large voltage swings in the corresponding channel with minimal cross-talk or interference to other channels.

Though the settling time for the new detector was measured to be sufficiently fast for the desired sampling rate of 1000 Hz per channel, the sampling rate had to be lowered to 700 Hz in the final implementation of the GULPS hardware. The cumulative effect of series-connected resistances meant filtering capacitors had to be lowered to almost parasitic values to try to maintain the required time constants, until the delays could be reduced no further. The per-channel bandwidth is limited to 10 Hz after additional filtering at the output stage.

The prototype is initiated and configured via USB by an application developed in Visual Studio. The application allows for frequency selection for the constant current source from 70 kHz to 2 MHz so that the most suitable injection frequency could be determined experimentally.

Due to time constraints the prototype was not tested on a test subject nor comparisons made to manometry readings. Testing with a simulated load designed to mimic human tissue demonstrated sufficient signal gain and low inter-channel interference, suggesting the device would be suitable to go to human trials.

Acknowledgements

I would like to thank my supervisors, Dr Paul Gaynor and Dr Maggie-Lee Huckabee for their support and guidance. From the Rose Centre I would like to thank Ester Gui Hernandez who contributed her time and knowledge to the project. I would also like to thank Hayden McKechnie, a final year student at the University of Canterbury at the time, for his contribution to the project.

Lastly, thank you to my family for your patience and support.

Contents

Abstract 2
Acknowledgements 4
Chapter 1.0 Introduction 9
1.1 Aims 9
Chapter 2.0 Background 11
2.1 Swallowing Biomechanics 11
2.1.1 Pre-Oral phase 11
2.1.2 Oral phase 11
2.1.3 Pharyngeal phase 12
2.1.4 Oesophageal phase 12
2.2 Dysphagia 13
2.2.1 Pharyngeal Manometry 13
2.3 Electrical Bio-impedance 14
2.3.1 Impedance dynamics during swallowing 15
2.4 Chapter Summary 15
Chapter 3.0 System Design 16
3.1 Previous Work 16
3.2 System Design 17
3.1 Concept block diagram 18
3.1.1 Multiplexing 20
3.2 Current Source System 20
3.3 Voltage Measurement System 21
3.3.1 High-impedance voltage sensing 22
3.3.2 The demodulator 22
3.3.3 Voltage difference amplifier 23
3.3.4 Individual channel outputs 24
Chapter 4.0 Hardware Implementation 26
4.1 Current Source 26
4.2 Voltage sensor input stage 29
4.3 Amplitude detector module 30
Chapter 5.0 Testing and Preliminary Results 34
5.1 Testing the current source module 34
5.2 Testing the envelope detector 34

5.3 Testing the averaging circuit 37
5.2 Live channel output 39
Chapter 6.0 Computer Software 42
Chapter 7.0 Operational Results 43
7.1 Simulated Resistive Load 43
7.2 Crosstalk test 44
7.3 Saline solution 45
Chapter 8.0 Conclusions and Future Work 48
References 49
Appendix A 50
Appendix B 56
Appendix C 67
PCB Artwork 68

Table of figures

Figure 1 The bolus and principal parts involved in swallowing 11
Figure 2 Bolus pushed into the pharynx by the raising of the tongue 12
Figure 3 Pharyngeal phase 12
Figure 4 Oesophageal phase 13
Figure 5 Resistor-Capacitor Model for biological tissue 14
Figure 6 Current path through biological tissue at low and high frequency. 14
Figure 7 Complex impedance measurement 15
Figure 8 Tetrapolar sensing arrangement 17
Figure 9 Hardware block diagram and Region of Interest, ROI 18
Figure 10 Current Source System block diagram 19
Figure 11 Voltage Measurement System block diagram 19
Figure 12 Constant current source concept of operation 21
Figure 13 Multiplexing of constant current source 21
Figure 14 Multiplexing the voltage measurement system 22
Figure 15 Function of a maximum and a minimum peak detector 23
Figure 16 Upper and Lower envelopes 23

Figure 17 Channel amplified against average value 24
Figure 18 Combining top and bottom change signals 24
Figure 19 Individual channel outputs 24
Figure 20 Circuit diagram and implementation of constant current source 26
Figure 21 Balanced bi-directional constant current source testing 27
Figure 22 Improved constant current source schematic 27
Figure 23 Constant current source used in the last implementation of the GULPS hardware 28
Figure 24 Functional block diagram of complete current source module 28
Figure 25 Final implementation of current source module 29
Figure 26 Voltage input section, AD8421 30
Figure 27 Phase signal, MAX941 31
Figure 28 Amplitude detector generates both top and bottom envelopes of the input signal 31
Figure 29 Difference amplifier, relative to 10 second channel average 32
Figure 30 Recombining the envelope change signals 33
Figure 31 Testing the current source and multiplexer. 100ns/div, 200 mV/div 34
Figure 32 Generating the phase signal, green, from the input signal, red. 5 volts, 2 us per division. 35
Figure 33 Signal input vs first peak detection storage capacitor. 1 volt, 2 us per division. 35
Figure 34 Top and bottom peak storage capacitors. 2 V/div, 2 µs/divi. 36
Figure 35 Top and bottom envelopes (blue traces) produced from input signal varying in amplitude (yellow trace). 500 mV/div, 20 µs/div. 37
Figure 36 Voltage present at averaging capacitors. 2 volts, 10 us per division. 38
Figure 37 Live reading from channel, bottom envelope. 2 volts, 10 us per division. 38
Figure 38 Output demultiplexer with 10 Hz LPF and buffer. 39
Figure 39 Detector board practical implementation. 40
Figure 40 Microcontroller hardware elements 41
Figure 41 Screenshot of Visual Studio application 42
Figure 42 Test load circuit schematic and implementation. 43
Figure 43 Test, load has 20% on time. 43

Figure 44 Test, load has 50% on time. 44

Figure 45 Test, load has 80% on time 44

Figure 46 Crosstalk test. 200mV/div, 500 ms/div. 45

Figure 47 Test chamber and test slug. 46

Figure 48 Four channel test results. 47

Figure 49 Detector PCB. 68

Figure 50 Current source and controller PCB 68

Chapter 1.0 Introduction

Swallowing is a function we all take for granted, but is critical for the consumption of food and liquids. Any disruption to normal swallowing activity can be extremely impactful to normal everyday quality of life. A dysfunction of swallowing is known as dysphagia.

Patients diagnosed with dysphagia may benefit from motor training exercises as part of their rehabilitation schedule. One difficulty expressed by patients is the lack of some sort of feedback, a visual representation would aid with positive and negative re-enforcement and thus accelerate the learning process [1].

Pharyngeal sequencing measurement devices aid in the diagnosis and treatment of pharyngeal mis-sequencing disorders. Pharyngeal manometry is one of the methods used in the Rose Centre for Stroke and Recovery Research to measure pharyngeal sequencing [1]. Pharyngeal manometry measures pressure variations in three locations along the length of the probe to provide sequencing data. However, pharyngeal manometry and other measurement methods are invasive and require the procedure to be carried out in a clinical setting and may only be administered by trained staff. Of particular interest to this project is pharyngeal sequencing measurement as a rehabilitation tool in a domestic setting. Kusuhara et al [2] developed and trialled a single channel bio-impedance based approach to asses swallowing function which was considered to reflect changes in impedance due to pharyngeal movements.

This project is a continuation of work done by two prior Master's students to develop a multi-channel bio-impedance measurement device. Previous research concluded that the number of channels needed to be expanded [3].

1.1 Aims

This project aims to further develop a multichannel bio-impedance sensor for swallowing rehabilitation suitable for use in a domestic setting. Improvements will be made by increasing the number of channels to 16 each sampled at 1 kHz. The following chapters present the project.

Chapter 1: An introduction to the project.

Chapter 2: Is a brief description of the biomechanics of swallowing. What is dysphagia, and how bio-impedance sensing can assist in diagnosis and treatment.

Chapter 3: Builds on the work carried out by two previous students and details the design and functionality of the new implementation of GULPS.

Chapter 4: Implementation of the designed system.

Chapter 5: Testing of the individual modules.

Chapter 6: Details of the computer software written to configure the hardware.

Chapter 7: Results obtained from the testing the system on a simulated load.

Chapter 8: Conclusions and future work.

Chapter 2.0 Background

Swallowing is a complex process involving voluntary actions and involuntary reflexes by which food or drink is passed from the mouth to the stomach. An understanding of the swallowing process is required in the design, implementation and testing of a biofeedback device. This section will describe a summarised overview of the swallowing process, followed by the main classifications of dysphagia and where and how biofeedback is beneficial. Finally, an overview of electrical bioimpedance will be discussed.

2.1 Swallowing Biomechanics

The swallowing process involves more than thirty nerves and muscles [4]. Failure to coordinate the swallowing process could result in choking or pulmonary aspiration which could in the long-term lead to other complications such a pneumonia. The process of swallowing is divided into four phases: the pre-oral, oral, pharyngeal and oesophageal phases.

2.1.1 Pre-Oral phase

The pre-oral phase is overlooked in traditional, anatomical, models of swallowing. However, this phase is of importance in rehabilitation techniques which take into account pre-oral environmental and cognitive factors. This phase takes into account the person's ability to think about swallowing.

2.1.2 Oral phase

This phase begins when food or drink enters the mouth. This phase includes chewing and forming of the bolus. Figure 1 is a cut-away illustration of the head depicting the bolus in the mouth, and identifies some of the key biological parts involved in the swallowing process.

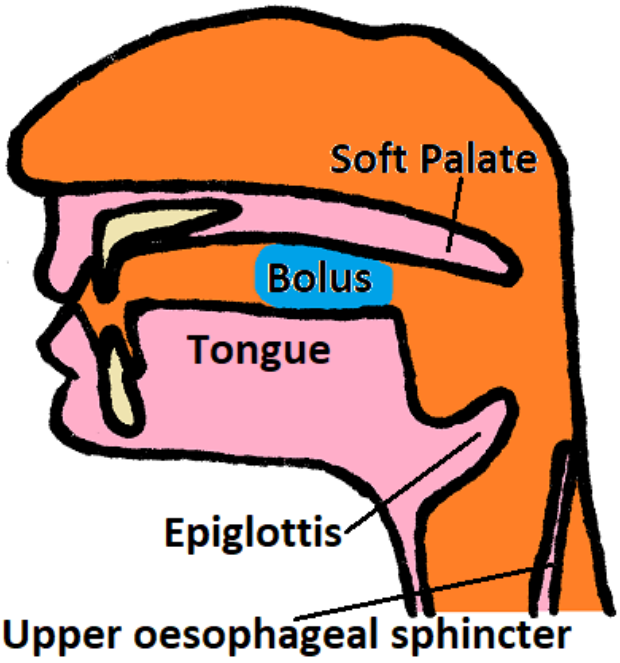


Figure 1 The bolus and principal parts involved in swallowing

The oral phase is the voluntary part of swallowing. When the bolus is sufficiently prepared, the tongue pushes the bolus to the back of the throat, the pharynx, as demonstrated in Figure 2.

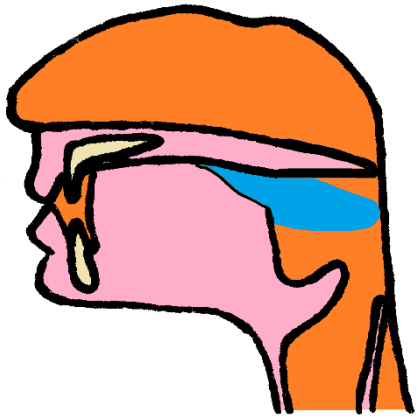


Figure 2 Bolus pushed into the pharynx by the raising of the tongue

When the bolus comes into contact with tactile receptors in the oropharynx the pharyngeal phase is triggered.

2.1.3 Pharyngeal phase

In the pharyngeal phase the swallow reflex is initiated under involuntary neuromuscular control. As depicted in figure 3, the tongue blocks the oral cavity and the soft pallets blocks the nasal cavity. The larynx is pulled up and the epiglottis folds down to cover the trachea. The upper oesophageal sphincter opens to allow passage of the bolus into the oesophagus.

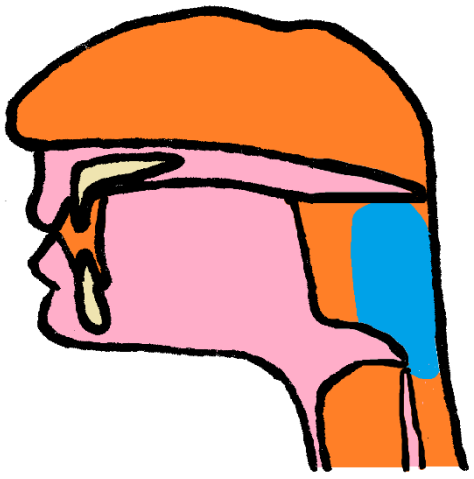


Figure 3 Pharyngeal phase

2.1.4 Oesophageal phase

The bolus is propelled down the oesophagus by peristaltic wave contractions as depicted in figure 4. The larynx moves back down to its original position.

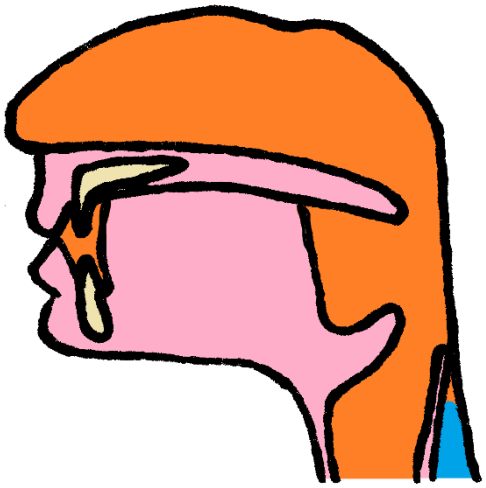


Figure 4 Oesophageal phase

2.2 Dysphagia

Dysphagia is the difficulty or disruption to the swallowing process. The cause may be structural or neurological and may occur in any of the three phases after the pre-oral phase. Several tools are available to help diagnose and classify dysphagia, these include evaluation charts, barium swallow, pharyngeal manometry, and endoscopy.

The classification of dysphagia is done by combining the affected phase with the cause, being either structural or neurological.

Treatment for structural dysphagia may involve corrective surgery. Patients diagnosed with neurological dysphagia may benefit from motor-training exercises. Some sort of biofeedback device may assist the patient as both positive and negative actions may be correlated visually in real time. Biofeedback also assists the patient in identifying if the correct muscles are being targeted by the exercise and the patient may take corrective action.

Different technologies exist to provide biofeedback, of interest to this thesis is electrical bio-impedance applied as a non-invasive alternative to pharyngeal manometry.

2.2.1 Pharyngeal Manometry

Pharyngeal manometry is an invasive procedure typically carried out in a clinical setting by trained staff. It is both a diagnosis and a rehabilitative tool. Pressure sensors arranged along the length of a probe are inserted into the test subject's throat. The sensors provide immediate pressure feedback which can give an indication of swallowing sequence. Pharyngeal manometry finds applications in swallowing rehabilitation motor-training as it provided immediate feedback indicating if the desired muscle is being targeted by an exercise.

2.3 Electrical Bio-impedance

Electrical impedance is the measure of the opposition to current flow. Electrical bio-impedance relates to the impedance presented by biological tissue. Impedance takes into consideration the combined effect of resistance and reactance; in the case of bio-impedance this reactance is typically capacitive. The resistor capacitor model for biological tissue, as shown in Figure 5, is a commonly used approximation to model biological impedance [5].

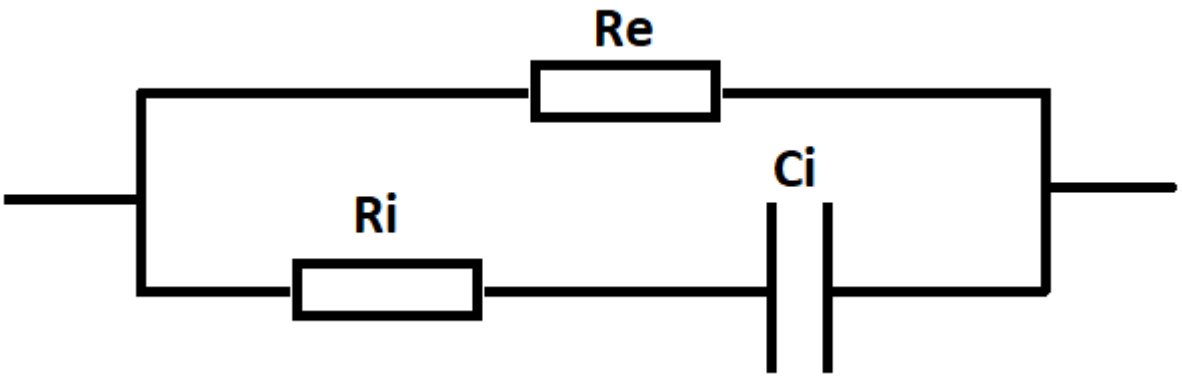


Figure 5 Resistor-Capacitor Model for biological tissue

Capacitive reactance is a function of frequency, where the reactance decreases as frequency increases.

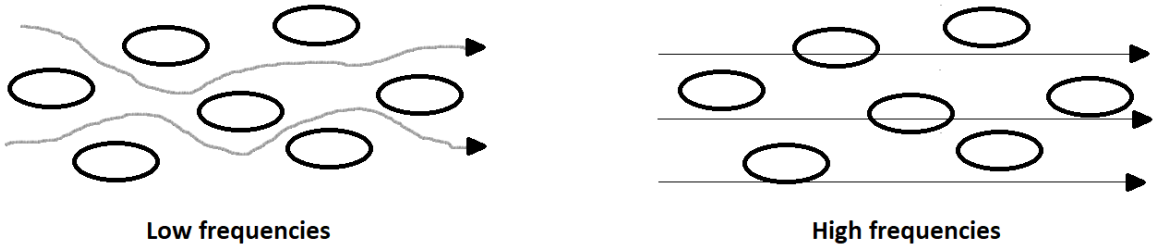


Figure 6 Current path through biological tissue at low and high frequency.

Body tissues and fluids have different levels of electrical impedance. At low frequencies, the current flows through the extracellular space, whereas at higher frequencies the cells present a lower impedance to the current which may also flow through the Intercellular space (depicted in Figure 6). These two readings, therefore can be used to calculate intercellular impedance. Patented Bio-Impedance Spectroscopy (BIS) uses relative impedances measured at over two hundred specific frequencies and modelling to provide a highly accurate body composition reading.

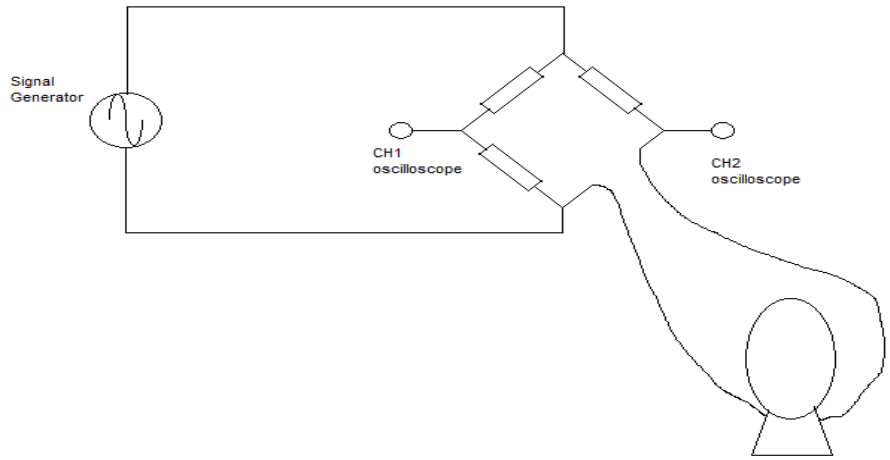


Figure 7 Complex impedance measurement

The author implemented a Wheatstone bridge approach, Figure 7, to attempt to measure bio-impedance at various frequencies, the results however, were inconclusive and did not correlate with expected values. This approach does not take into consideration the impedance of the leads nor the electrode to skin resistance.

2.3.1 Impedance dynamics during swallowing

Single frequency impedance readings have been applied in the development of a bioimpedance tool to asses swallowing function. Kusuhara et al [2] concluded that bioimpedance changes correlated with normal movements during the pharyngeal stage. Lippitt [3] demonstrated a repeatable correlation with manometry readings. A single frequency measurement approach therefore is sufficient, as it is the immediate change in impedance that is of interest as opposed to tissue composition measurements. It is expected that a drop in impedance should be measurable as the pharynx and larynx contract during a swallow event, expelling air and making tissue to tissue contact.

2.4 Chapter Summary

Bio-impedance has been evaluated as a potential substitute for currently used invasive methods. A bio-impedance sensing device could potentially be used by the patient outside of a clinical setting, with little or no supervision by clinical staff. Bio-impedance is a function of frequency and thus a fixed single injection frequency would help with measurement consistency.

Chapter 3.0 System Design

3.1 Previous Work

Both previous implementations of the GUPLS hardware implemented a tetrapolar approach (see Section 3.2), with subsequent signal rectification, filtering, envelope detection and amplification. The second implementation of the GULPs hardware saw improvements to the constant current source as well as the addition of a second channel [3]. Lippitt concluded that further channels would need to be added.

Previous implementations of this project took a multi-frequency approach as a means to create separate channels, with 40 kHz and 70 kHz as the selected initial current injection frequencies (although 400 kHz and 700 kHz, and 4 MHz and 7 MHz were also investigated) [3]. This approach makes adding further channels very challenging as each channel is in effect a complete impedance measurement device, each with its own signal injection circuit, amplifiers, filters and detectors. Each signal injection circuit must have its power supply isolated from the rest [3] posing further design complexities. As discussed in Chapter 2, impedance is a function of frequency. This means that as further channels are added, each at their own separate frequency, each impedance measurement system would be located further away from a preferred injection frequency, and be measuring slightly differently weighted sections of the biological tissue impedance.

