University of Southern Queensland

Faculty of Health, Engineering & Sciences

DSP Based Lock-in Amplifier

A dissertation submitted by

Robert George Skillington

in fulfilment of the requirements of

Courses ENG4111 and ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Electrical and Electronic)

Submitted: October, 2013

Abstract

This project aims to test the feasibility of, and identify key specifications for a small stand-alone lock-in amplifier using an embedded device. This may be used in conjunction with sensors generating a small signal and used in high noise environments. The intention is that this form of implementation will drastically reduce the cost and size of a lock-in amplifier while maintaining sufficient accuracy and noise immunity.

The specification for this project on "DSP-Based Lock-in Amplifier" was developed and agreed on with the Faculty of Health, Engineering & Sciences.

For speed, cost and other considerations, computer simulations of microprocessor hardware components were used in this project.

A Dual-phase lock-in amplifier was specifically chosen and examined for this project so as to minimise the component count of future hardware implementations.

Specifications identified as key parameters were ADC resolution, ADC sample frequency and integration time for hardware implementation. The specifications determined by this project are within the specifications of current microprocessors and DSP's.

Recommendations were made for possible future study

University of Southern Queensland

Faculty of Health, Engineering & Sciences

ENG4111/2 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled ``Research Project'' is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Robert George Skillington

0050013265

_____

Signature

_____

Date

Acknowledgements

I would like to take this opportunity to thank all the people who helped and supported me during the development of this project.

Firstly, I would like to thank my supervisor Dr John Leis for his support and his advice during this project.

I would like to thank my university colleagues for their support and providing a positive working atmosphere.

On a personal note, I would like to thank my friends and family for their prayers and support during the time of this project and degree.

A special thanks to Ray Hawkins for his advice and help proofreading this dissertation.

Table of Contents

List of Figures

List of Tables

Nomenclature and Acronyms

LIA	Lock-in amplifier
SNR	Signal to Noise Ratio
DSP	Digital Signal Processor
AC	Alternating Current
DC	Direct Current
LPF	Low Pass Filter
BPF	Band Pass Filter
PLL	Phase Locked Loop
dB	Decibel

Chapter 1 Introduction

1.1 Project Aim

This project aims to test the feasibility of, and identify key specifications for a small stand-alone lock-in amplifier using an embedded device. This may be used in conjunction with sensors generating a small signal and used in high noise environments. The intention is that this form of implementation will drastically reduce the cost and size of a lock-in amplifier while maintaining sufficient accuracy and noise immunity.

A real world application for such a device might be a gas detector. The current size of a laboratory lock-in amplifier (rack size) and cost greater than $4000 prohibits the use of these devices in small low cost applications.

The final implementation (not part of this project) will be carried out using a microprocessor that has a suitable architecture to implement a digital signal processing (DSP) based lock-in amplifier. Thus, this project will identify the general specifications of the hardware and software to be implemented in a detection systems.

For speed, cost and other consideration computer simulations of the hardware components will be used in this project.

1.2 Research Objectives

The specification for this project on “DSP-Based Lock-in Amplifier” was developed and agreed on with the Faculty of Health, Engineering & Sciences. The objectives identified were:

1. Research the design and use of the Lock-in Amplifier, both analogue and digital.

2. Determine the set of algorithms to be implemented for the lock-in, including the reference signal multiplication, low pass filtering, and the separate phase-locked loop.

3. Evaluate the performance of the algorithms as implemented, and show the signal recovery performance for various parameter settings at different SNR levels.

4. Evaluate the performance in MATLAB using sampled real-world signals.

5. Investigate suitable processor architectures and development systems for an embedded lock-in amplifier.

Chapter 2 Understanding the Environment for Lock-in Amplifiers

2.1 Chapter Overview

This chapter examines the different type of lock-in amplifiers and their function. The digital lock-in amplifier and existing literature on microprocessor based lock-in amplifiers.

2.2 Environment and Issues

The signal is hidden in the noise. The target environment is one in which the desired signal is much smaller than the noise in which it is embedded. To analysis the signal we need it amplified.

The following examples will demonstrate methods of retrieving the signal of interest adapted from Wenn (2007). A general amplifier will amplify the signal and the noise within the amplifiers band width and as well, injects additional noise.