As opposed to a multi-frequency approach, this project aims to increase the number of channels by means of multiplexing. Multiplexing is a technique employed by which a single measuring system is routed to various target locations by a series of switches. In effect, the measuring system is taking impedance readings from a single location over a short sample time, before being routed to a new target location at the next sample time. This process continues until all channels are sampled, at which point the multiplexing cycle repeats. All four leads of the tetrapolar configuration need to be routed. As the switching is iterating (multiplexing) through channels at a very fast rate compared to swallowing dynamics, it creates the impression that all channels are active simultaneously after signal reconstruction.

Multiplexing has the benefits of using a single current injection frequency, therefore all channels can be making bio-impedance measurements at the most suitable current injection frequency. Only one measuring system needs to be implemented, which eliminates interference from other channels using injection frequencies that cannot easily be filtered out. The need for multiple isolated power supplies is also eliminated.

Multiplexing, however, does present its own challenges. The previously implemented and tested detector circuits cannot be reused in the new design. When multiplexing, the detector board must acquire a reading before switching over to the next channel. The low pass filtering used in previous implementations make the response time too slow for a multiplexing application. Previous detector implementations settle on a baseline or steady state, and amplify relatively fast deviations from that baseline. In a multiplexing system, a baseline or steady state needs to be stored, per channel. The current source must also be capable of settling on a steady current within the sampling period.

3.2 System Design

As per in previous implementations of GULPS [3] a tetrapolar measurement arrangement was selected (illustrated in Figure 8). This arrangement has the desired benefit of nulling out impedance changes caused by disturbances to the neck-connection electrodes, and only impedance changes within the region of interest are translated to changes in the measured voltage amplitude.

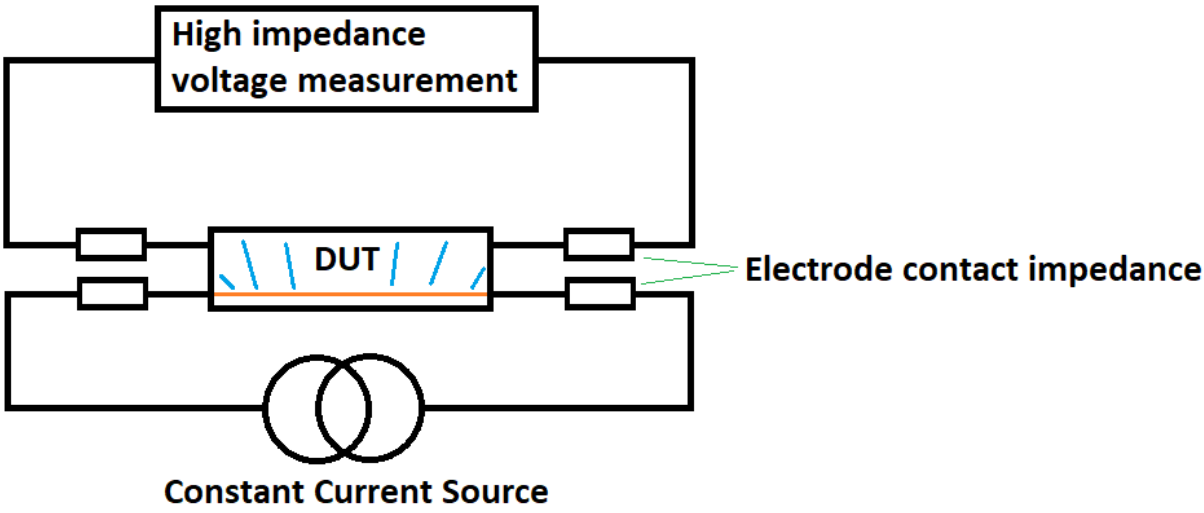


Figure 8 Tetrapolar sensing arrangement

The challenges in this approach arise from a need for a constant current source which can provide a stable sinusoidal ac waveform current. The current source must have low distortion with a sufficiently fast settling time to allow for multiplexing of the current source to the desired channel leads. The current source is configured to deliver 0.5 mA_{rms} as per previous design considerations, and should not contain any DC offset so as to not produce any form of electrolysis in the test regions.

The design of the current source is important as harmonics introduced at this stage will affect impedance measurements in the test region owing to the frequency dependency of the impedance being measured.

Several designs of the current source were trialled, with an initial differential-mode design being produced by the author which was subsequently used as inspiration for the final successful prototype that was designed and implemented by a final year engineering student.

Likewise, the design of the detector module faced similar challenges, as the detector must have a sufficiently fast settling time to allow for multiplexing. The amplitude of the voltage signal must be accurately measured and any small variations from the baseline signal amplified for channel output. Previous implementations of GULPs utilised full-wave rectification of the measured voltage, however this approach requires relatively aggressive low pass filtering to smooth out the half-waves to a sufficiently steady DC voltage. Such aggressive filtering made this approach unsuitable for multiplexing. Like the current source, the detector module went through several design iterations before settling on the final hardware implementation, which utilises both a maximum and a minimum peak detector to store the peak voltage in a set of capacitors. The peak detector capacitors are updated every half-cycle and the response time is typically only two or three cycles of the AC waveform.

Channel expansion is achieved by routing the current injection and voltage sensing electrodes to the various regions of interest. The multiplexing method, is achieved by the use of analogue switches. Figure 9 demonstrates a single current source and a single voltage measurement board, each requiring a pair of connections, or four electrodes per channel in total. Addressable analogue switches isolate the electrodes from the rest of the circuit, with only the desired channel having the switches in the closed position, and thus connected to the rest of the circuit. For simplicity not all electrodes are drawn due to the large number of connections. Electrodes corresponding to a channel must be grouped together and cannot be interchanged. In Figure 9, current is made to flow through the region of interest via the bottom pair of electrodes. The central circular object represents the region of interest within the neck (surface defined by the outer circular region). Changes in impedance correspond to a change voltage which is measured by the top pair of electrodes.

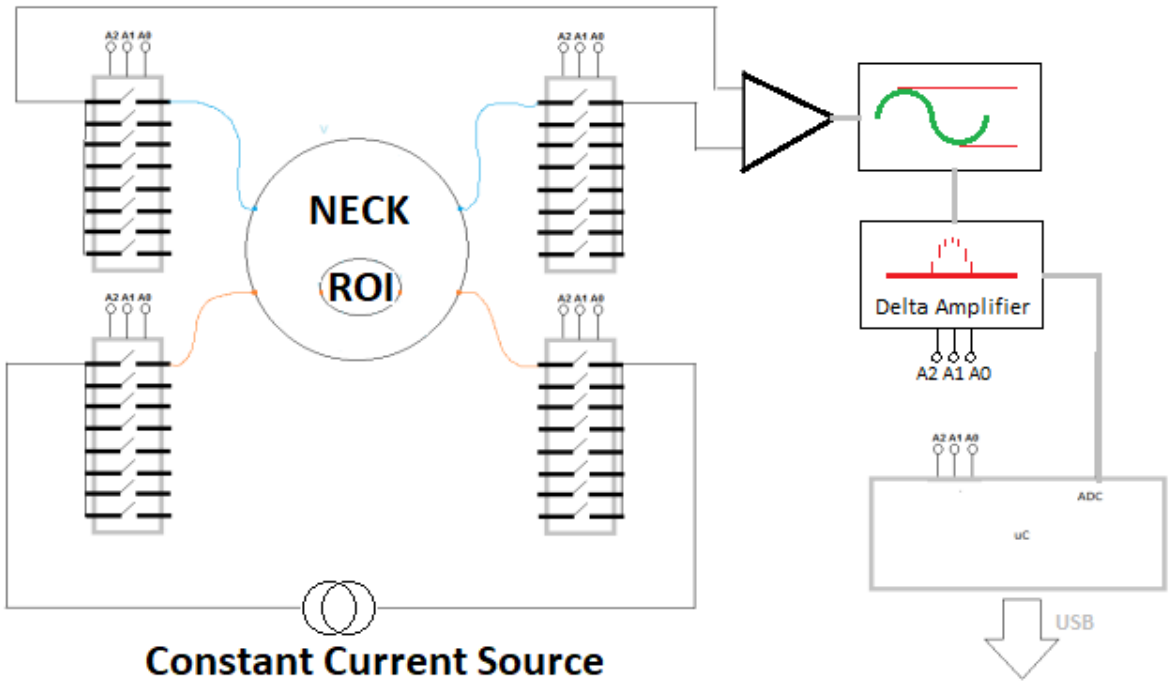


Figure 9 Hardware block diagram and Region of Interest, ROI

Multiplexing the current source, detector, and subsequent amplifiers was carried out under microcontroller control with configuration options via a Visual Studio application.

3.1 Concept block diagram

The block diagram for the multiplexed system implementation is split into two almost independent sub-systems for the source and detector module respectively. The common element for both systems is the multiplexer addresses as these must always be switched in synchronous operation with each other and thus originate from the same source. A microcontroller sets the address and controls the enable pins on each multiplexed stage. This prevents voltage swings during transition and subsequent settling times to ripple through to following channels. The microcontroller also configures the signal generator which drives the current source. Figures 10 and 11 depict the current

source modules and voltage sensing and processing modules respectively. The purpose of each block is discussed in the following subsections.

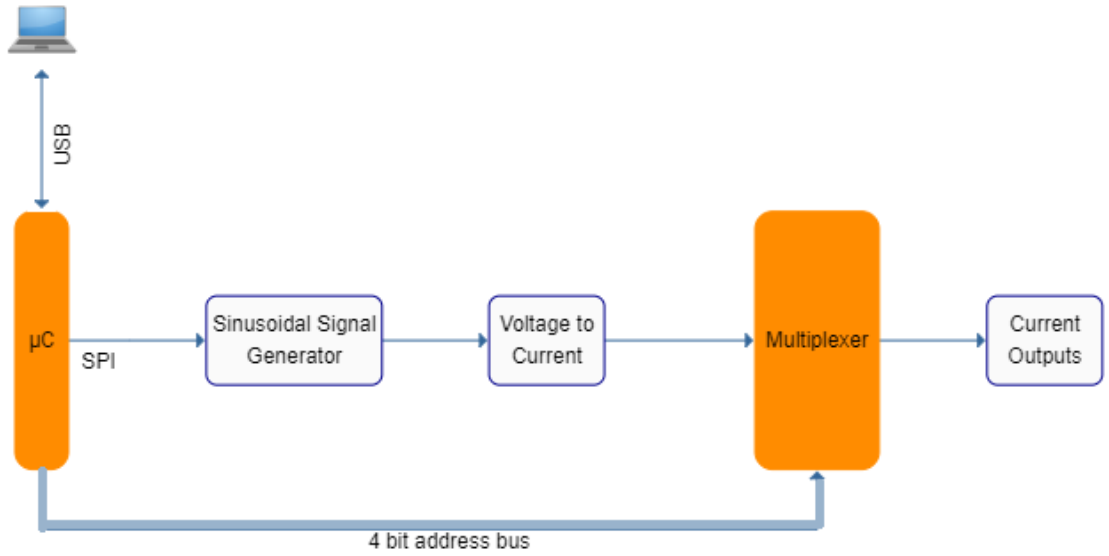


Figure 10 Current Source System block diagram

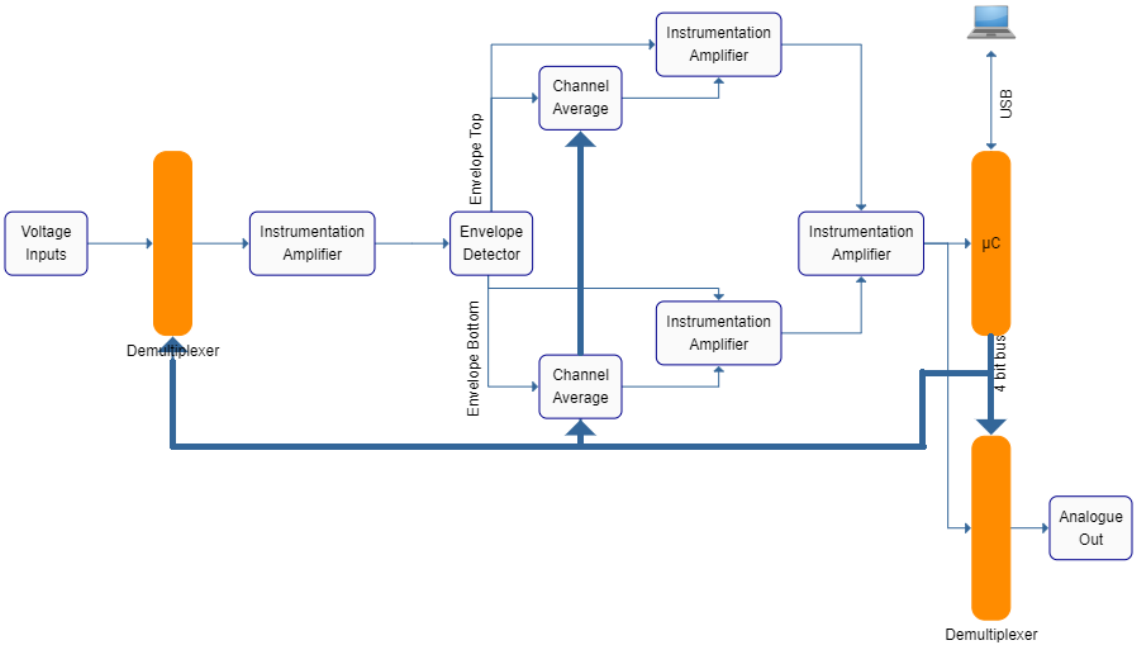


Figure 11 Voltage Measurement System block diagram

3.1.1 Multiplexing

Multiplexing allows the use of a single constant current source and voltage measurement system. The pair of leads from the current source, and the pair leads to the voltage measurement system are routed to different locations on the test subject's neck, each location is called a channel and only one can be active at a time. A microcontroller sets the address of the switches i.e. it selects the channel which will be connected to the channel source, and consequently which pair of voltage sensing electrodes are connected to the measurement system. As the microcontroller selects the channel, any measured signal is assigned to that channel. The switches also select the relevant averaging circuit. A further set of switches route the measured signal to an output pin, so that each channel can be viewed on an oscilloscope or analogue data acquisition system.

One of the main requirements on both the current source and voltage measurement system is that they settle on a steady and repeatable value in less time than the channels are being selected by the microcontroller. As the microcontroller is attempting to sample all channels at 1 kHz, and there are a total of 16 channels, the current source and measurement system need to settle in less than

$$t = \frac{1}{16 * 1\ kHz} = 62.5\ \mu s$$

In practice the hardware was designed to settle in half that time, or 31.25 µs.

The switching time is sufficiently fast that the system gives the impression to the user that all channels are read simultaneously.

3.2 Current Source System

The current source is a very important component. If the current source is not accurate and stable, the voltage measurement detector will not be able to recover meaningful signals. For this reason, a reliable current source had to be designed and implemented.

The final current source used in this project was designed and implemented by another student as part of a final year engineering project [5]. The current source is implemented using the differential operational amplifier AD8132 [7].

The final current source solution was developed from the concept of a prior current source designed by the author based on the principle of an H-bridge.

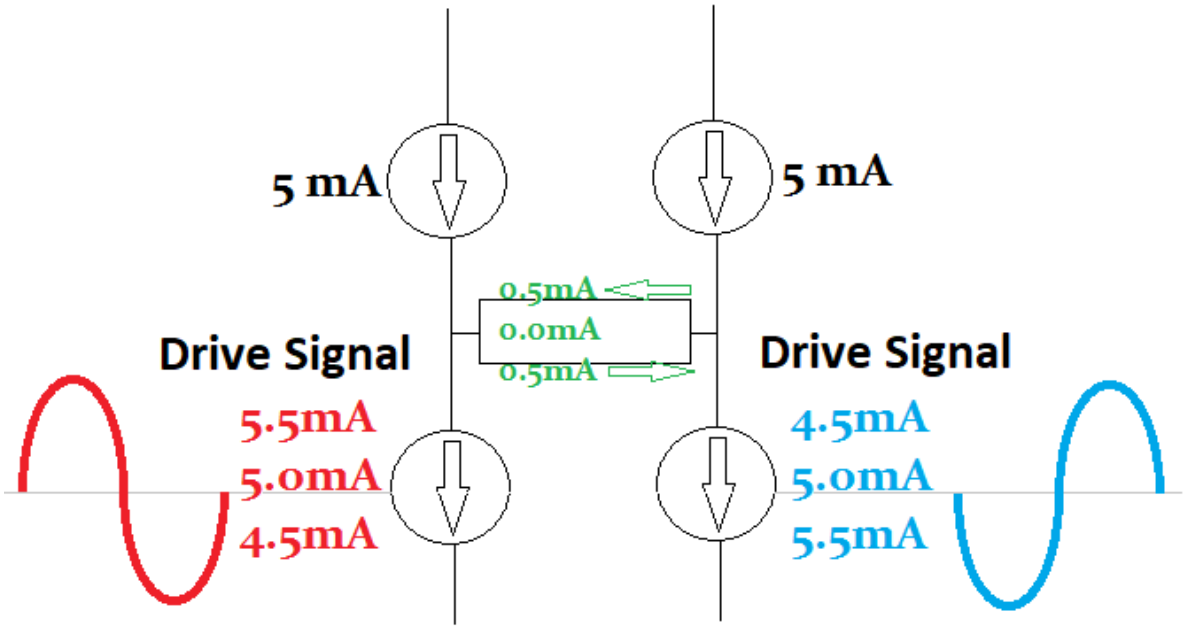


Figure 12 Constant current source concept of operation

The concept utilised is shown in Figure 12, where the top current sources are fixed at 5 mA, and the bottom current sinks can alternate between 5.5 mA and 4.5 mA in response to the voltage magnitude of a sinusoidal drive signal.

The current source is routed to the desired location to be measured by a pair of sixteen channel analogue switches. For simplicity, four channel analogue switches are depicted in Figure 13.

The output of the constant current source on the left is routed via the analogue switches to the desired location on the test subject. Switches, numbered 1 to 4 are addressable and only one switch per device may be active at any one time. The open switches isolate the current source from the regions not being measured. The switches are closed in sequence in very quick succession compared to the dynamics of swallowing, creating the impression that all channels are being measured simultaneously (after signal reconstruction).

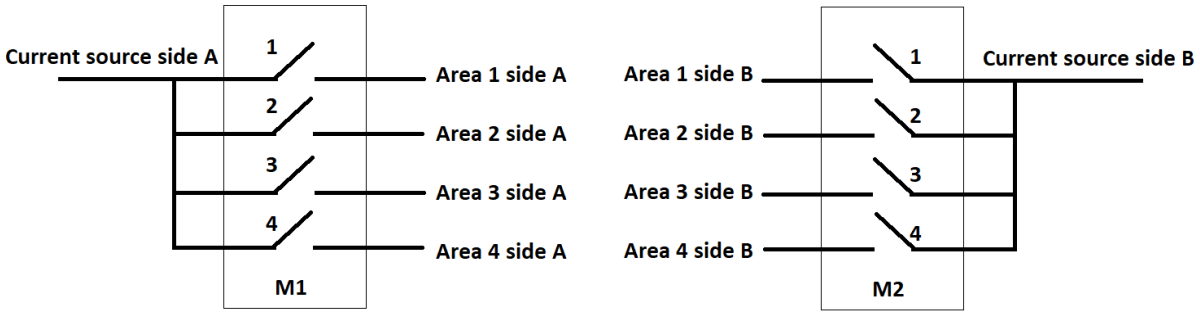


Figure 13 Multiplexing of constant current source

3.3 Voltage Measurement System

The voltage measurement system consists of a high impedance Instrumentation Amplifier (IA). As with the current source, a pair of sixteen channel analogue switches route the voltage measurement system to the desired region of interest and is demonstrated in Figure 14.

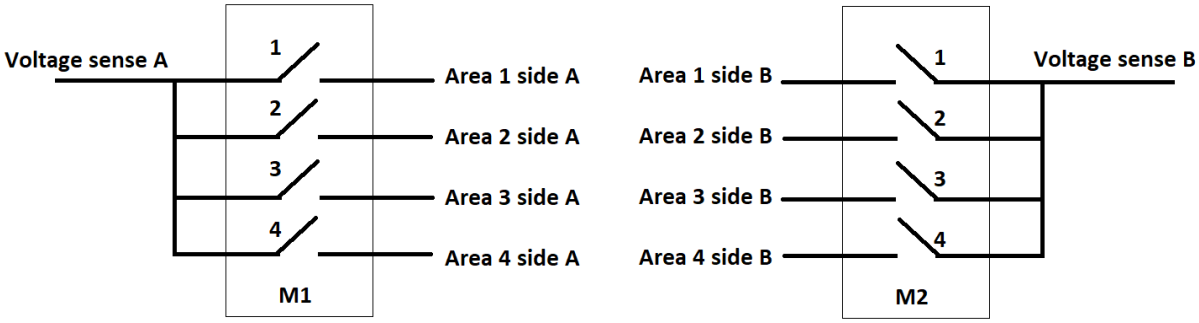


Figure 14 Multiplexing the voltage measurement system

It is essential that the active voltage measurement connections be routed to the same location as the active current source electrodes, i.e., all four connections that constitute a channel are not interchangeable with those of other channels.

The demultiplexed voltage signals are sampled by an IA. An IA is a high input impedance differential amplifier which amplifies the difference between its two inputs.

It is the *change* in signal amplitude per channel that must be accurately measured. This is the function of the amplitude detector.

3.3.1 High-impedance voltage sensing

An instrumentation amplifier serves the purpose of the high-impedance voltage sensor, its two inputs are switched to the corresponding region of interest, i.e., to its corresponding active pair of current injection electrodes.

The instrumentation amplifier amplifies the difference between its two inputs, this amplified voltage is added to a reference signal. In this way the measured signal is centred around a known voltage offset and it also raises the negative portion of the signal above the 0-volt ground.

A high impedance voltage sensor is important as it mitigates the effects of electrode to skin impedance. With negligible current flow, impedance changes in electrode to skin contact result in insignificant amplitude changes in the measured voltage signal.

3.3.2 The demodulator

The demodulator extracts the measured signal's top and bottom envelopes. The demodulator needs to respond quickly to changes in impedance as it is switched from channel to channel.

The demodulator was designed around the concept of peak detectors. A peak detector tracks the signal change in one direction only, if the sampled signal changes direction, then the peak detector will simply hold a steady value. Either maximum or minimum peaks can be detected this way, with the demodulator circuit being configured for either operation. Figure 15 demonstrates the concept of a peak detector.

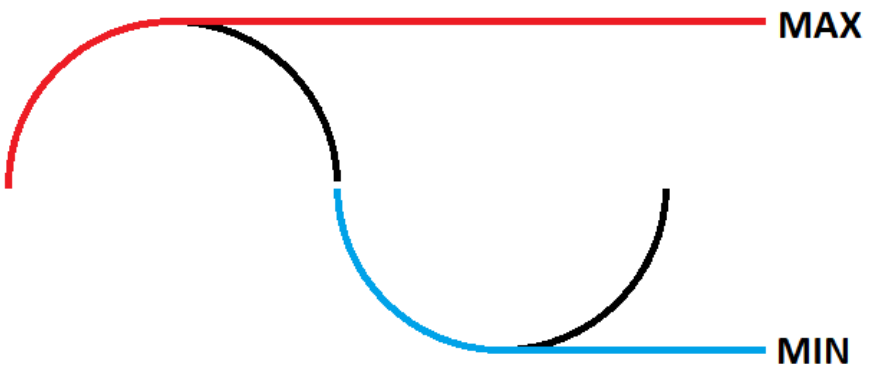


Figure 15 Function of a maximum and a minimum peak detector

When configured for maximum peaks, the demodulator tracks the signal whilst it is rising. When the signal changes direction the peak detector retains the maximum amplitude reached. Likewise minimum peaks can be tracked by re-configuring the peak detector circuit.

Tracking both maximum and minimum peaks on a cycle-by cycle basis yields the amplitude 'envelope' of the signal. A term commonly associated with AM radio. Figure 16 demonstrates the concept of the envelope of a signal. For the GULPS system, the envelope is created by the change in electrical impedance during a swallow event, and when switching from one channel to the next.

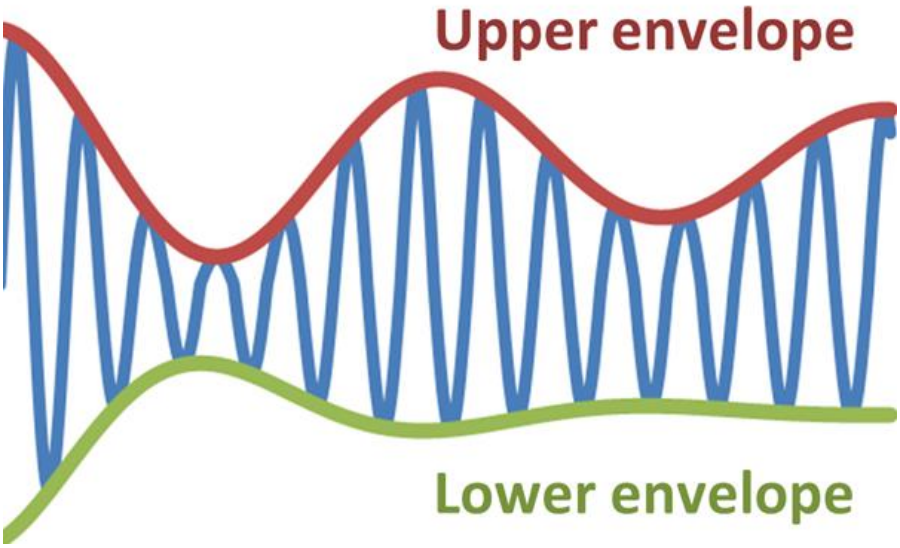


Figure 16 Upper and Lower envelopes

3.3.3 Voltage difference amplifier

The top and bottom voltage envelope signals follow parallel circuit paths. The envelope signal needs to be compared and amplified against a per-channel baseline, as small deviations from this baseline voltage represents the small impedance change expected during a swallow. Figure 17 demonstrates how the envelope signal is amplified against its own 10 second channel average; the averaging module is addressable to select the desired channel. This module is repeated for both the top and

bottom envelope signals.

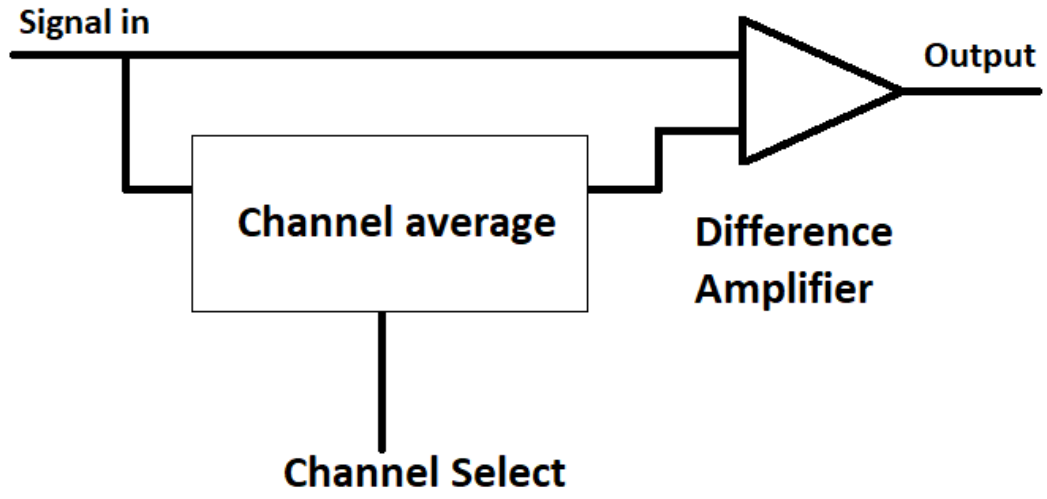


Figure 17 Channel amplified against average value

The amplified top and bottom envelope impedance change signals are subsequently re-recombined via a difference amplifier, figure 18.

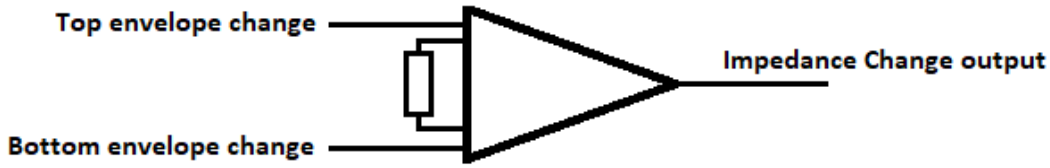


Figure 18 Combining top and bottom change signals

The output of the difference amplifier is at this point measured by the microcontroller. As the microcontroller knows which channel is currently being sampled the measured value can be assigned to the corresponding channel for display on the Visual Studio application.

3.3.4 Individual channel outputs

The final section of the design is the demultiplexing of the individual channels to their respective output pins. This is achieved by a final sixteen channel analogue switch (Figure 19). All analogue switches in the design share the same switch selector control bus.

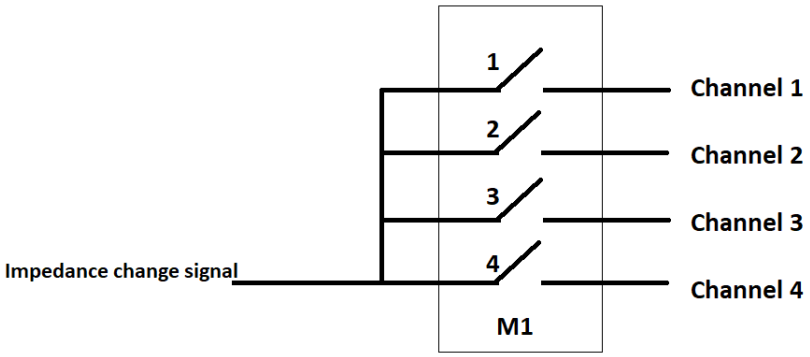


Figure 19 Individual channel outputs

The individual channel analogue outputs are necessary as these have more filtering than the sample taken by the microcontroller for displaying on the Visual Studio application. Physical analogue outputs are also needed for comparison purposes to pharyngeal manometric sensors, which also provide analogue outputs.

Chapter 4.0 Hardware Implementation

4.1 Current Source

The new current source was initially designed around the concept of an H-bridge. Complementary drive voltage signals, 180° out of phase, cause an imbalance in the bridge and thus a bidirectional constant current can be made to flow through a load.

Figure 20 shows the circuit schematic of the discrete device current source solution as well as a photograph of the constructed system.

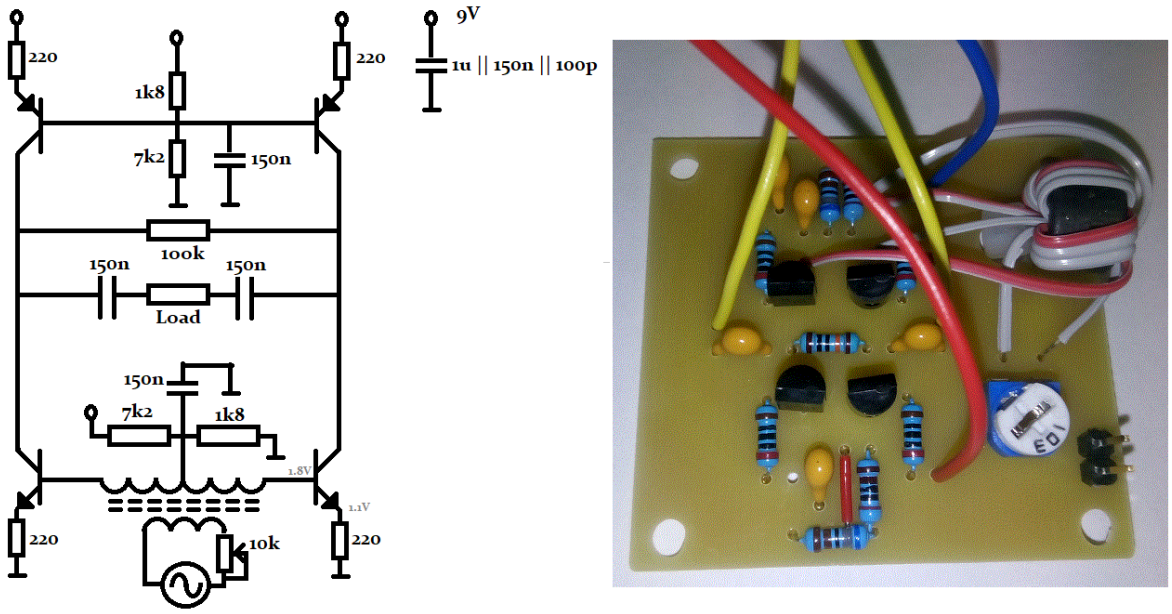


Figure 20 Circuit diagram and implementation of constant current source

When tested with resistive loads the current source showed low distortion, fast response times and linearity, all of which are requirements of the source. Figure 21 shows an example of the differential current produced when connected to a load resistance of 1 kΩ. This current is at a frequency of 2 MHz (frequency chosen to show adequate performance is possible at the higher end of the expected preferred measurement frequency), and has a differential amplitude of 500 mA.

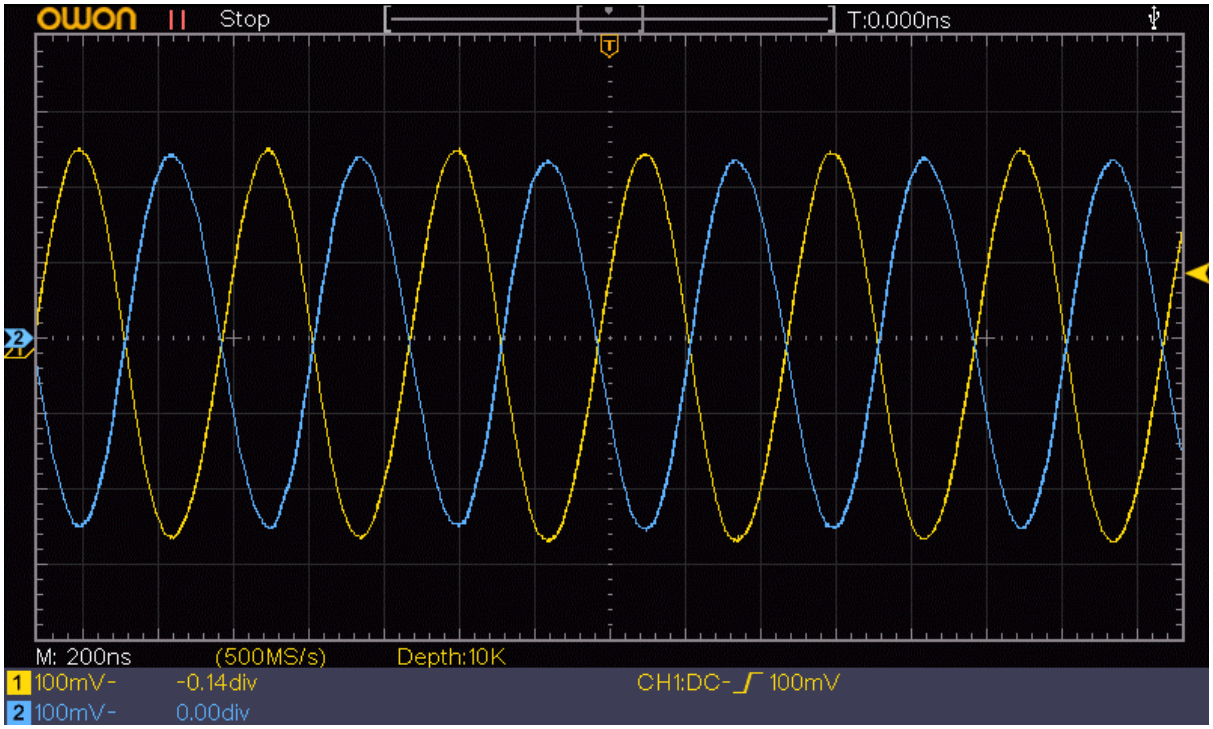


Figure 21 Balanced bi-directional constant current source testing

However, the source did not perform well on loads which more closely resemble a human test subject, such as saline solution. Under these conditions the source demonstrated unacceptably high distortion.

In the final current source solution, the primary signal generation is accomplished by using an AD9850 DDS sinusoidal signal generator module [8]. The associated microcontroller sets the desired frequency. A voltage to current converter delivers a bidirectional sinusoidal constant current with a Root Mean Square (RMS) value of 0.5 mA [3]. The voltage to constant current converter was designed by another student as part of a final year project. Figures 22 and 23 represent the schematic and final hardware implementation respectively.

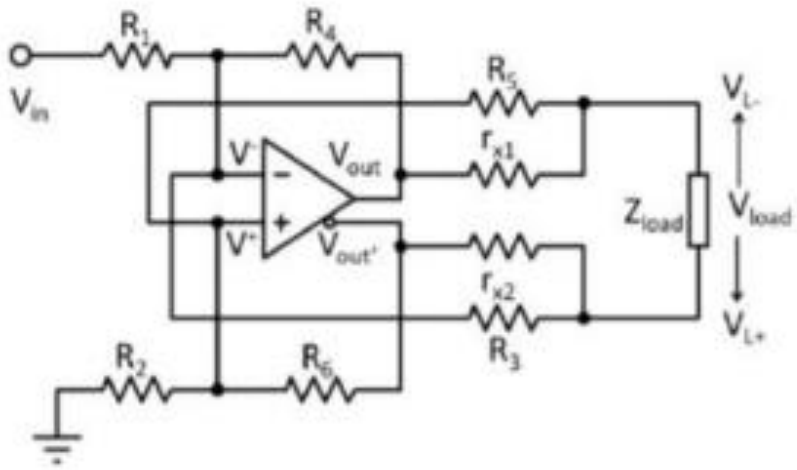


Figure 22 Improved constant current source schematic

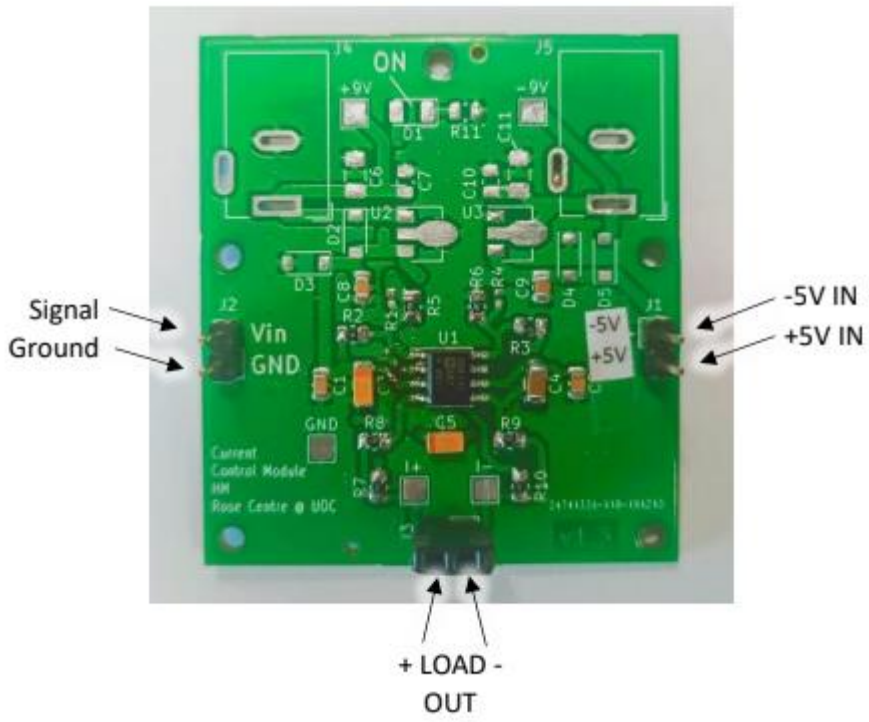


Figure 23 Constant current source used in the last implementation of the GULPS hardware

A pair of 74HC4067 [9] addressable analogue switches route the current source to the selected channel electrodes. This device was selected for its low on resistance of 70 Ω, low 'off' leakage current of 8 µA, fast 500 ns switching time, and frequency response up to 89 MHz.

Figure 24 demonstrates the functional block diagram of the current source module, consisting of the microcontroller, signal generator, constant current source, and multiplexing switches.

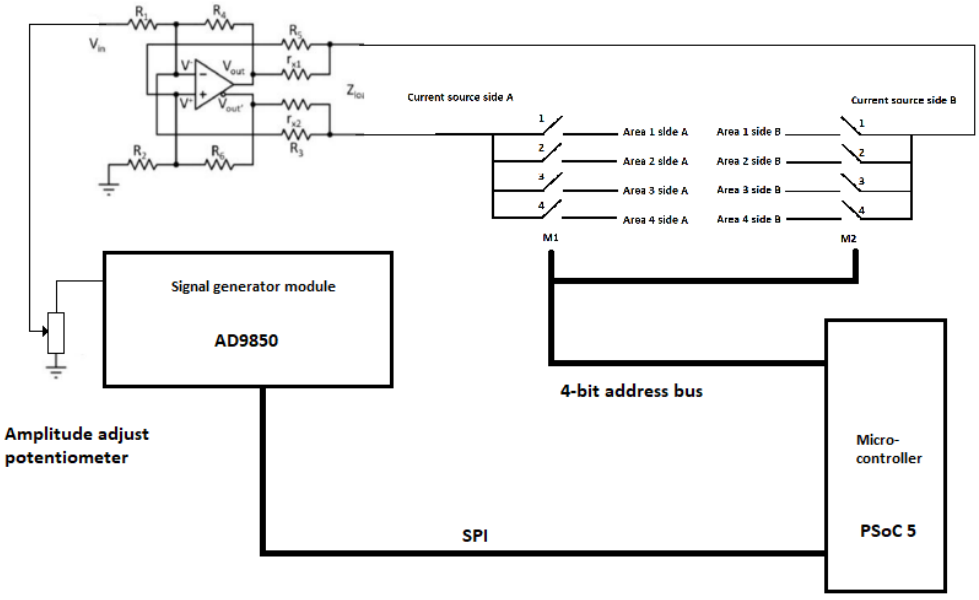


Figure 24 Functional block diagram of complete current source module

The complete hardware implementation with Cypress PSoC microcontroller, DDS signal generator, constant current source and multiplexing switches is shown in figure 25.

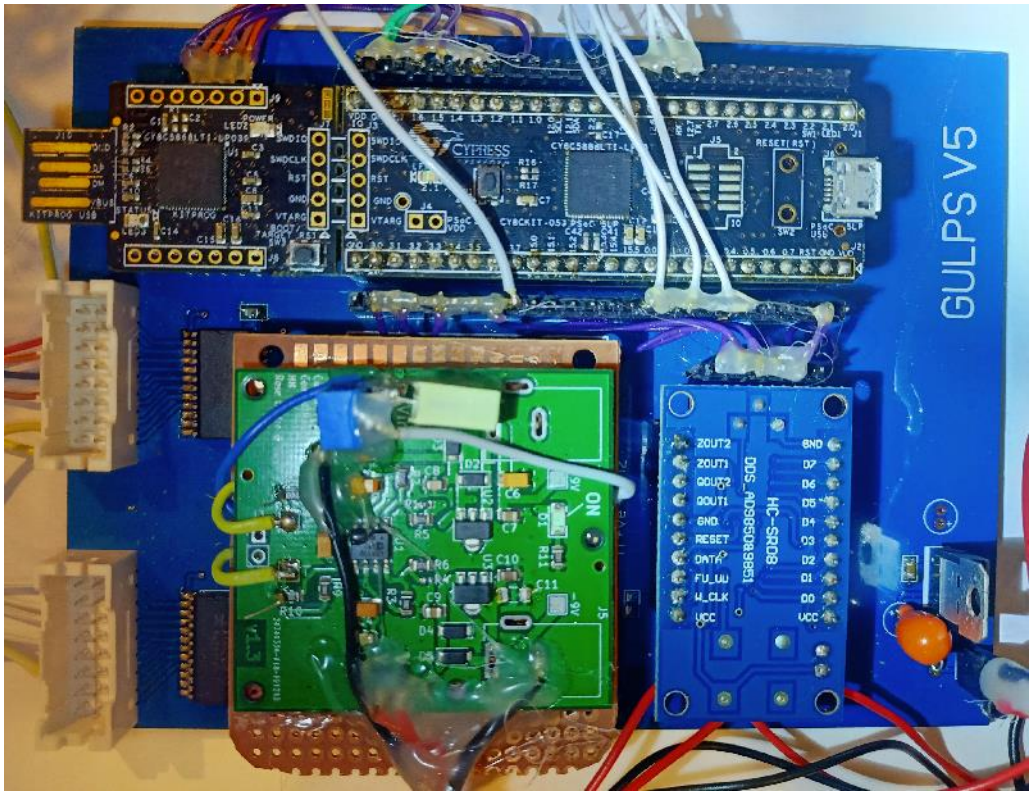


Figure 25 Final implementation of current source module

4.2 Voltage sensor input stage

Like the current source, a pair of 74HC4067 analogue switches route the voltage sensing Instrumentation amplifier (IA) to the selected channel. The IA selected is the AD8421 [10], this device was selected for its bandwidth of 2 MHz at a gain of 100, high input impedance of 30 GΩ, and low power consumption. The device also has an easy-to-use voltage reference input pin, which sets the baseline voltage of the output pin. Figure 26 shows a schematic of the implemented IA arrangement.