Consider a clean sine wave with an amplitude of 100 nV at a frequency of 50 kHz. After the signal is amplified through an amplifier with an input noise of 5 $nV/\sqrt{Hz}$ and a gain of 1000. If the amplifier has a bandwidth of 100 kHz it will add bandwidth noise equal to that in equation 2.2.1.

$$5\ nV\ \times 1000 \times \sqrt{100\ kHz} = 1.6\ mV \qquad (2.2.1)$$

After the signal is amplified it becomes 100 μV ($1000 \times 100\ nV$) which is 16 times smaller than the surrounding noise of 1.6 mV, this is demonstrated by the area under the amplifier bandwidth in Figure 2-1.

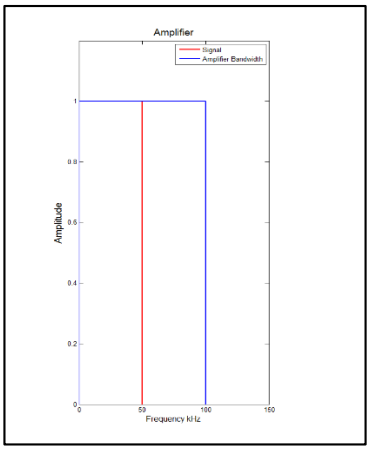

Figure 2-1: Weak signal amplified ready for measurement

When we examine these equations we see that it is the bandwidth of the noise that has caused the high output noise levels. The solution is to reduce the bandwidth.

One method is to use a band pass filter. If the signal is first passed through a high quality band pass filter with a Q factor of (say) 100 which has a bandwidth of 500 Hz (50 kHz/Q) at a centre frequency of 50 kHz. The noise level present after filtering and amplification is given by equation 2.2.2.

$$5\ nV\ \times 1000 \times \sqrt{500\ Hz} = 112\ \mu V \qquad (2.2.2)$$

After the filtered signal has been amplified the signal is still 1.1 times smaller than the surrounding noise of 112 μV, this is demonstrated by the area under the band pass filter bandwidth in Figure 2-2 the diagram demonstrates the difference of filtering and amplification.

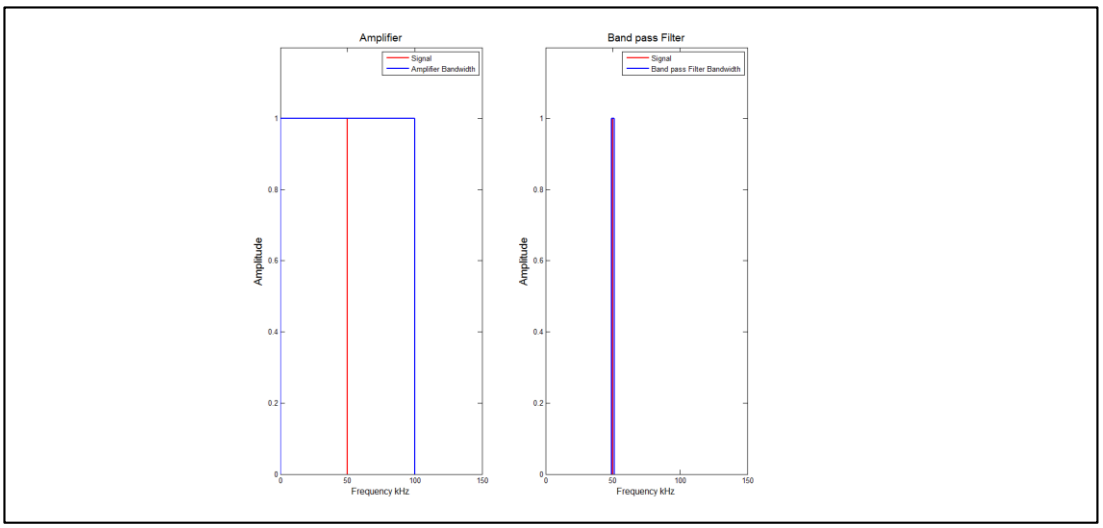

Figure 2-2: Weak signal passed through a band pass filter then amplified

Even though the filtered signal is approximately equal to the noise it is still difficult to measure and analysis.

An even tighter bandwidth is required. Lock-in amplifiers are one way to achieve this.

The Q factor of a lock-in amplifier can be as high as 10,000 this Q factor gives a bandwidth of 5 Hz (50 kHz/Q). The noise level present after the lock-in is set to the signal frequency, is given by equation 2.2.3.

$$5\ nV\ \times 1000 \times \sqrt{5\ Hz} = 11.2\ \mu V \qquad (2.2.3)$$

The noise present after using the lock-in amplifier is approximately 9 times smaller than the signal, an accurate measurement can then be taken of signal. Figure 2-3 shows the progression of the noise levels produced by amplification, filtering and the use of a lock-in amplifier.

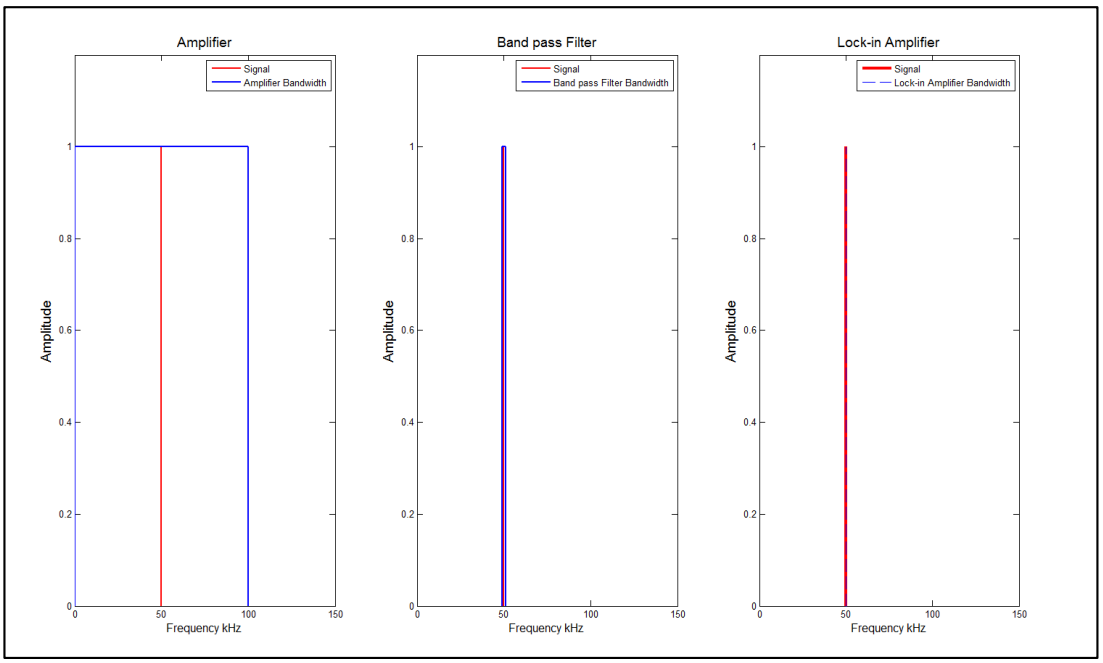

Figure 2-3: Weak signal passed through a band pass filter then amplified, then run through a LIA

As can be seen from the example above, using a lock-amplifier will reduce the bandwidth of the noise to the extent that the signal is sufficiently larger than the noise and can be identified and analysed. This is the rationale for the use of lock-in amplifies with remote sensors that produce weak signals.

2.3 Lock-in amplifier Technologies

A basic lock-in amplifier system is comprised of a wave generator, 'experiment' and the lock-in amplifier as shown in Figure 2-4. The wave generator produces a reference sine wave which is the input into the experiment. As the wave passes through the experiment it is phase shifted and noise is induced on the signal. An example of this is when a light source is pulsed at the reference frequency and is then received by a photo detector. The output of the experiment and the reference sine wave are then multiplied together in the lock-in amplifier (Vogelgesang, 2004). This multiplication results in the cancelation of the frequencies other than the reference signal frequency (Aguirre et al. 2011).