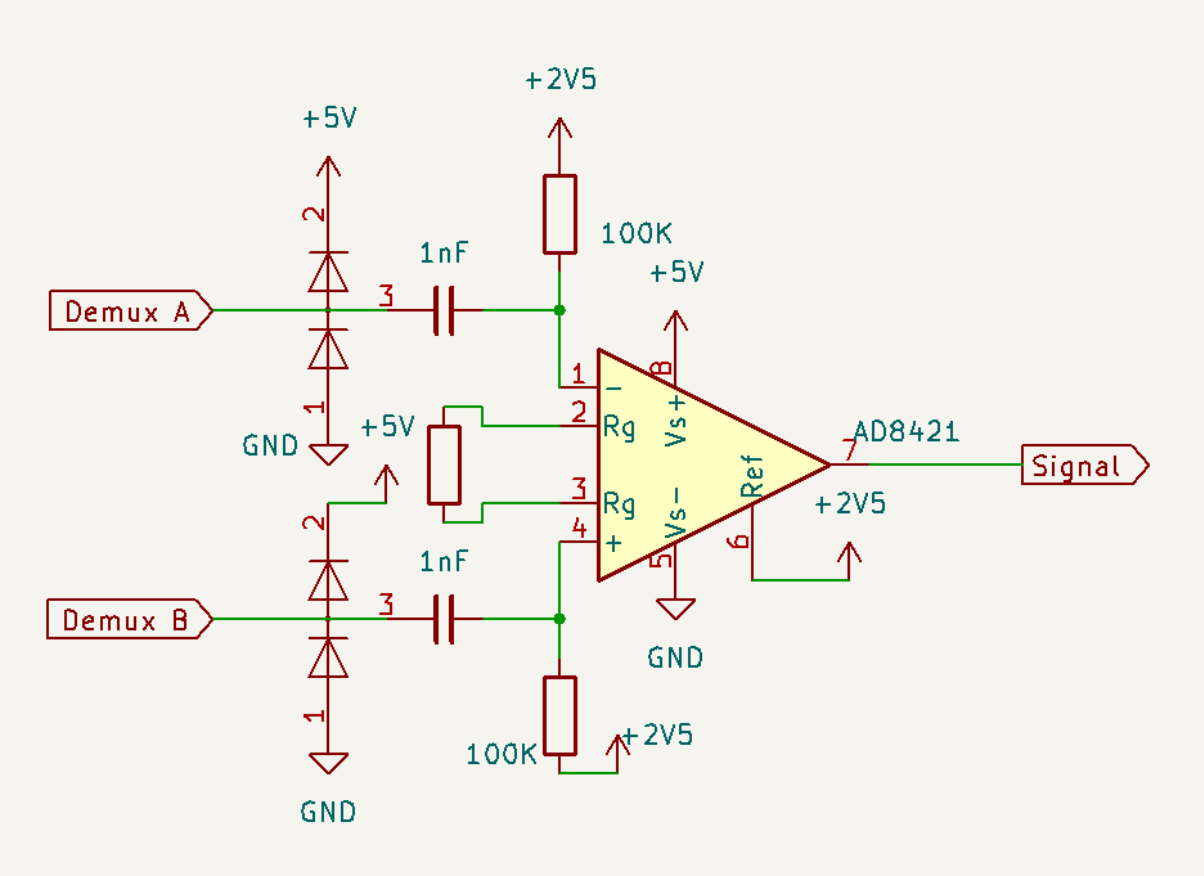


Figure 26 Voltage input section, AD8421

4.3 Amplitude detector module

As discussed in the design System Design Chapter, the amplitude detector module is designed around peak detectors. In its simplest form a peak detector can be implemented from a single diode and capacitor. The orientation of the diode determines the mode of operation, either sensing for maximum or minimum peaks. The system requires that both maximum and minimum peaks be detected. The detectors also need to be reset for each sample as subsequent peaks may be of lesser amplitude than a preceding peak, and the circuit needs to accurately track these to provide true envelope signals.

The design requires the polarity of the input signal to control the peak detect function. This signal, called the phase signal, is generated by a voltage comparator MAX941 [11]. This device was selected for its fast propagation delay of 80 ns, and its 1 mV offset sensitivity. Figure 27 shows how the comparator is configured to generate the required phase signal. The input voltage is compared to mid-rail 2.5V.

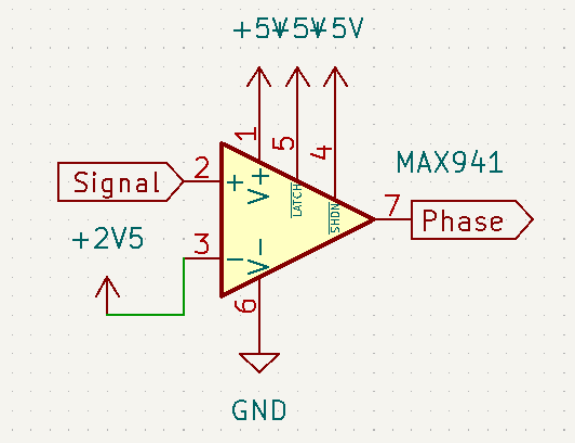


Figure 27 Phase signal, MAX941

Figure 28 shows the schematic for the envelope detector. The op-amps are configured as voltage buffers. A series of two pole analogue switches select the mode of operation and route the peak voltage signals to the corresponding storage capacitor.

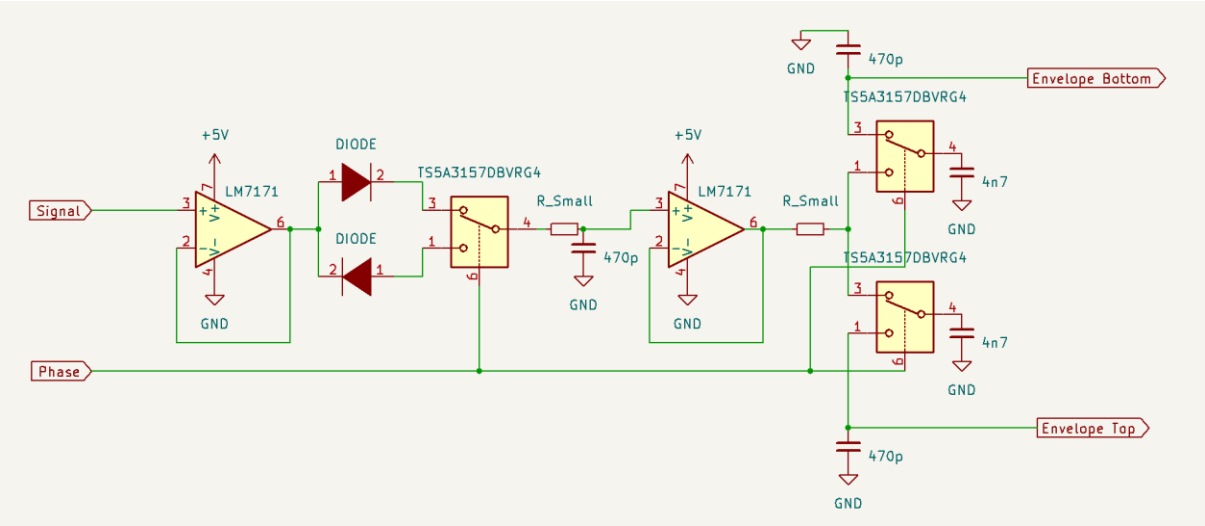


Figure 28 Amplitude detector generates both top and bottom envelopes of the input signal

The design is an implementation of a buffered diode and capacitor peak detector, with the polarity of the diode being switched by the phase signal. The peak voltage is buffered again and used to charge a temporary storage capacitor. The final voltage is switched again to the corresponding maximum or minimum signal level capacitor at the end of the phase signal.

The small value series resistors of 10 Ω were necessary in practice to protect the analogue switches from peak currents when switching to an opposite peak charged capacitor. This detector produces a top and bottom envelope which closely tracks the AC measured signal. Voltage buffers between stages mitigate the cumulative effect of the in-series resistors and load capacitors. The LM7171 [12] operational amplifier was selected for this function for its high bandwidth. The switches used are two-pole, single-side TS5A3157DBVRG4 [13] analogue switches, selected for their high speed and low on resistance. The output capacitors are buffered using FET input Operational Amplifiers AD822 [14] selected for their high input impedance.

Figure 29 depicts the signal path for an envelope signal, in this case the top envelope. The currently selected channel's signal is amplified against the channel's average. In the top path the signal, via a low pas filter, is wired to one side of the difference amplifier. The other side is the channel average, which comes from the signal path being routed via a more aggressive low pass filter. Each channel average must have its own storage capacitor. This capacitor retains the channel average while the channel is not selected and is isolated from the envelope signal.

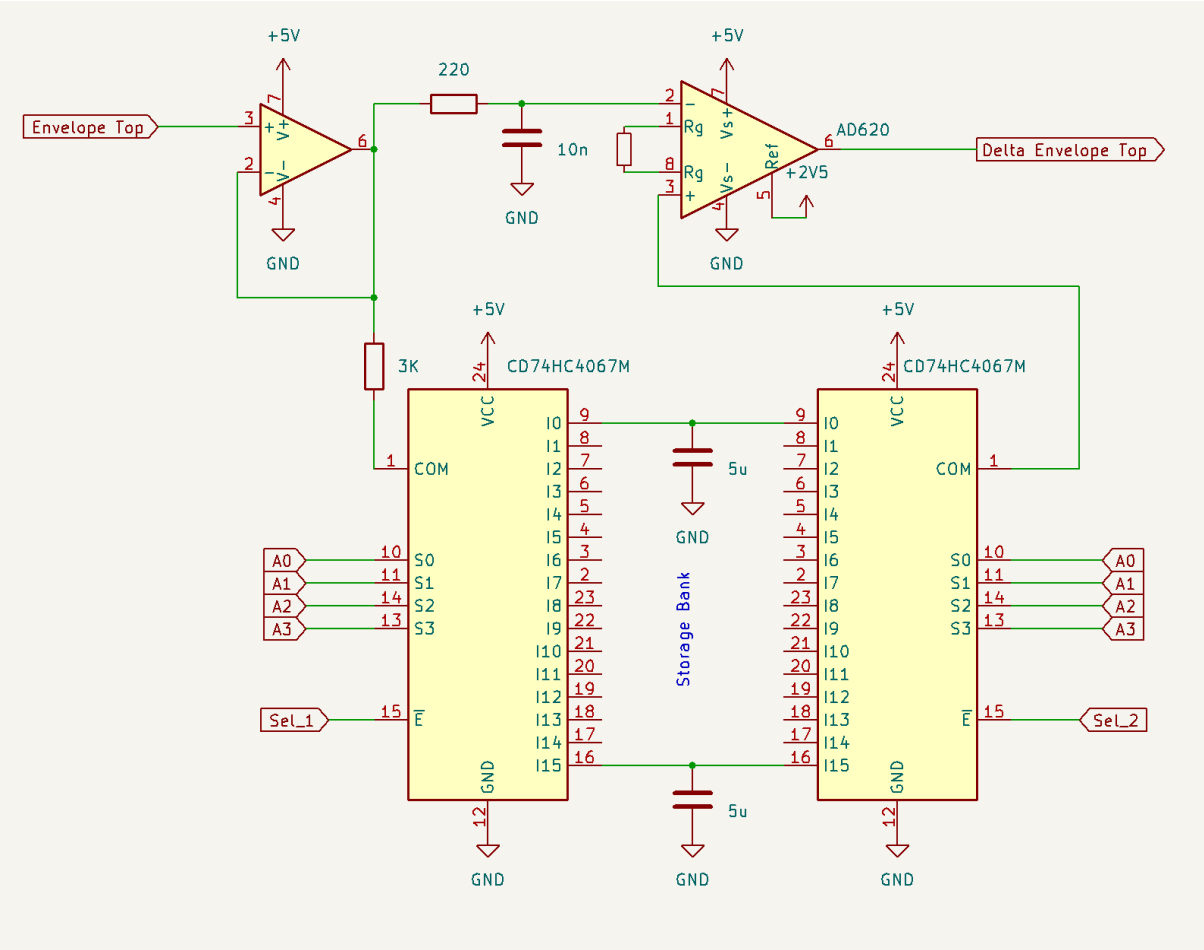


Figure 29 Difference amplifier, relative to 10 second channel average

The immediate signal path is via the top low pass filter, and the average signal path utilises a multiplexed capacitor bank so that each channel may retain its own independent baseline value.

After amplification against each channel's baseline reading, a ten second average, the top and bottom envelope 'delta' (change in amplitude) signals are recombined into a single 'impedance change' signal, figure 30. For this purpose, an AD620 [15] was selected, the device has 120 kHz bandwidth, low noise, and low power consumption. The ten second average is created by switching the currently active channel signal via a 3 kΩ resistor into a 5 µF capacitor. According to the time constant equation:

$$T = RC = 3\ k\Omega * 5\ \mu\text{F} = 15\ \text{ms}$$

If five time constants are required to charge the capacitor then:

$$5 * 15\ ms = 75\ ms$$

However, due to multiplexing the RC is only enabled 1 / 16 th of the time, and only then for half of the period to allow for settling. Thus:

$$75\ ms * 2 * 16 = 2.4\ s$$

Using these values an averaging time of 2.4 seconds should be expected. In practice however, the averaging delay was measured to be the desired 10 seconds. The cause of this delay was not identified and it could be one of the contributing factors to the less than desired sampling rate achievable.

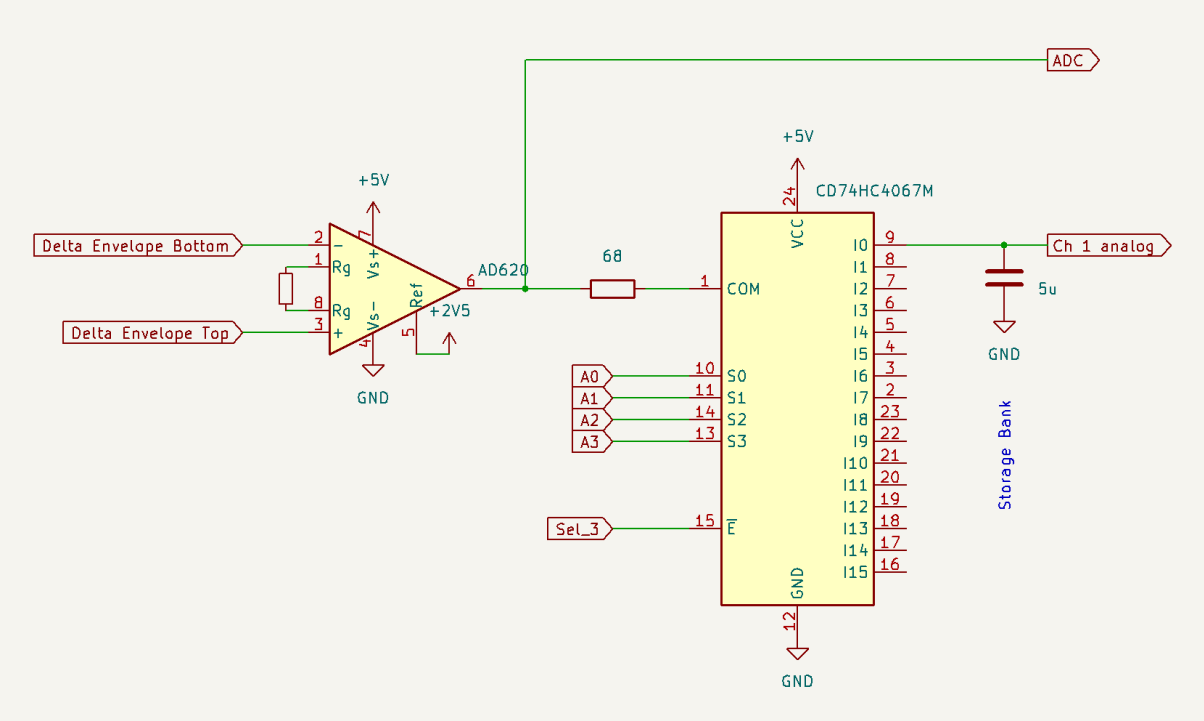


Figure 30 Recombining the envelope change signals

The resulting signal can be directly read by the microcontroller and assigned to the corresponding channel trace on the Visual Studio application. At the same time the 'impedance change' signal is switched via a resistor to a storage capacitor to produce a filtered analogue output signal. This signal can be viewed on an oscilloscope or analogue data acquisition system.

Chapter 5.0 Testing and Preliminary Results

This chapter describes the testing of the individual hardware modules, and finally the testing of the system as a whole on a simulated load.

5.1 Testing the current source module

Figure 25 shows the multiplexer transitioning from channel 1 to channel 2. Each channel is connected to an 800 Ω resistive load. The current source is configured to deliver 0.5 mA at 2 MHz. The horizontal scale is 100 ns per division, and 200 mV per division for the vertical scale. As can be seen in the figure, the channel transients last for less than 100 ns, and the deselected channels are effectively isolated from the current source.

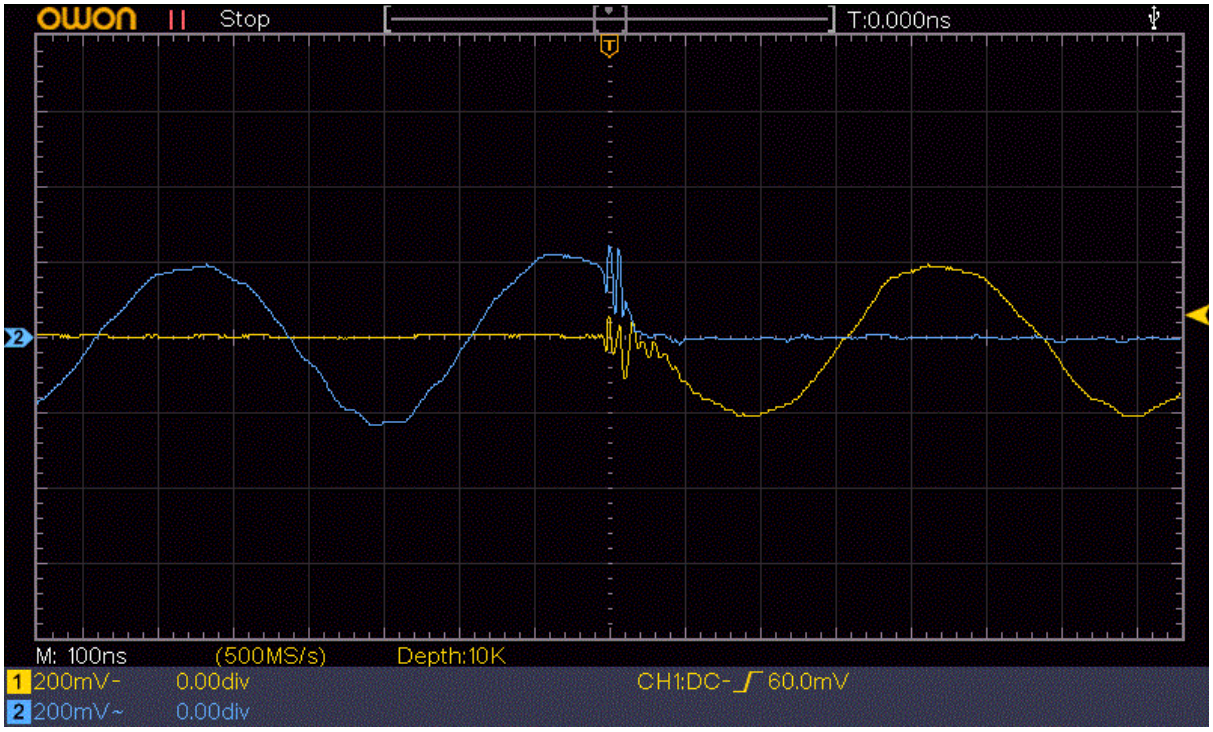


Figure 31 Testing the current source and multiplexer. 100ns/div, 200 mV/div

5.2 Testing the envelope detector

To test the envelope detector board, the current source module is configured to source 2 MHz constant current, multiplexing is disabled, i.e., constantly set to a single channel. The load is a 1 kΩ resistance across which the voltage sensing electrodes are connected. The first important step for the operation of the envelope detector is to generate a phase signal from the incoming signal, this is achieved by a comparator. Figure 32 demonstrates how a square wave is generated by the comparator MAX941 [10], representing the upper and lower portions of the signal.