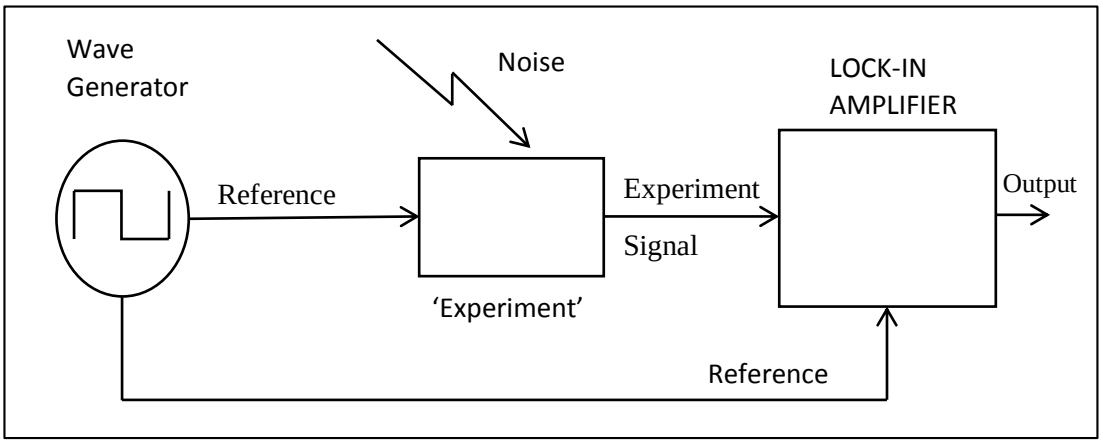

Figure 2-4: A basic lock-in amplifier system adapted from Vogelgesang, 2004.

For a lock-in amplifier to operate effectively it must be set to detect the signal of interest. This is done by supplying the lock-in with a fixed reference voltage of the same frequency and phase as the experimental signal. This reference signal is used to lock onto the same frequency in the experimental signal to retrieve the experiment result. This locking ensures that any reference signal changes are taken into account (Kim et al. 2009); this process is how the instrument gets its name.

There are a number of ways of achieving this 'locking'. The following section looks at the different ways of achieving this 'locking' function by examining the different types and sub-types of lock-in amplifiers.

2.4 Types of Lock-in amplifiers

There are two main types of lock-in amplifiers. These can be separated analogue and digital

- **Analogue**
  Analogue lock-in amplifiers use analogue multipliers which are expensive, complex and can be nonlinear (Bengtsson, 2012). The analogue lock-in amplifiers also require a phase shifter to align the reference signal and the experimented signal to compensate for line length or experimental delay (Davies & Meuli, 2010).
- **Digital**
  There two types of digital lock-in amplifiers categorised by their use of multipliers:
  - **Digital Switch Lock-in Amplifier**. A digital switch lock-in amplifier uses an analogue polarity-reversing switch operated at the reference frequency; these are linear but introduce the unwanted odd harmonics (Davies & Meuli, 2010).
  - **Digital Lock-in Amplifier**. A digital lock-in amplifier uses software to perform the signal multiplication, low pass filtering and to generate the reference signal.

This project will look at the lock-in amplifier which uses a digital lock-in amplifier.

2.5 Single and Dual-Phase Lock-in amplifiers

Some ‘experiments’ cause phase changes which cause a decrease in sensitivity. Single lock-in amplifiers are affected by this phase change. The single multiplying lock-in amplifier shown in Figure 2-5 uses a phase shifter to align reference signal with the experimental signal so that the signal of interest is maximised (Son et al. 2010).

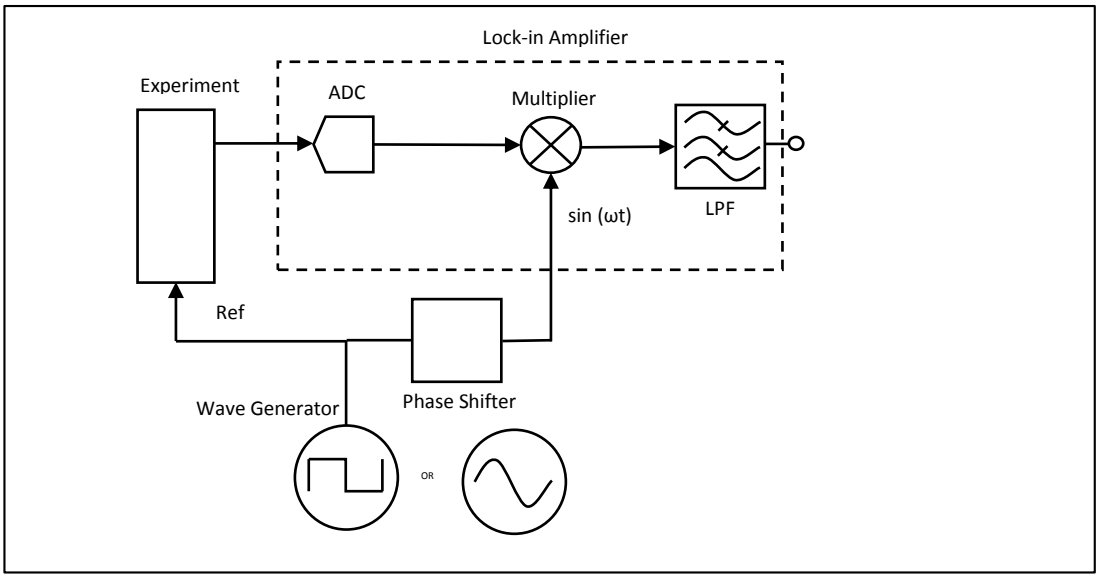

Figure 2-5: A diagram of a single multiplying lock-in amplifier, adapted from Li et al. (2011)