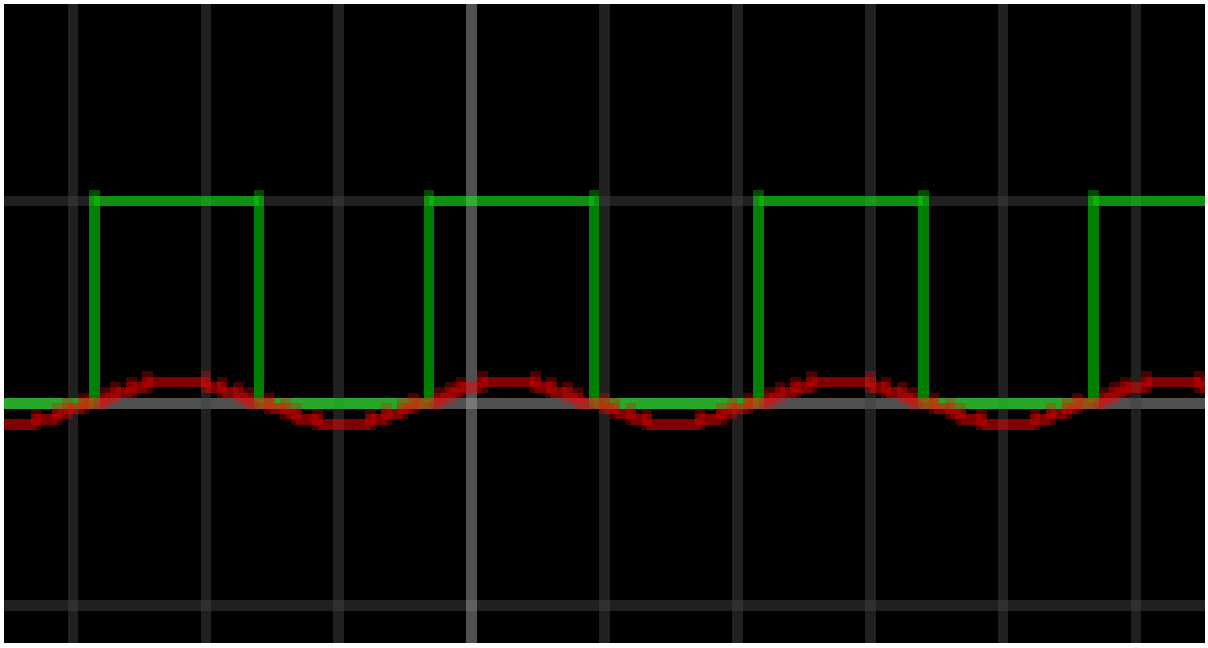


Figure 32 Generating the phase signal, green, from the input signal, red. 5 volts, 2 us per division.

Figure 33 depicts the input signal in green, the voltage signal on the first storage capacitor is shown in red. It should be noted that a loss in amplitude occurs due to the voltage drop across the detection diodes, Schottky diodes were selected for their fast-switching speeds and low 0.2 V drop, however this loss must be compensated for. The gain of the voltage sensing instrumentation amplifier must be set to yield a signal of about 1 V peak to peak. This gain is likely to change depending on the impedance characteristics of the load.

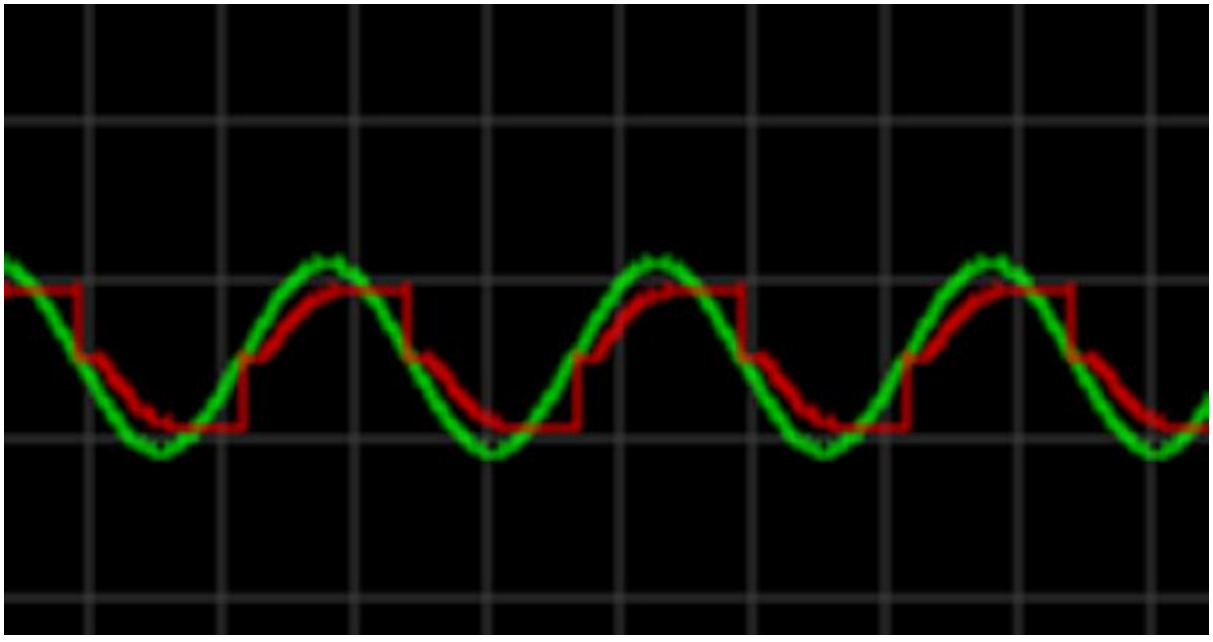


Figure 33 Signal input vs first peak detection storage capacitor. 1 volt, 2 us per division.

The red trace can be seen tracking and then retaining the peak voltage for the duration of half the waveform of the input signal. This is due to the polarity of the phase signal quickly changing the mode of operation on the peak detector at the end of each phase. Simultaneously, the second stage

peak storage capacitors are disconnected from the peak detector, thus splitting the red trace into top and bottom traces (figure 34 demonstrates this).

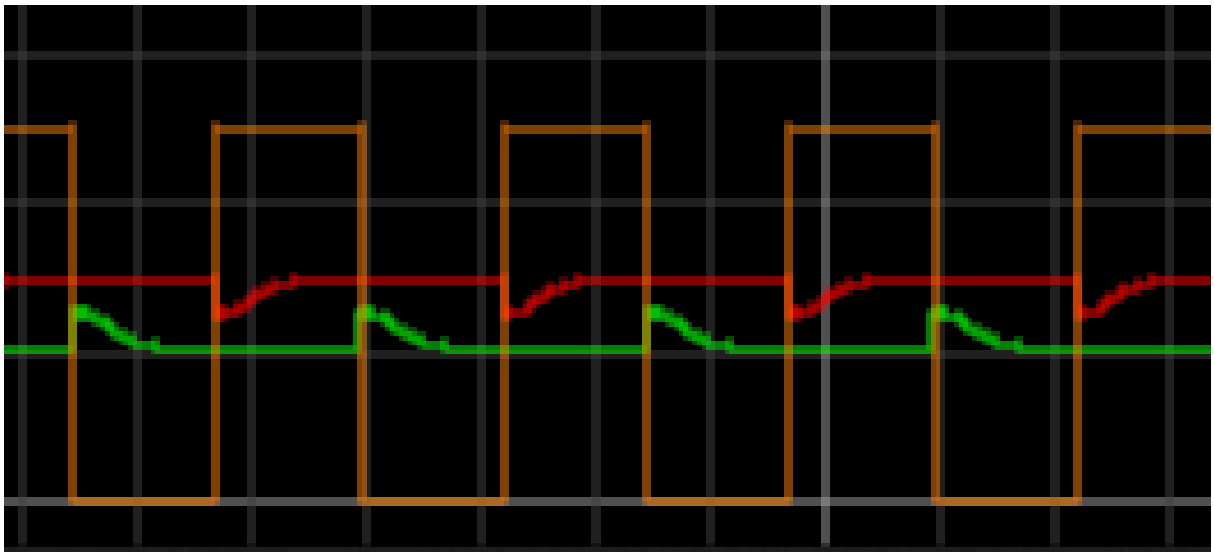


Figure 34 Top and bottom peak storage capacitors. 2 V/div, 2 µs/divi.

During the high period of the phase signal, orange, the secondary positive peak capacitor tracks the rise in amplitude of the peak detector. At the end of the positive period as the peak detector is switched to minimum peak mode, the second stage capacitor is isolated from the detector circuit, thus retaining the maximum peak voltage. The minimum peak second stage storage capacitor works in the same way, during the low period of the phase signal.

Finally, the second stage storage capacitors transfer their stored voltage to the corresponding third stage storage capacitor. The third stage capacitors are only updated with peak voltage readings, and thus produce steady envelope signals with none of the transition voltages present. Figure 34 demonstrates the top (red) and bottom (green) envelope signals produced from an input signal which is varying in amplitude vs the phase signal (orange).

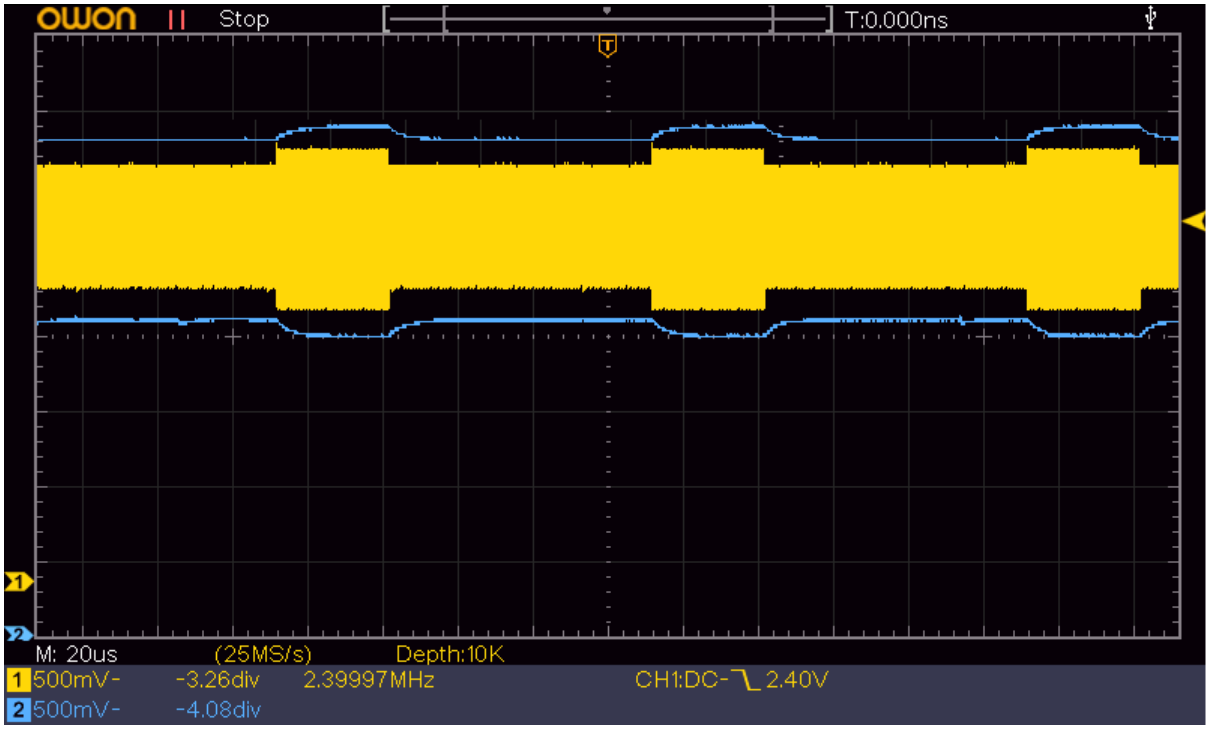


Figure 35 Top and bottom envelopes (blue traces) produced from input signal varying in amplitude (yellow trace). 500 mV/div, 20 µs/div.

The delay in the transition of the envelope signals is introduced by the resistance of the analogue switches, as well as the 10 Ω current limiting resistors, and storage capacitors forming a time constant. However, this transition time is below 10 µs (less than half a division), figure 35.

$$f = \frac{1}{T} = \frac{1}{10\ us} = 100\ kHz$$

The 100 kHz bandwidth is significantly higher than the 32 kHz maximum bandwidth requirement. The bandwidth of the current source is too fast to measure on this scale.

5.3 Testing the averaging circuit

Each channel must be amplified against a reference signal. The reference signal is generated by taking a ten second average of the desired channel, generated by switching the selected channel to its corresponding low pass filter. The average value is stored in a capacitor, two per channel (one for the top envelope and a second for the bottom envelope). Multiplexing and demultiplexing was required for this step as each channel must have its own averaging filter.

This approach greatly simplifies how the GULPS hardware will be used, as there are no gains and reference voltages that must be configured per channel.

Figure 36 shows two adjacent channels (yellow), the averaging circuit is only enabled for half of the duration of the sampling period, this is to avoid making transition voltages part of the average channel voltage. The figure demonstrates a transition voltage spike at the start of the first channel, which is not included in the average value. The figure also demonstrates how the channel average (blue) for the first channel is slightly higher than the average for the second channel. A larger

amplitude channel means that the minimum peaks are lower down, and thus have a lower channel average voltage for the lower envelope signal. The top envelope signal has a separate averaging circuit.

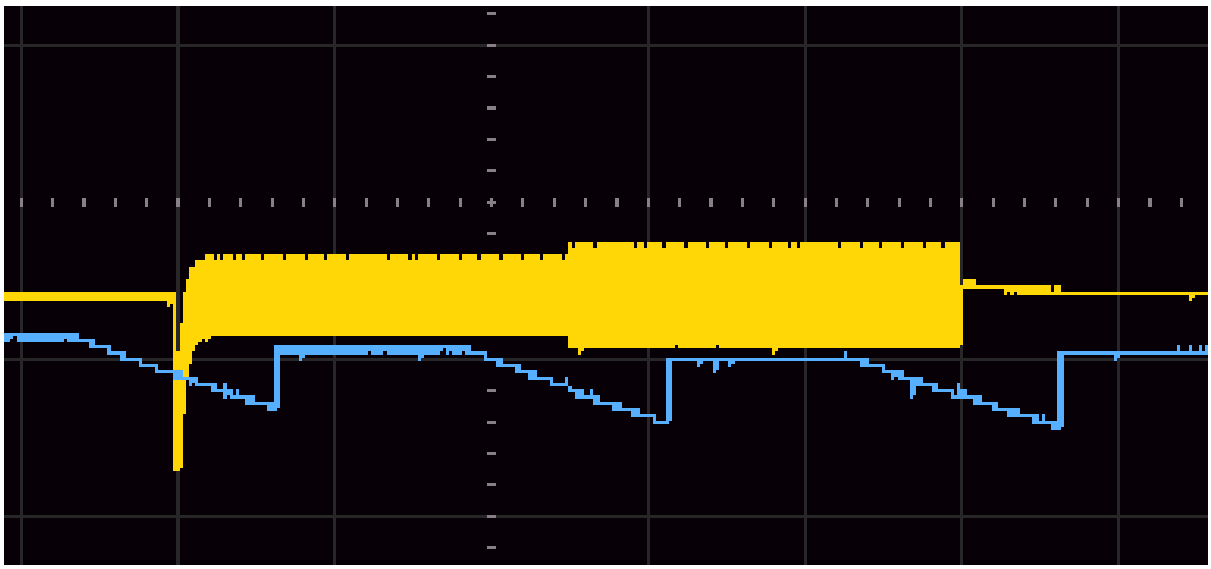


Figure 36 Voltage present at averaging capacitors. 2 volts, 10 us per division.

The second signal required is the current channel value, this signal is first filtered using an RC low pass filter LPF.

$$Fc = \frac{1}{2\pi RC} = \frac{1}{2\pi * 220\ \Omega * 10\ nF} = 72\ kHz$$

An instrumentational amplifier AD620 [15] is used to amplify the difference between the currently selected channels live value against its stored channel average value. The AD620 has a bandwidth of 120 KHz. The filtered channel signal, at 72 kHz, is well below this limit yet significantly higher than the required minimum 32 kHz bandwidth required to sample the 16 channels at a sufficiently fast rate.

Figure 37 shows the 72 kHz limited bottom envelope (blue) as measured on channels one and two. Different value loads were used to highlight channel switching. Other channels are disconnected and thus produce a flat trace (yellow).

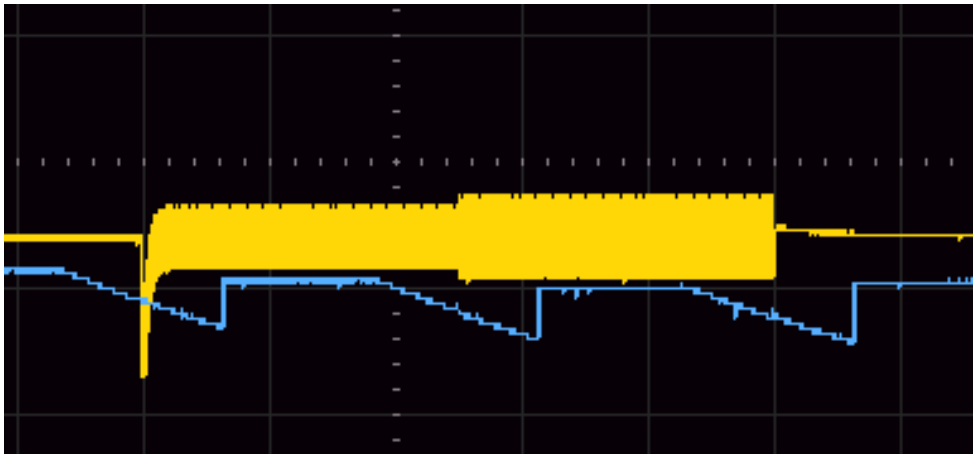


Figure 37 Live reading from channel, bottom envelope. 2 volts, 10 us per division.

The top envelope signal follows the same process, the resulting amplitude variation signals are re-combined by a third operational amplifier AD620.

The resulting impedance change signal is now sample by the microcontroller and assigned to the corresponding channel data buffer and sent to the Visual Studio application for displaying on a live trace.

5.2 Live channel output

A final demultiplexer switches the current channel value to a buffered (LM324 [16]) storage capacitor, again a resistance is used in conjunction with the storage capacitor to form a 10 Hz LPF per channel. This RC filter is very aggressive to switching and carrier frequency noise whilst presenting a fast enough response for the signal of interest.

All multiplexers 74HC4067 [9] are enabled in sequence by the controller IC, PSoC [17], to minimise noise and to allow the signal enough settling time before transferring to the next signal processing stage.

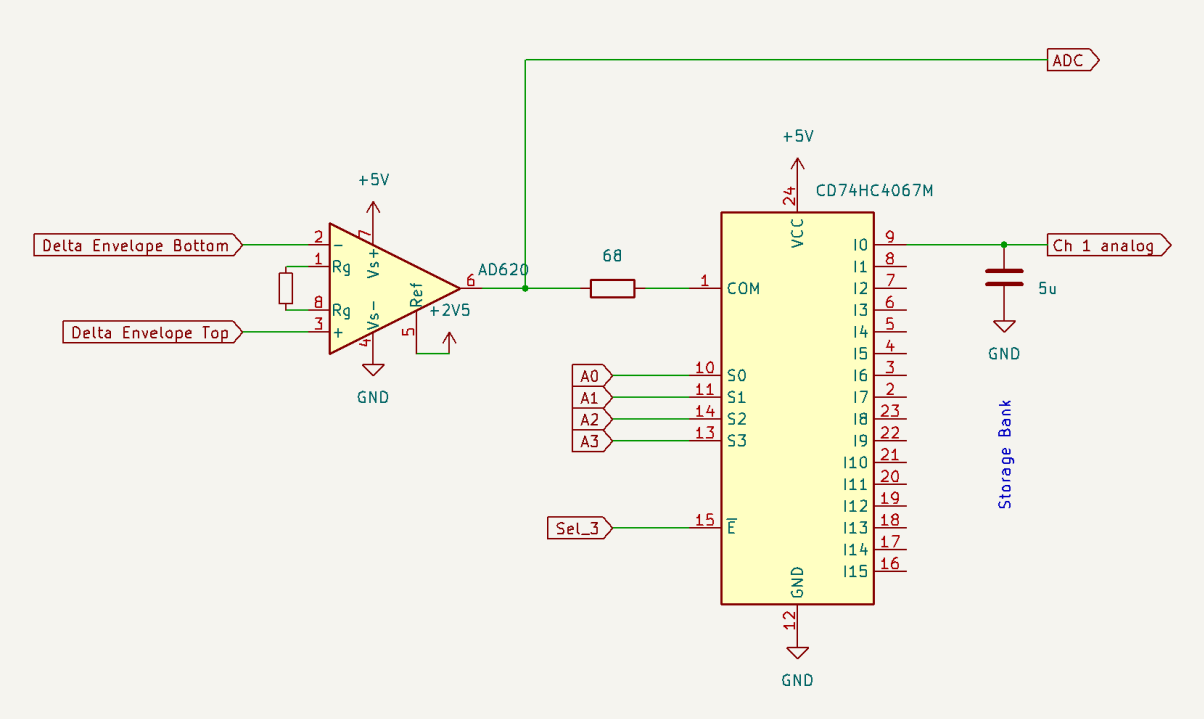


Figure 38 Output demultiplexer with 10 Hz LPF and buffer.

Figure 38 depicts the schematic for the output multiplexer with storage capacitor, in the practical implementation an operational amplifier LM324 [16] buffers the output voltage.

The complete schematic for the detector board can be found is in Appendix C.