To overcome the need to tune the setup with the phase shifter, two multiplication blocks are introduced in to the lock-in amplifier. The two multiplication block lock-in amplifier is referred to as a **digital dual-phase lock-in amplifier** and is shown in **Error! Reference source not found.**. It shows the dual-phase multiplication of the experimental signal with both the reference signal and the reference signal phase shifted by 90° (Son et al. 2010). The signal and the reference are multiplied together to produce the multiplier output which consists of a zero frequency (DC) and harmonics. This is passed through a low pass filter to remove the harmonics and thus obtain the remaining DC signal. This signal is proportional to the input signal (Son et al. 2010).

This layout removes the need for a phase shifter to correct the alignment of the reference signal and the experiment signal (Bengtsson, 2012).

The multiplier outputs for the X channel (V_{OUTX}) and Y channel (Y_{OUTX}) are shown below where θ is the phase of the signals (not the phase output).

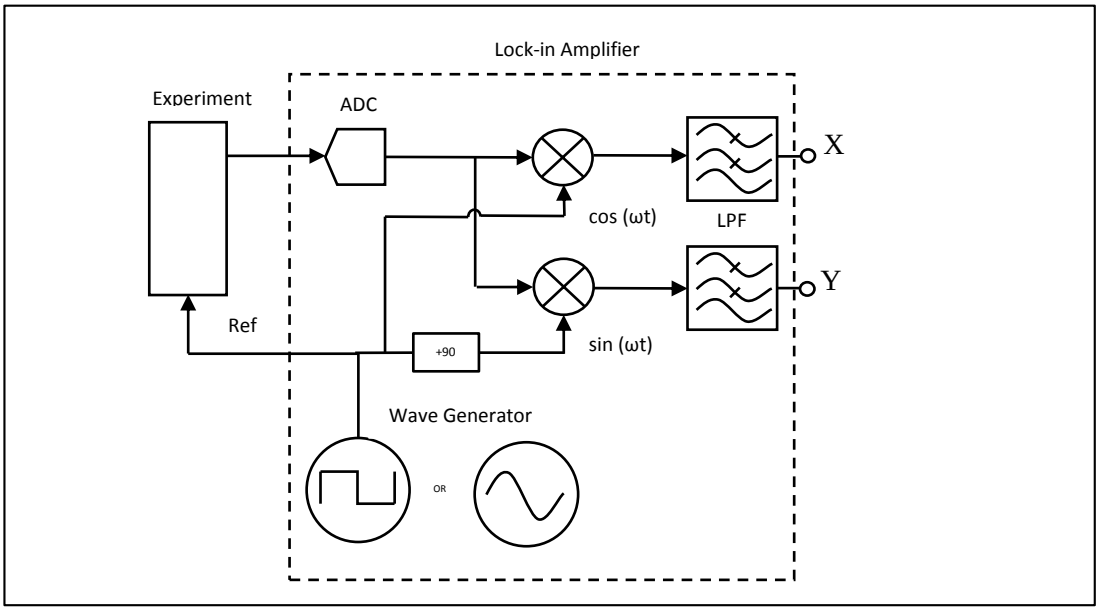

Figure 2-6: Digital lock-in amplifier, adapted from Li et al. (2011)

The following derivation shows the principle of the lock-in amplifier adapted from Li et al. (2011)

$$V_{IN} = V_{sig}\cos\left(\omega_{ref}t + \theta_{sig}\right) \tag{2.5.1}$$

$$V_{REFcos} = V_{ref}\cos\left(\omega_{ref}t + \theta_{ref}\right) \tag{2.5.2}$$

$$V_{REFsin} = V_{ref}\sin\left(\omega_{ref}t + \theta_{ref}\right) \tag{2.5.3}$$

$$V_{OUTX} = V_{IN} \times V_{REFcos} \tag{2.5.4}$$

$$V_{OUTX} = V_{sig}\cos\left(\omega_{ref}t + \theta_{sig}\right) \times V_{ref}\cos\left(\omega_{ref}t + \theta_{ref}\right) \tag{2.5.5}$$