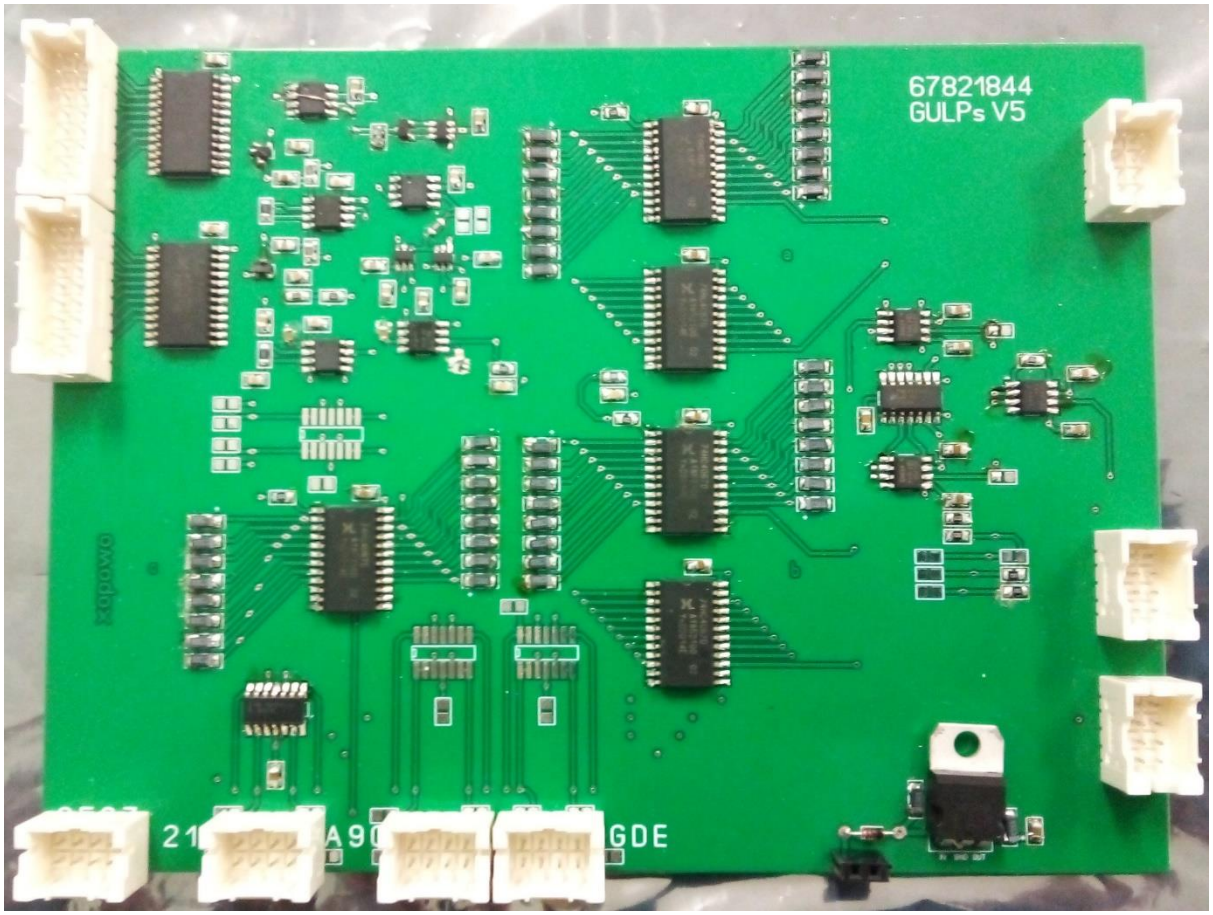


Figure 39 Detector board practical implementation.

A PCB was developed by the author and populated to implement the design. Figure 39 demonstrates this board which contains the demultiplexers, instrumentation amplifiers, envelope detectors, ten second averaging filters, difference amplifiers and output demultiplexer.

A Programable System on Chip (PSoC) was selected as the microcontroller for this project. The PSoC [17] facilitates bi-directional bulk data transfer over USB, a 32-bit ARM Cortex-M3 CPU core as well as reconfigurable analogue and digital blocks. The hardware configuration of the PSoC, figure 40, sets the channel address, reads the analogue to digital converter, sequences the multiplexers, configure digital potentiometer values and handles bidirectional data transfers to the Visual Studio PC application.

The firmware for the PSoC is in Appendix A.

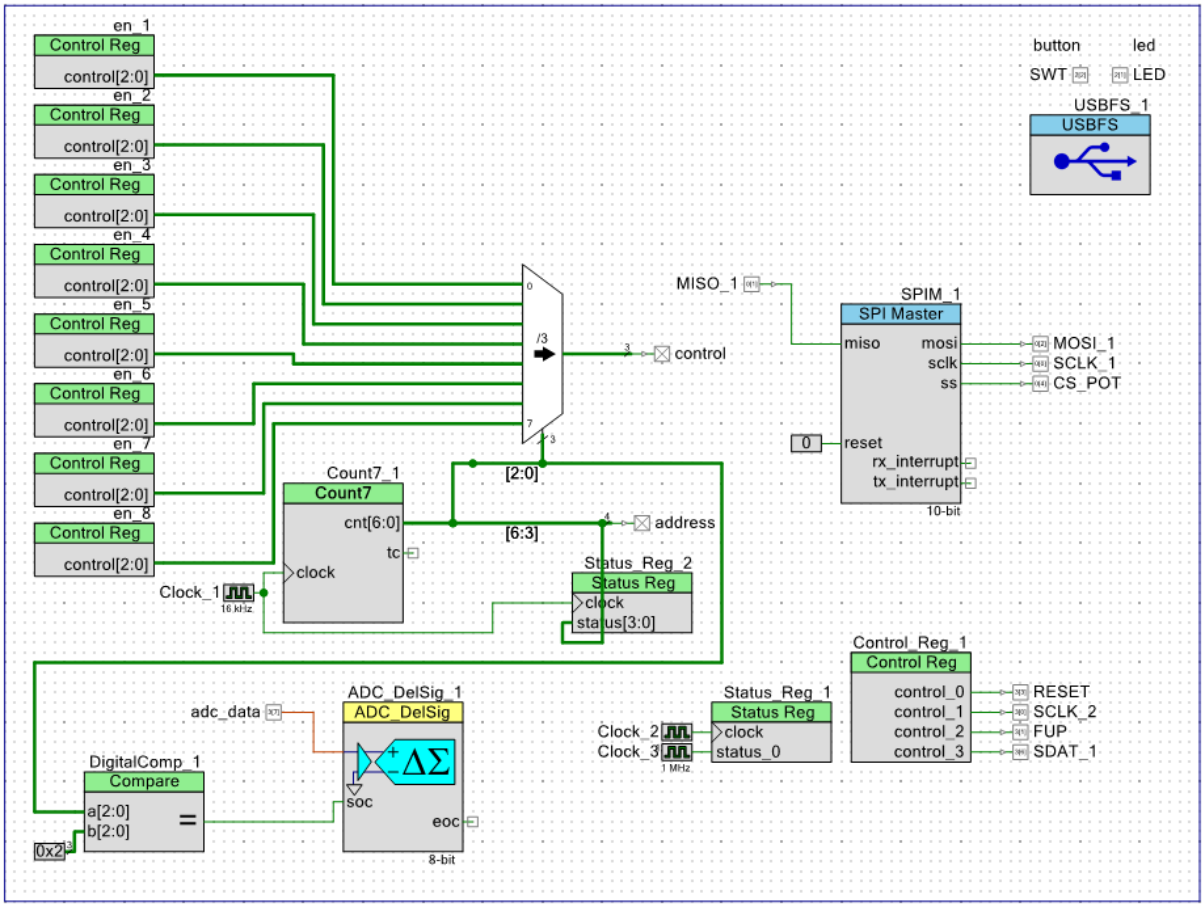


Figure 40 Microcontroller hardware elements.

Chapter 6.0 Computer Software

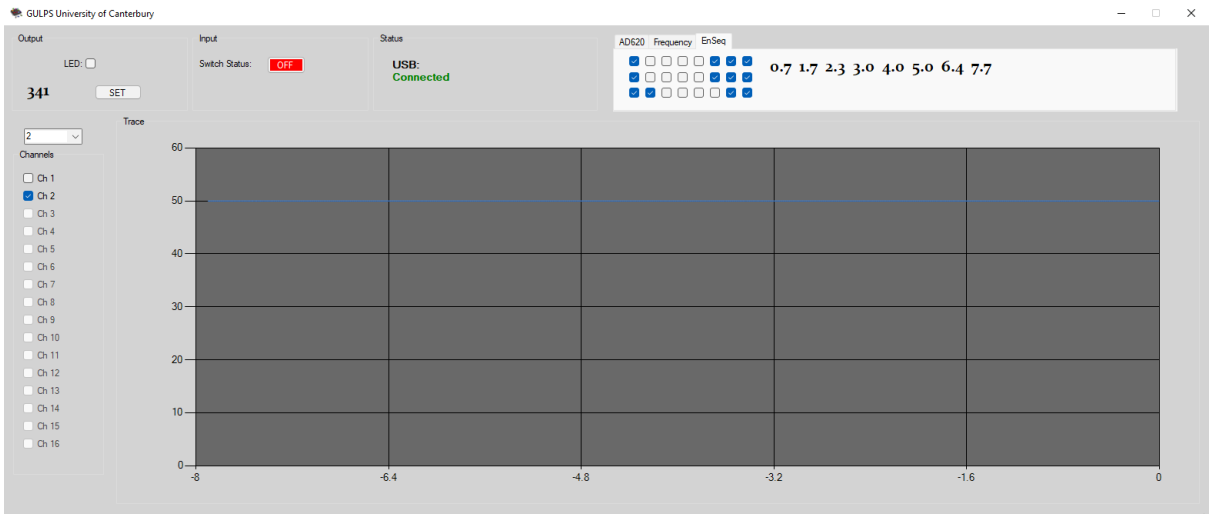


Figure 41 Screenshot of Visual Studio application.

The GULPs hardware was designed to be flexible so that parameters may be changed in real-time during testing on a human subject, in order to determine experimentally the most suitable configuration. Configurable parameters are injection frequency, sequencing of the multiplexers and gains for operational amplifiers.

The multiplexer sequencing depicted in Figure 41 is the configuration used when testing on saline solution. Note, enable pins on the multiplexers are active low. Injection frequency is displayed in kilohertz, where 341 kHz yielded the best results when testing with saline solution.

Additionally, the PC application displays a live trace from all channels, a tick-box allows the user to only display a trace from channels of interest. The value read by the analogue to digital converter is not filtered as aggressively as live analogue data available on the PCB, thus the traces displayed may contain more noise than its analogue counterpart.

The application was designed to also configure gain values for some of the instrumentational amplifiers, however the digital potentiometers selected for the hardware implementation were introducing noise and distortion. These were replaced by fixed value resistors or a potentiometer during testing.

The main function for the Visual Studio code is available in Appendix B.

Chapter 7.0 Operational Results

To verify circuit functionality and performance a variable resistive load circuit was designed and implemented. A relay powered by a signal generator decreases the effective load presented to the current source in a controlled manner. As channels are multiplexed, adjacent channels needed to be tested for cross-interference. The sequencing of the multiplexers was determined this way. When cross-talk could not be eliminated by configuring multiplexers then RC filters had their component values reduced to minimise time constants. A practical limit was reached with resistor values approaching that of analogue switches and capacitor values approaching that of parasitic capacitances inherent in the circuit. Sampling speed had to be reduced to eliminate cross-channel interference.

7.1 Simulated Resistive Load

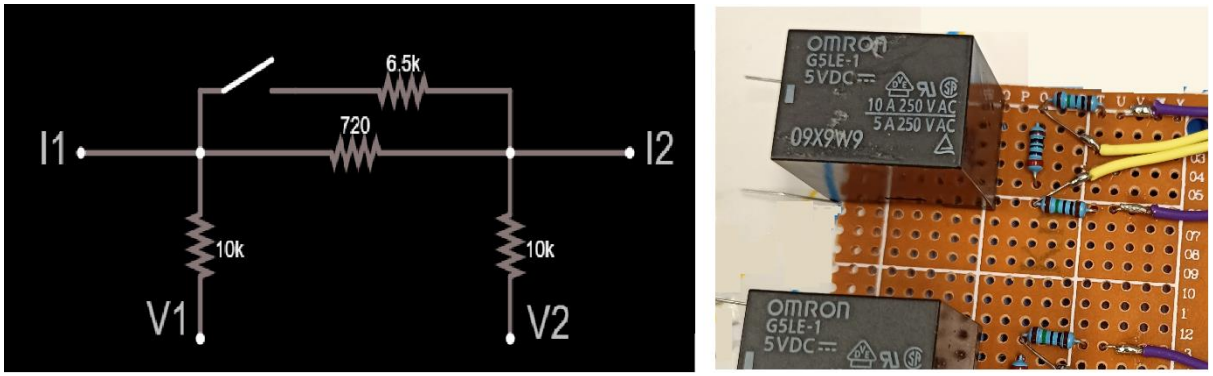


Figure 42 Test load circuit schematic and implementation.

A base load value of 720 ohms was selected as this was the base resistance measured in the author's neck. This value differs significantly to the 50 Ω test loads used by previous students. A relay switches in a 6.5 kΩ resistor to reduce the effective load resistance by 10%. A parallel arrangement was selected, as the relay on resistance approaches the value of resistance needed to achieve the same impedance change in a series configuration, figure 42.

The relay on and off time was adjusted and tested on one channel to verify that the hardware behaves as expected under specific conditions.

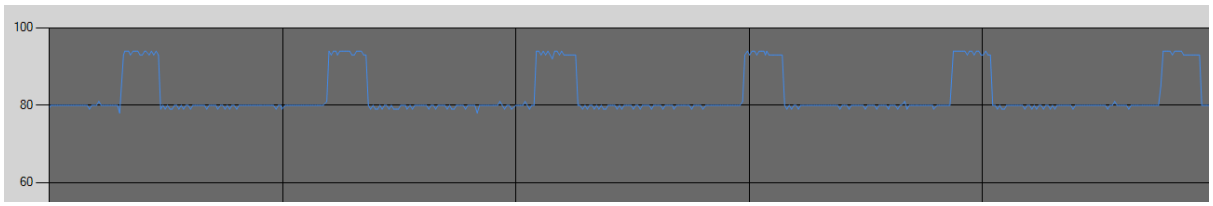


Figure 43 Test, load has 20% on time.

Figure 43 shows Test 1, where the output analogue signal is provided after demultiplexing and filtering where the low impedance was presented 20% of the time. The average value remains close to baseline, about 80 in raw analogue to digital readings, with the signal rising during low impedance periods (this signal characteristic was specifically implemented to emulate the manometry signal characteristics during a swallow where an increase in pressure causes an upward increase in amplitude).

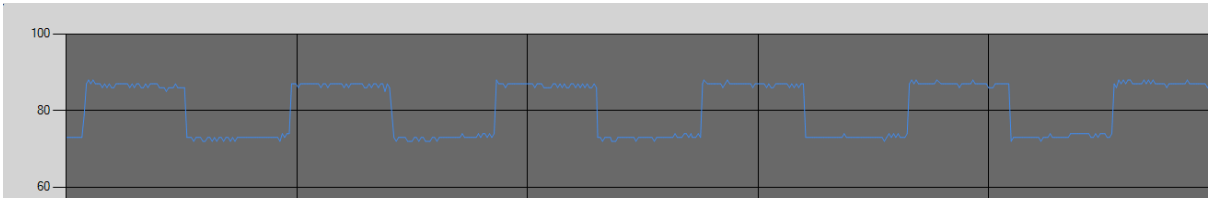


Figure 44 Test, load has 50% on time.

Test 2. Figure 44, low impedance for 50% of the time. The square wave settles halfway on 80.

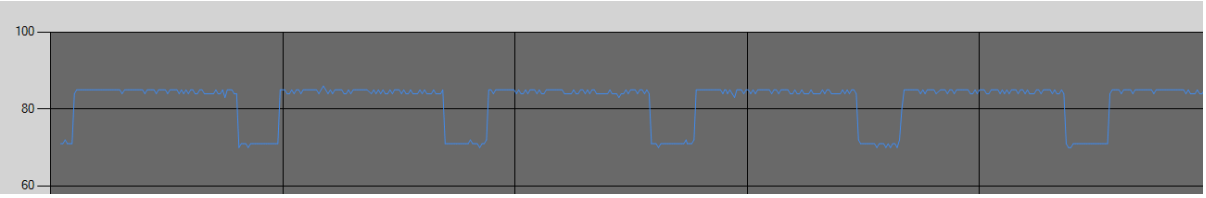


Figure 45 Test, load has 80% on time

Test 3. Figure 45, low impedance for 80% of the time. The baseline settles close to 80.

7.2 Crosstalk test

As only one channel is selected at any time, there are no other currents and voltages present to interfere with the currently selected channel, other than external sources of noise. Crosstalk however, does occur due to delays in the signal path. The effect is most noticeable on adjacent channels, with the first active channel affecting the next, and not vice versa. During testing, resistor and capacitor values had to be reduced from their calculated values. Particularly the third stage voltage storage capacitors on the envelope detector circuit, and the low pass filter on the active signal path of the difference amplifiers. When these values could be reduced no further it was decided to decrease the sampling rate to 750 Hz. Though this is below the intended sampling rate, in translates to each channel being sampled every 1.4 ms, which is not a noticeable delay and still useful for the intended application. The source of this unexpected delay remains to be identified.

Figure 46 shows a cross talk test between two adjacent channels, the yellow trace depicts the first channel, and the blue trace the next channel in the sampling sequence. The change in impedance was a 10% drop on a 720 Ω load resistor. The load was switched in manually thus the uneven duration of the pulses. No noticeable disturbance is detected on the second channel. Noise present in the signal was later attributed to the digital potentiometer used to set the gain of the final operational amplifier. The effect became more significant with higher gain values. Also noticeable in the image is the effect of the averaging filter. As the yellow trace remains in a lower impedance state (higher amplitude position), the channel average slowly begins to climb. When the load returns to its

usual state the live channel reading is now slightly below the channel average, causing the output signal to dip below the steady state. The signal can then be seen slowly settling back to mid-rail as the channel average returns to its normal value. The channel average time was increased as a result of this test, to minimise this effect.

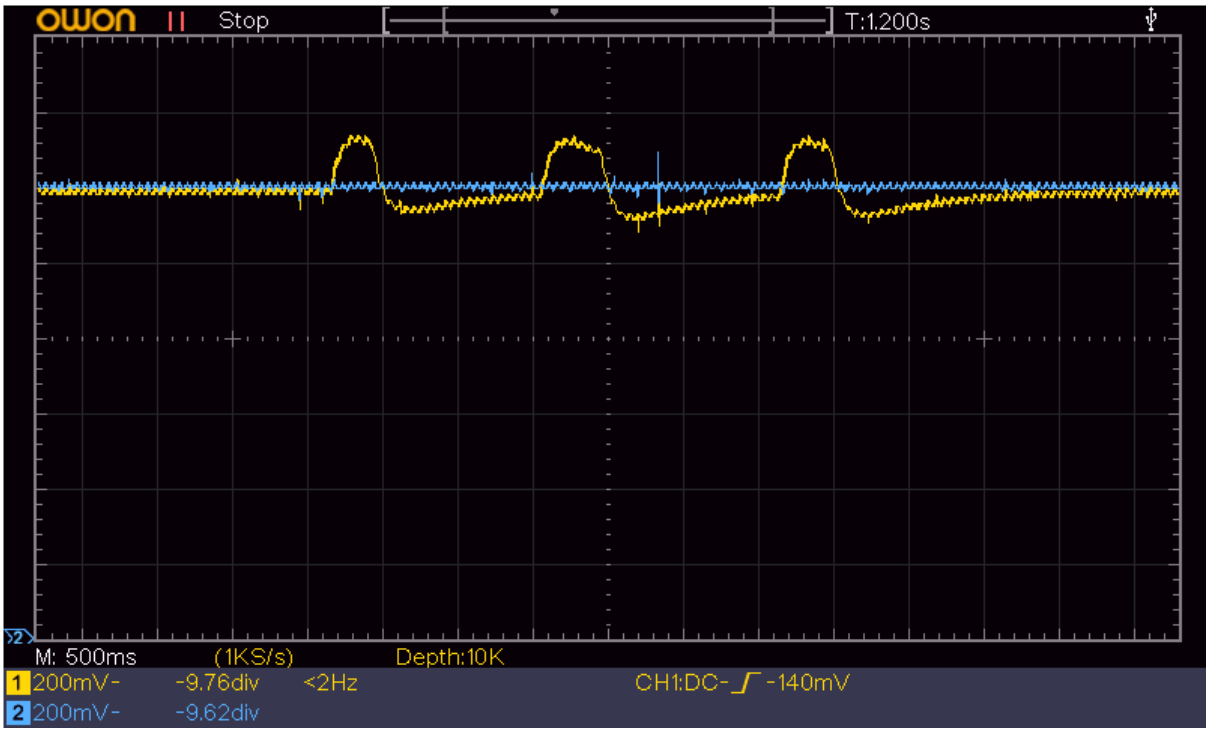


Figure 46 Crosstalk test. 200mV/div, 500 ms/div.

7.3 Saline solution

A vertical test chamber was constructed and filled with saline solution, metal pegs/electrodes along the height of the chamber serve as connection points for channel leads. Up to four channels where implemented this way as a substantial gap was left between channels to allow for the length of the test slug.

The chamber measures 20 cm in diameter. The test slug is made of brass and measures 2 cm in diameter and 4 cm in length, figure 47.

The chamber was filled with saline solution and the results of moving the test slug down and up the chamber length were recorded. The nominal impedance of the saline solution was calculated by measuring the voltage amplitude across the constant current source, and diluted with distilled water to be 720 Ω, a value selected by measuring the impedance of the author's own neck. The diameter of the teste slug was selected experimentally to give a 10% drop in impedance when placed in the middle of the chamber across a channel.

The colour traces in figure 49 each represent an individual channel. With yellow, purple, white and blue representing channels 1, 2, 3, and 4 respectively. The traces can each be seen registering a drop in impedance in sequence as the slug was inserted and then removed from the test chamber with no noticeable cross-talk on other channels.

The yellow trace, figure 49, is the bottom channel where the test slug did not fully clear the electrodes before being pulled up again. The drop in impedance caused by the brass slug can be seen as positive going pulses. All are of comparable amplitude and all baselines across all channels are equal as expected, as the system normalises impedances and only variations from the baseline are registered. The baseline figure of 100 differs slightly to the baseline of 80 during resistive load testing, this can be attributed to slight modifications to the multiplexer sequencing configuration via the Visual Studio application.

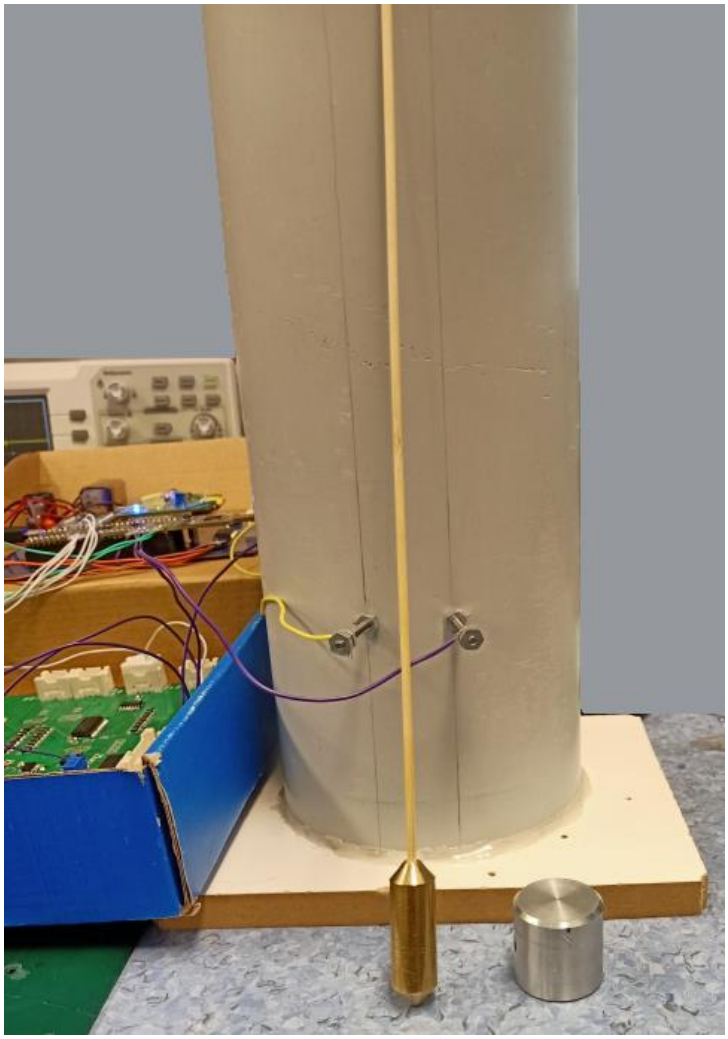


Figure 47 Test chamber and test slug.

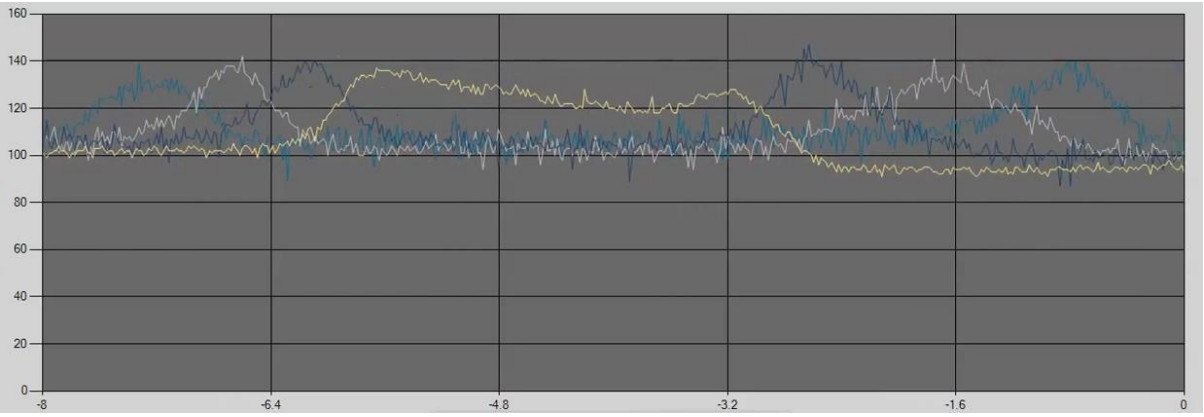


Figure 48 Four channel test results.

The system demonstrates sufficient sensitivity as it matches the 10% change sensitivity achieved in previous implementations. The number of channels was increased from 2 in previous implementations to a possible 16, with a sufficiently fast sampling rate.

Chapter 8.0 Conclusions and Future Work

Before testing on human subjects, the sensitivity of previous implementations of GULPS were tested with a 10% drop on a resistive load. Channel cross-talk and step response were also tested.

The new implementation of GULPS performed these tests suitably well as demonstrated in this thesis. Based on these results, the current implementation of GULPS is ready to move onto trials on human subjects. Additionally, the new system was tested with a saline solution column which electrically closely resembles conditions on a human test subject. The impedance value selected for the saline solution was determined experimentally from tests on the author's own neck.

The new system is capable of measuring 16 channels each with a 750 Hz sampling rate. This is an improvement on previous implementation's two channels.

Additionally, the new system utilises the same excitation frequency on all channels. As shown in Chapter 2, the impedance of biological tissue is a function of frequency. A multi-frequency approach would be measuring each channel at a different frequency, and thus the excitation signal would have different effects in each location.

The multiplexing approach to expanding the number of channels ensures all test regions are subjected to similar measurement conditions, if an ideal excitation frequency can be determined experimentally then all channels could be measured at the most effective frequency.

The new system can also be configured via an application, so that only utilized channels are multiplexed, this would increase the sampling rate achievable on the utilized channels.

Testing on human subjects, data-logging, investigating the source of unexpected delays in the signal path, and investigating the source of noise around the digital potentiometers remains for future work.

References

[1] S. K. Daniels and M.-L. Huckabee, “Dysphagia Following Stroke”. San Diego: Plural Publishing Inc, 2008.

[2] Kusuhara T., et al., "Impedance pharyngography to assess swallowing function”, Journal of International Medical Research, vol. 32, pp. 608-616, 2004.

[3] Lippett A., “Development of a Bio impedance-Based Swallowing Biofeedback Device with Smart Device Integration”, Masters in Engineering, Electrical and Electronic Engineering, University of Canterbury 2014.

[4] Jones B, editor. Normal and abnormal swallowing: imaging in diagnosis and therapy. 2nd ed. Springer-Verlag; New York: 2003.

[5] R. Pethig, "Dielectric Properties of Biological Materials: Biophysical and Medical Applications," Electrical Insulation, IEEE Transactions on, vol. EI-19, pp. 453-474, 1984.

[6] FYP Final Report, Rose Centre FYP Group E18, Medical Devices to Measure Swallowing Reflex.

[7] Low-Cost, High-Speed Differential Amplifier, Analog Devices

https://www.analog.com/media/en/technical-documentation/data-sheets/ad8132.pdf

[8] AD9850 CMOS, 125 MHz Complete DDS Synthesizer Data Sheet (REV. H) (analog.com)

[9] 74HC4067, “16-channel analogue multiplexer/demultiplexer”, Texas Instruments, 1998, datasheet.: www.ti.com/lit/ds/symlink/cd74hc4067.pdf

[10] Low-power Instrumentation Amplifier

https://www.analog.com/media/en/technical-documentation/data-sheets/ad8421.pdf

[11] High-Speed, Low-Power, 3V/5V, Rail-to-Rail, Single-Supply Comparators

https://datasheets.maximintegrated.com/en/ds/MAX941-MAX944.pdf

[12] Very High Speed, High Output Current, Voltage Feedback Amplifier

[13] TS5A3157 10-Ω SPDT Analog Switch, Texas Instruments

[14] Single-Supply, Rail-to-Rail Low Power FET-Input Op Amp

https://www.analog.com/media/en/technical-documentation/data-sheets/ad822.pdf

[15] Low-Cost Low Power Instrumentation Amplifier, Analog Devices

https://www.analog.com/media/en/technical-documentation/data-sheets/AD620.pdf

[16] Quad, 32-V, 1-MHz op amp, Texas Instruments

[17] PSoC Development Board CY8CKIT-059, Cypress Semiconductor

https://docs.rs-online.com/774b/0900766b8167f761.pdf

Appendix A

PSoC Code

```
/* ========================================
 *
 * Copyright University of Canterbury, 2019
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF University of Canterbury.
 * Author: Ryan McGurk
 *
 * ========================================
*/
#include "project.h"


// machine for setting frequency
uint8 freq0;
uint8 freq1;
uint8 freq2;
uint8 freq3;
uint8 freqB;
uint8 fmask;
uint8 fdata;

uint8 flags;
uint8 cnt1;
uint8 tx;

unsigned char buffer[64];
unsigned char outfer[64];
unsigned char data[64];

unsigned short raw_adc_burst[4];

uint8 len2;
uint8 fsm_freq;

uint16 new_gain_val;
uint8   fsm_gain;
uint8 delay_gain;

unsigned char x;

//unsigned char calvals[16];   // calibration values for channels

//
/* Defines for DMA_1 */
#define DMA_1_BYTES_PER_BURST 2
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_1_DST_BASE (CYDEV_SRAM_BASE)

/* Variable declarations for DMA_1 */
/* Move these variable declarations to the top of the function */
//uint8 DMA_1_Chan;
//uint8 DMA_1_TD[1];
```

```
uint16 sum_adc_raw;
uint8   destination_address;


//


//CY_ISR(my_isr) {
//    flag |= 0x01;
   // destination_address = Status_Reg_2_Read();
    //++destination_address;

    //sum_adc_raw = raw_adc_burst[0];
   // sum_adc_raw += raw_adc_burst[1];
    //sum_adc_raw += raw_adc_burst[2];
    //sum_adc_raw += raw_adc_burst[3];

   // sum_adc_raw >>= 1;

   // outfer[destination_address] = 123;//sum_adc_raw;

    // for debug ONLY
    // outfer[destination_address] = destination_address * 10;
    // if(destination_address > 5) Count7_1_WriteCounter(0);
    //

  //  isr_1_ClearPending();
//}
void set_frequency(void) {
    //
    //if(SPIM_2_GetTxBufferSize()) return;
    switch(fsm_freq) {
        case 0:
            // never get here
            break;
        case 1:
            tx = 01;
            cnt1 &= 0xf8;
            cnt1 |= 0x01;
            ++fsm_freq;
            break;
        case 2:
            cnt1 &= 0xf8;
            tx <<= 1;
            ++fsm_freq;
            if(tx & 0xf8) fsm_freq = 4;
            break;
        case 3:
            cnt1 &= 0xf8;
            cnt1 |= tx;
            fsm_freq = 2;
            break;
        case 4:
            // reset sequence has been sent
            freqB = 0;
            fdata = freq0;
            fmask = 0x01;
            fsm_freq = 6;
            break;
        case 5:
            cnt1 &= 0xfd;
```

```
            ++freqB;
            fdata = 0x00;
            if(freqB == 1) fdata = freq1;
            if(freqB == 2) fdata = freq2;
            if(freqB == 3) fdata = freq3;

            fmask = 0x01;

            ++fsm_freq;
            if(freqB > 4) fsm_freq = 8;
            break;
        case 6:
            cnt1 &= 0xf0;
            if(fdata & fmask) cnt1 |= 0x08;
            ++fsm_freq;
            break;
        case 7:
            cnt1 |= 0x02;   // clk
            fsm_freq = 6;
            fmask <<= 1;
            if(fmask == 0) fsm_freq = 5;
            break;
        case 8:
            cnt1 |= 0x04;
            ++fsm_freq;
            break;
        case 9:
            //end of transmission
            cnt1 &= 0xf0;
            fsm_freq = 0;
            flags &= 0x02;
            break;
        default:
            cnt1 = 0;
            fsm_freq = 0;
            flags &= 0x02;
            break;
    }

    Control_Reg_1_Write(cnt1);
}

void set_gain(void) {
    switch(fsm_gain) {
        case 0:
            // should never be here.
            break;
        case 1:
            SPIM_1_WriteTxData(new_gain_val);
            delay_gain = 0;
            fsm_gain = 2;
            break;
        case 2:
            if(SPIM_1_GetTxBufferSize() == 0) ++delay_gain;
            if(delay_gain > 30) fsm_gain = 3;
            break;
        case 3:
            new_gain_val |= 0x0100; // next pot address
            SPIM_1_WriteTxData(new_gain_val);
            fsm_gain = 4;
            break;
```

```
        case 4:
            if(SPIM_1_GetTxBufferSize() == 0) fsm_gain = 0;
            break;
        default:
            fsm_gain = 0;
            break;
    }
}

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    //isr_1_StartEx(my_isr);

    /* DMA Configuration for DMA_1 */
 //   DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
DMA_1_REQUEST_PER_BURST,
 //        HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
 //   DMA_1_TD[0] = CyDmaTdAllocate();
 //   CyDmaTdSetConfiguration(DMA_1_TD[0], 4, CY_DMA_DISABLE_TD,
DMA_1__TD_TERMOUT_EN  | CY_DMA_TD_INC_DST_ADR);
 //   CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)ADC_SAR_1_SAR_WRK0_PTR),
LO16((uint32)raw_adc_burst));
 //   CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
 //   CyDmaChEnable(DMA_1_Chan, 1);

    //
    //


    USBFS_1_Start(0u, USBFS_1_3V_OPERATION);

    // wait enumeration
    while(!USBFS_1_bGetConfiguration());

    //enable out
    USBFS_1_EnableOutEP(2);

    // DC offsets for each channel
    //calvals[0] = 0;     //
    //calvals[1] = 50;
    //calvals[2] = 100;
    //calvals[3] = 150;

    // data out
    outfer[0] = 0;      // switch status
    outfer[1] = 10;     // ch 1
    outfer[2] = 20;     // ch2
    outfer[3] = 30;   // .... etc
    outfer[4] = 40;
    outfer[5] = 50;
    outfer[6] = 60;
    outfer[7] = 70;
    outfer[8] = 80;
    outfer[9] = 90;
    outfer[10] = 100;
    outfer[11] = 110;
    outfer[12] = 120;
    outfer[13] = 130;
    outfer[14] = 140;
```

```
outfer[15] = 150;
outfer[16] = 160;

fsm_freq = 0;
fsm_gain = 0;

// enable SPI master
SPIM_1_Start();

//init_AD9850();

// channel circuitry
Count7_1_Start();

// adc
//ADC_SAR_1_Start();
ADC_DelSig_1_Start();
ADC_DelSig_1_StartConvert();

flags = 0;
cnt1 = 0;

//Clock_1_SetDividerValue(15);

for(;;)
{
    // ADC
    if(ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_RETURN_STATUS)) {
        outfer[1 + Status_Reg_2_Read()] = ADC_DelSig_1_Read8();
    }
    // get back LED status
    if(USBFS_1_bGetEPState(2) == USBFS_1_OUT_BUFFER_FULL) {
        len2 = USBFS_1_wGetEPCount(2);
        USBFS_1_ReadOutEP(2, &buffer[0], len2);

        if(buffer[0] != 0) {
            LED_Write(1);
        } else {
            LED_Write(0);
        }

        // get new pot values
        //calvals[0] = buffer[1];
        //calvals[1] = buffer[2];
        //calvals[2] = buffer[3];
        //calvals[3] = buffer[4];

        //freq0 = buffer[10];
        //freq1 = buffer[11];
        //freq2 = buffer[12];
        //freq3 = buffer[13];

        if(fsm_freq == 0) {
            fsm_freq = 0x01;
            freq0 = buffer[10];
            freq1 = buffer[11];
            freq2 = buffer[12];
            freq3 = buffer[13];
        }

        if(buffer[28]) {
```

```
                en_8_Write(buffer[19]);
                en_7_Write(buffer[20]);
                en_6_Write(buffer[21]);
                en_5_Write(buffer[22]);
                en_4_Write(buffer[23]);
                en_3_Write(buffer[24]);
                en_2_Write(buffer[25]);
                en_1_Write(buffer[26]);
            }

            //viewLive = buffer[14] + 1;  // what channel we want to view
live

            // get GAIN data - documentation
            //pots_data_array[3] = buffer[5];  // AD620 byte 1
            //pots_data_array[4] = buffer[6];
            //pots_data_array[5] = buffer[7];  // IA byte 1
            //pots_data_array[6] = buffer[8];
            //pots_data_array[7] = buffer[26];  // flags
            // if set, we can accept new data

            if(buffer[9]) {
                if(fsm_gain == 0) {
                    new_gain_val = buffer[7];
                    new_gain_val <<= 8;
                    new_gain_val |= buffer[8];

                    fsm_gain = 1;

                    //SPIM_1_WriteTxData(new_gain_val);
                }
            }
        }

        outfer[0] = 0xff;
        if(SWT_Read()) outfer[0] = 0x00;    // update switch status

        if(USBFS_1_bGetEPState(1) == USBFS_1_IN_BUFFER_EMPTY) {
            USBFS_1_LoadInEP(1, outfer, 32);
        }

        if(fsm_freq) flags |= 0x01;

        if(Status_Reg_1_Read()) {
            if(flags & 0x02) {
                flags &= 0xfd;
                if(flags & 0x01) set_frequency();
            }
        } else {
            if((flags & 0x02) == 0) if(fsm_gain) set_gain();
            flags |= 0x02;
        }
    }
}

/* [] END OF FILE */
```

Appendix B

Visual Studio Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using CyUSB;
using System.Windows.Forms.DataVisualization.Charting;

namespace WindowsFormsApplication1
{
    public partial class GenericHidForm : Form
    {

        // ********************** START OF PROGRAM MAIN
**************************** //

        //Constant variables for locations in Input and Output Data Arrays

        //Constants fro Input Buffer Array
        const uint Switch_Status_Position = 1;

        //Constants fro Output Buffer Array
        const uint LED_State_Position = 1;
        const uint Pot1 = 2;
        const uint Pot2 = 3;
        const uint Pot3 = 4;
        const uint Pot4 = 5;

        const uint PotAD620_H   = 6;
        const uint PotAD620_L   = 7;
        const uint PotIA_H      = 8;
        const uint PotIA_L      = 9;

        const uint Pot_flags    = 10;

        const uint Enable_Bits_Base = 20;   // +9 bytes

        // frequency
        const uint AD9850_Clock = 75000000;
        const uint MULT = 4294967295;
        const uint freq0 = 11;
        const uint freq1 = 12;
        const uint freq2 = 13;
        const uint freq3 = 14;

        // view live data
        const uint viewLive = 15;

        USBDeviceList usbDevices = null;        // Pointer to list of USB
devices
        CyHidDevice myHidDevice = null;         // Handle of USB device

        int Vendor_ID = 0x04B4;                       // Cypress Vendor ID
```

```
        int Product_ID = 0x0010;                        // Example Project
Product ID

        // flags
        int flags = 0;          // 0x01 - calibrate - autoset feature
        int delay_samples = 0;
        int cal_mchn = 0;
        uint tmp_cal;
        int tmp_pot;
        //int ghost = 0;
        int line_number;

        NumericUpDown DutyCycleUpDown;          // Handler for PWM updown
control

        List<Reading> readings = new List<Reading>();

        List<CheckBox> lcb;
        List<CheckBox> les;

        internal struct Reading
        {
            //internal uint value;
            internal DateTime time;
            internal uint ch1;
            internal uint ch2;
            internal uint ch3;
            internal uint ch4;
            internal uint ch5;
            internal uint ch6;
            internal uint ch7;
            internal uint ch8;
            internal uint ch9;
            internal uint chA;
            internal uint chB;
            internal uint chC;
            internal uint chD;
            internal uint chE;
            internal uint chF;
            internal uint chG;
        }


        //uint[,] channel_data = new uint[16, 512];


/*********************************************************************
        * NAME: GenericHIDForm
        *
        * DESCRIPTION: Main function called initalled upon the starting of
the
        * application. Used to unitialize variables, the GUI application,
register
        * the event handlers, and check for a connected device.
        *

*********************************************************************/

        public GenericHidForm()
        {
            //Initialize the main GUI
```

```
            InitializeComponent();

            /*
             */
            lcb = groupBox5.Controls.OfType<CheckBox>().ToList();


            for (int i = 0; i < lcb.Count; ++i)
            {
                lcb[i].Enabled = false;
            }


            les = tabPage1.Controls.OfType<CheckBox>().ToList();
            for (int i = 0; i < les.Count; ++i)
            {
                les[i].Checked = true;
                les[i].Click += new EventHandler(enable_sequence);
            }
            /*
             */

            //Set initial values
            DutyCycleUpDown = new NumericUpDown();
            DutyCycleUpDown.Value = 10;
            DutyCycleUpDown.Minimum = 0;
            DutyCycleUpDown.Maximum = 100;

            //Callback to set numeric updown box value
            DutyCycleUpDown.ValueChanged += new
EventHandler(DutyCycleUpDown_OnValueChanged);

            // Create a list of CYUSB devices for this application
            usbDevices = new USBDeviceList(CyConst.DEVICES_HID);

            //Add event handlers for device attachment and device removal
            usbDevices.DeviceAttached += new
EventHandler(usbDevices_DeviceAttached);
            usbDevices.DeviceRemoved += new
EventHandler(usbDevices_DeviceRemoved);

            //Connect to the USB device
            GetDevice();

        }

        private void enable_sequence(object sender, EventArgs e)
        {
            String str = "";
            byte tmp_sum = 0;
            int ch_cnt = 0;
            line_number = 0;

            if (myHidDevice != null)
myHidDevice.Outputs.DataBuf[Enable_Bits_Base + 9] = 0x01;

            for (int i = 0; i < les.Count; ++i)
            {
                tmp_sum <<= 1;
                if (les[i].Checked == true) tmp_sum += 1;
                ++ch_cnt;
```

```
                if((ch_cnt % 3) == 0)
                {
                    str += line_number + ".";
                    str += tmp_sum.ToString() + "  ";

                    // add to out buffer
                    if (myHidDevice != null)
myHidDevice.Outputs.DataBuf[Enable_Bits_Base + (7 - line_number)] =
tmp_sum;

                    ++line_number;
                    tmp_sum = 0;
                }
            }
            label2.Text = str;

            if (myHidDevice != null) myHidDevice.WriteOutput();
        }



/**********************************************************************
        * NAME: DutyCycleUpDown_OnValueChanged
        *
        * DESCRIPTION: Event Handler for the numeric up/down text field.
This
        * function is called when the value in the next field is updated
with
        * the arrows so that the value is the text field can be properlly
updated
        * to reflect the change.
        *

**********************************************************************/
        private void DutyCycleUpDown_OnValueChanged(object sender,
EventArgs e)
        {
            Console.WriteLine(DutyCycleUpDown.Value);
        }


/**********************************************************************
        * NAME: usbDevices_DeviceRemoved
        *
        * DESCRIPTION: Event handler for the removal of a USB device. When
the removal
        * of a USB device is detected, this function will be called which
will check to
        * see if the device removed was the device we were using. If so,
then reset
        * device handler (myHidDevice), disable the timer, and update the
GUI.
        *

**********************************************************************/

        public void usbDevices_DeviceRemoved(object sender, EventArgs e)
        {

            USBEventArgs usbEvent = e as USBEventArgs;
```

```
            if ((usbEvent.ProductID == Product_ID) && (usbEvent.VendorID ==
Vendor_ID))
            {
                InputTimer.Enabled = false;         // Disable interrupts
for polling HID device
                myHidDevice = null;
                GetDevice();                        // Process device
status
            }
        }


/*****************************************************************
        * NAME: usbDevices_DeviceAttached
        *
        * DESCRIPTION: Event handler for the attachment of a USB device.
The function
        * first checks to see if a matching device is already connected by
seeing
        * if the handler (myHidDevice) is null. If no device is previously
attached,
        * the function will call GetDevice to check and see if a matching
device was
        * attached.
        *

*****************************************************************/

        public void usbDevices_DeviceAttached(object sender, EventArgs e)
        {
            if (myHidDevice == null)
            {
                GetDevice();                        // Process device
status
            }
        }



/*****************************************************************
        * NAME: GetDevice
        *
        * DESCRIPTION: Function checks to see if a matching USB device is
attached
        * based on the VID and PID provided in the application. When a
device is
        * found, it is assigned a handler (myHidDevice) and the GUI is
updated to
        * reflect the connection. Additionally, if the device is not
connected,
        * the function will update the GUI to reflect the disconnection.
        *

*****************************************************************/

        public void GetDevice()
        {
            //Look for device matching VID/PID
            myHidDevice = usbDevices[Vendor_ID, Product_ID] as CyHidDevice;
```

```
            if (myHidDevice != null)                //Check to see if
device is already connected
            {

                Status.Text = "Connected";
                Status.ForeColor = Color.Green;
                SwStatus.Enabled = true;
                InputTimer.Enabled = true;          //Enable background
timer

                Update_LED();                       //Initialize the LED
based on current GUI configuration
                //Update_PWMDutyCycle();              //Initialize the PWM
based on current GUI configuration

            }

            else
            {
                Status.Text = "Disconnected";
                Status.ForeColor = Color.Red;
            }
        }



/***********************************************************************
         * NAME: Update_PWMDutyCycle
         *
         * DESCRIPTION: Function used to update the state of the PWM Duty
Cycle
         * on the PSoC device end by reading the value on the PWM Duty
Cycle text field
         * (DutyTextBox), applying the change to the Output Data Buffer,
and
         * writing the OUT report.
         *

***********************************************************************/

        //public void Update_PWMDutyCycle()
       // {
            //Update the Output Buffer for PWM based on selected checkbox
setting
        //    DutyCycle = Convert.ToUInt32(DutyTextBox.Text);
        //    myHidDevice.Outputs.DataBuf[PWM_DutyCycle_Position] =
(byte)DutyCycle;
        //    myHidDevice.WriteOutput();
        //}


/***********************************************************************
         * NAME: Update_LED
         *
         * DESCRIPTION: Function used to update the state of the LED on the
PSoC
         * device end by checking the current status of the LED check box,
         * applying the change to the Output Data Buffer, and writing the
OUT report.
         *
```

```
*********************************************************************/

        public void Update_LED()
        {
            //Update the Output Buffer for LED based on selected checkbox
setting
            if (LED_State_CheckBox.Checked)
            {
                myHidDevice.Outputs.DataBuf[LED_State_Position] = 0xFF;
            }

            else
            {
                myHidDevice.Outputs.DataBuf[LED_State_Position] = 0x00;
            }

            myHidDevice.WriteOutput();
        }


/*********************************************************************
         * NAME: InputTimer_Tick
         *
         * DESCRIPTION: Function called by Timer (InputTimer) which is used
to poll
         * for input data every 10ms. Function will check contents of Input
Data
         * and change display of switch status and update the ADC Value
field.
         *

*********************************************************************/

        private void InputTimer_Tick(object sender, EventArgs e)
        {
            if (myHidDevice != null)
            {
                //InputTimer.Enabled = false;
// Disable timer so we don't get another interrupt until we service this
interrupt
                myHidDevice.ReadInput();
// This CyUSB.DLL method uses the Win32 ReadFile() function to read IN data
transferred to our application from the device

                if (myHidDevice.Inputs.DataBuf[Switch_Status_Position] !=
0x00)                 // Check to see if Push Button is pressed. Update GUI
based on results
                {
                    SwStatus.BackColor = Color.Lime;
                    SwStatus.Text = "ON";
                }
                else
                {
                    SwStatus.BackColor = Color.Red;
                    SwStatus.Text = "OFF";
                }



                // read all channel data
```

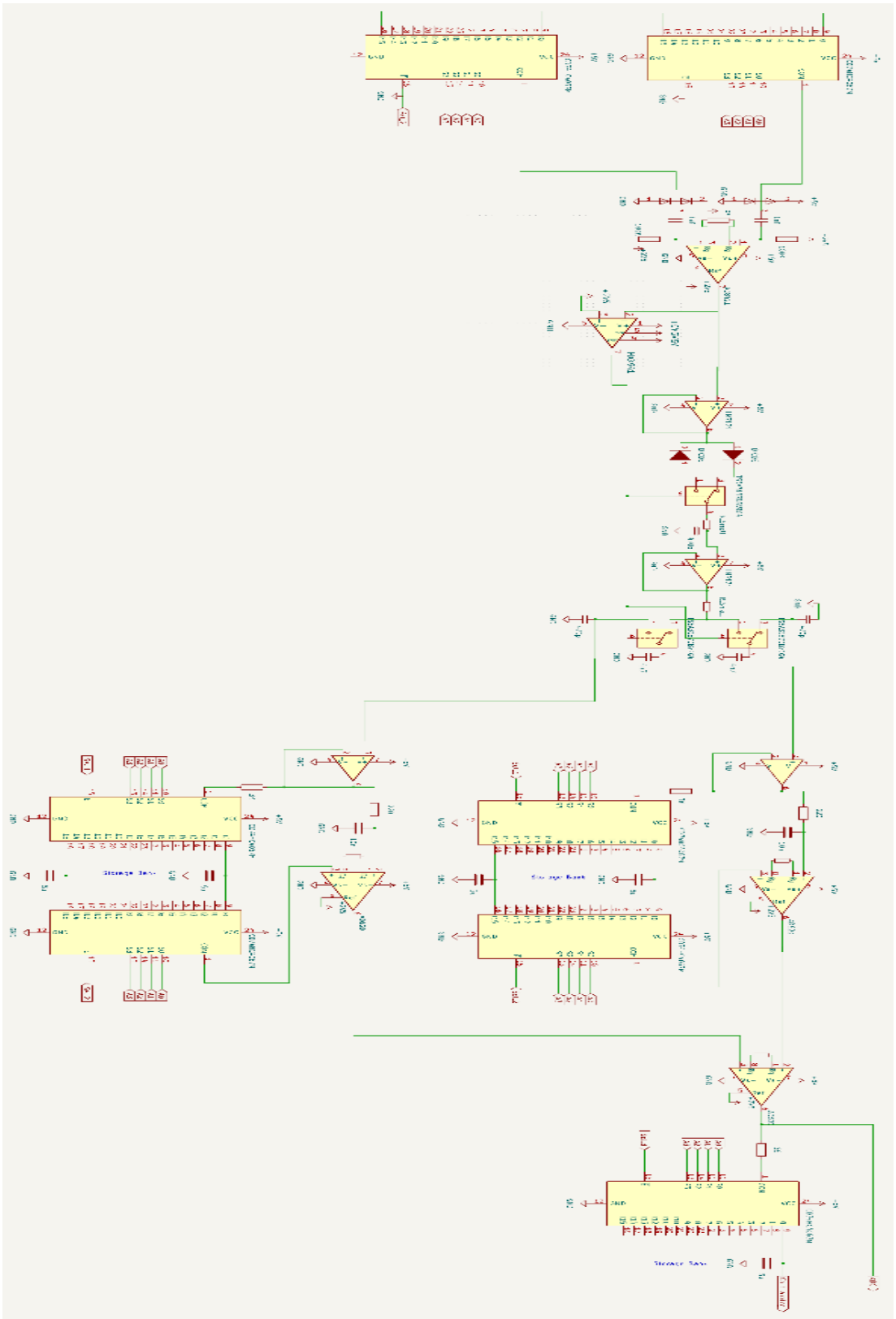
```
                readings.Add(new Reading()
                {
                    //value = 0,
                    time = DateTime.Now,
                    ch1 = myHidDevice.Inputs.DataBuf[2],
                    ch2 = myHidDevice.Inputs.DataBuf[3],
                    ch3 = myHidDevice.Inputs.DataBuf[4],
                    ch4 = myHidDevice.Inputs.DataBuf[5],
                    ch5 = myHidDevice.Inputs.DataBuf[6],
                    ch6 = myHidDevice.Inputs.DataBuf[7],
                    ch7 = myHidDevice.Inputs.DataBuf[8],
                    ch8 = myHidDevice.Inputs.DataBuf[9],
                    ch9 = myHidDevice.Inputs.DataBuf[10],
                    chA = myHidDevice.Inputs.DataBuf[11],
                    chB = myHidDevice.Inputs.DataBuf[12],
                    chC = myHidDevice.Inputs.DataBuf[13],
                    chD = myHidDevice.Inputs.DataBuf[14],
                    chE = myHidDevice.Inputs.DataBuf[15],
                    chF = myHidDevice.Inputs.DataBuf[16],
                    chG = myHidDevice.Inputs.DataBuf[17],
                });

                while (readings.Count > 500) readings.RemoveAt(0);


                //Re-enable the timer
                //InputTimer.Enabled = true;
            }
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            DateTime now = DateTime.Now;

            int x, y;
            DateTime t;

            int ci;
            //int lmt;


            //lmt = comboBox1.SelectedIndex;

            for (x = 0; x < 16; ++x)
            {
                chart1.Series[x].Points.Clear();
            }


            //for (y = 0; y < 16; ++y)
            //{
                for (x = 0; x < readings.Count; ++x)
                {

                    t = readings[x].time;
                    ci = 0;
                    if (lcb[ci++].Checked) chart1.Series[0].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch1));
                    if (lcb[ci++].Checked) chart1.Series[1].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch2));
```

```
                    if (lcb[ci++].Checked) chart1.Series[2].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch3));
                    if (lcb[ci++].Checked) chart1.Series[3].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch4));
                    if (lcb[ci++].Checked) chart1.Series[4].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch5));
                    if (lcb[ci++].Checked) chart1.Series[5].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch6));
                    if (lcb[ci++].Checked) chart1.Series[6].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch7));
                    if (lcb[ci++].Checked) chart1.Series[7].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch8));
                    if (lcb[ci++].Checked) chart1.Series[8].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].ch9));
                    if (lcb[ci++].Checked) chart1.Series[9].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].chA));
                    if (lcb[ci++].Checked) chart1.Series[10].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].chB));
                    if (lcb[ci++].Checked) chart1.Series[11].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].chC));
                    if (lcb[ci++].Checked) chart1.Series[12].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].chD));
                    if (lcb[ci++].Checked) chart1.Series[13].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].chE));
                    if (lcb[ci++].Checked) chart1.Series[14].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].chF));
                    if (lcb[ci++].Checked) chart1.Series[15].Points.Add(new
DataPoint((t - now).TotalSeconds, readings[x].chG));


                }

            //}

        }


/*********************************************************************
         * NAME: Update_PWM_Click
         *
         * DESCRIPTION: When "Update" button is pressed to update PWM
value, function
         * checks to see if a matching USB device is currently connected.
If so, function
         * calls another function to update the PWM Duty Cycle on the
device.
         *

*********************************************************************/

        private void Update_PWM_Click(object sender, EventArgs e)
        {
            //Respond to user pressing "Update" button. Make sure device is
connected before hand.
            if (myHidDevice != null)
            {
                //Update_PWMDutyCycle();
            }
        }
```

```
/*********************************************************************
         * NAME: Set_VidPid_Click
         *
         * DESCRIPTION: Updates the applications Vendor ID and Product ID
based on
         * user input when the "Set" button is clicked. This will cause the
default VID
         * and PID of 0x04B4 and 0xE177 to be overwritten. The function
will then
         * call GetDevice() to check for matching USB device.
         *
*********************************************************************/

        //private void Set_VidPid_Click(object sender, EventArgs e)
        //{
            //Respond to update of VID and PID value by pressing the "Set"
button
           // Vendor_ID = Convert.ToInt32(VidTextBox.Text, 16);
           // Product_ID = Convert.ToInt32(PidTextBox.Text, 16);
            //GetDevice();
        //}



/*********************************************************************
         * NAME: LED_State_CheckBox_CheckedChanged
         *
         * DESCRIPTION: When state of the LED State check box is changed,
function
         * checks to see if a matching USB device is currently connected.
If so, function
         * calls another function to update the LED state on the device.
         *
*********************************************************************/

        private void LED_State_CheckBox_CheckedChanged(object sender,
EventArgs e)
        {
            //Respond to change in LED checkbox state. Make sure device is
connected before hand.
            if (myHidDevice != null)
            {
                myHidDevice.Outputs.DataBuf[Pot_flags] = 0;
                Update_LED();
            }
        }

        private void VidTextBox_MaskInputRejected(object sender,
MaskInputRejectedEventArgs e)
        {

        }

        private void AdcValueBox_TextChanged(object sender, EventArgs e)
        {

        }
```

```
        private void TrackBar6_Scroll(object sender, EventArgs e)
        {
            if (myHidDevice != null)
            {
                int val_t = trackBar6.Value;
                myHidDevice.Outputs.DataBuf[PotIA_L] = (byte)(val_t &
0xff);
                val_t >>= 8;
                myHidDevice.Outputs.DataBuf[PotIA_H] = (byte)(val_t &
0x03);
                myHidDevice.Outputs.DataBuf[Pot_flags] = 0x08;
                myHidDevice.WriteOutput();
            }
        }

        private void TrackBar7_Scroll(object sender, EventArgs e)
        {
            label1.Text = (trackBar7.Value.ToString());
        }

        private void Button1_Click(object sender, EventArgs e)
        {
            if (myHidDevice != null)
            {
                uint val_t = (uint)trackBar7.Value;
                uint tmp = MULT / AD9850_Clock;
                val_t *= 1000;  // frequency in kHz
                //val_t *= MULT;
                //val_t /= AD9850_Clock;
                val_t *= tmp;

                myHidDevice.Outputs.DataBuf[freq0] = (byte)(val_t & 0xff);
                val_t >>= 8;
                myHidDevice.Outputs.DataBuf[freq1] = (byte)(val_t & 0xff);
                val_t >>= 8;
                myHidDevice.Outputs.DataBuf[freq2] = (byte)(val_t & 0xff);
                val_t >>= 8;
                myHidDevice.Outputs.DataBuf[freq3] = (byte)(val_t & 0xff);
                myHidDevice.Outputs.DataBuf[Pot_flags] = 0x02;
                myHidDevice.WriteOutput();
            }
        }
        private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
        {
            List<CheckBox> lcb =
groupBox5.Controls.OfType<CheckBox>().ToList();

            for(int i = 0; i < lcb.Count; ++i)
            {
                lcb[i].Enabled = false; // assume all checkboxes will be
dissabled
                lcb[i].Checked = false; // all check boxes default to
unselected
                if (i <= comboBox1.SelectedIndex) lcb[i].Enabled = true;
// enable of within channel count range
            }
        }

    }
}
```

Appendix C



PCB Artwork

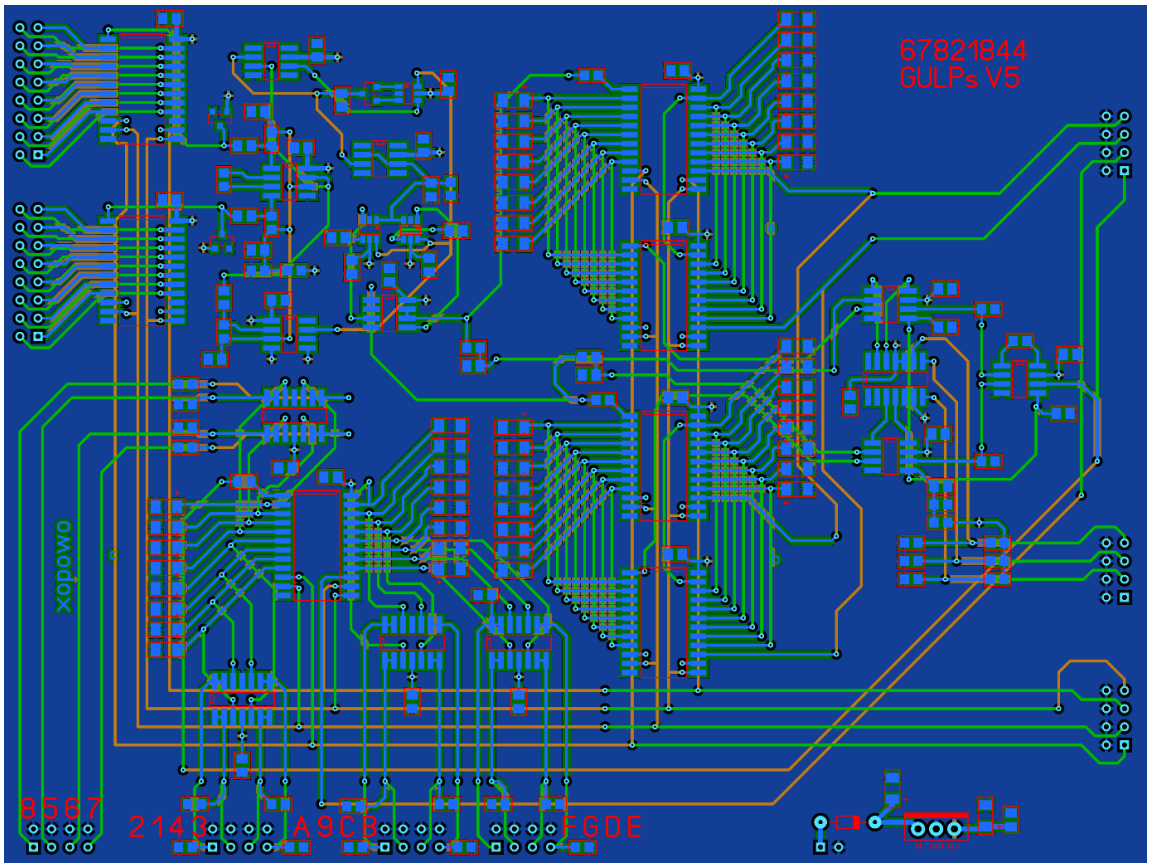


Figure 49 Detector PCB.

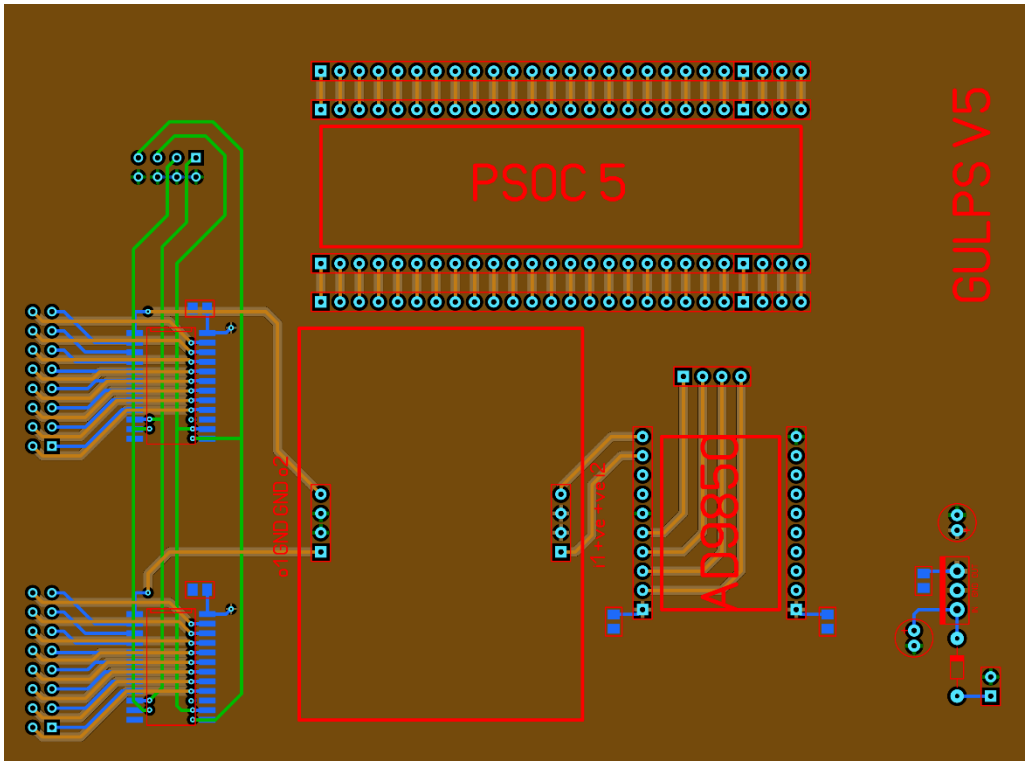


Figure 50 Current source and controller PCB