$$V_{OUTX} = \frac{1}{2}V_{sig}V_{ref}\cos\left(\theta_{sig} - \theta_{ref}\right) + \frac{1}{2}V_{sig}V_{ref}\cos\left(2\omega_{ref}t + \theta_{sig} + \theta_{ref}\right) \tag{2.5.6}$$

$$V_{OUTX} = \frac{V_{sig}V_{ref}}{2}\cos\left(\theta_{sig} - \theta_{ref}\right) \quad \text{(After low pass filter)} \tag{2.5.7}$$

$$V_{OUTY} = V_{IN} \times V_{REFsin} \tag{2.5.8}$$

$$V_{OUTY} = V_{sig} \cos(\omega_{ref} t + \theta_{sig}) \times V_{ref} \sin(\omega_{ref} t + \theta_{ref}) \tag{2.5.9}$$

$$V_{OUTY} = \frac{1}{2} V_{sig} V_{ref} \sin(2\omega_{ref} t + \theta_{sig} + \theta_{ref}) - \frac{1}{2} V_{sig} V_{ref} \sin(\theta_{sig} - \theta_{ref}) \tag{2.5.10}$$

$$V_{OUTY} = \frac{V_{sig} V_{ref}}{2} \sin(\theta_{sig} - \theta_{ref}) \quad \text{(After low pass filter)} \tag{2.5.11}$$

The output is maximised when the phase difference between the signal and reference is zero (Son et al. 2010). The outputs are passed through low pass filters (LPF) to remove the alternating current (AC) component leaving behind the direct current (DC) amplitude. This results in equations 2.5.12 and 2.5.13

$$V_{OUTX} = \frac{V_{sig} V_{ref}}{2} \cos(\theta) \tag{2.5.12}$$

$$V_{OUTY} = \frac{V_{sig} V_{ref}}{2} \sin(\theta) \tag{2.5.13}$$

The two outputs of the dual-phase multipliers (V_{OUTX} & V_{OUTY}) are converted into the magnitude of the signal and phase difference between the signal and the reference with equations 2.5.14 and 2.5.15 (Li et al. 2011). When the reference signal is 1 volt and the phase (θ) is zero the output is maximised. The magnitude does not depend on the phase difference between the signal and the reference signal (Aguirre et al. 2011).

$$Magnitude\ (V_{sig}) = 2 \times \sqrt{{V_{OUTX}}^2 + {V_{OUTY}}^2} \tag{2.5.14}$$

$$Phase\ (\varphi) = \tan^{-1}\left(\frac{V_{OUTY}}{V_{OUTX}}\right) \tag{2.5.15}$$

The importance of the dual-phase lock-in amplifier is that for this project the phase shifter and additional components are not required. The signal will be maximised and the project can be simplified.

2.6 General Considerations

Second Harmonic

Use of the 2nd harmonic. Son et al. (2010) identified that $1/f$ noise exists at the DC and low frequency range of sensor output signals. If the signal is lower than the $1/f$ noise amplitude the DC lock-in method will not detect the signal. He suggests that using the second harmonic of the reference signal which is away from the $1/f$ noise will recover a signal with a better SNR than that at the reference frequency.

Tolerance accuracy

The general tolerable noise level of a lock-in amplifier is one that does not affect the output more than a few percent above the input amplitude (web document thinksrs, 2013).

Output noise immunity

Aguirre et al. (2011) proposed a low power *analogue lock-in amplifier* for portable sensing which was able to recover signal information from a SNR of -24 dB with errors below 6%. These levels may be a bench mark to compare the performance of the DSP based lock-in amplifier.

The SNR is a measurement to determine the amount of noise present on the signal of interest is calculated by equation 2.6.1 adapted from Li et al. (2011).

$$SNR = 10 \times \log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)\ (dB) \qquad (2.6.1)$$

Lock-in amplifier cost

A simple google search for 'Lock-in amplifier cost' points to a link showing the price of a Stanford Research Systems lock-in amplifier priced from $4150. This quick search shows the large expense of dedicated lock-in amplifiers and the need to reduce this cost for use in reasonable inexpensive remote sensors.

Chapter 3 Development Methodology

3.1 Chapter Overview

This chapter discusses the methodology used to carry out this project. This includes:

- Testing environment
- Constraints (2nd harmonic, physical hardware, and other things not taken into account in this project)
- Constants (Noise, Phase, 2% acceptable output, etc.)
- Algorithm development

3.2 Testing Environment

The final target implementation environment of the output of this study is, either:

- DSP
- microprocessor
- combined DSP plus microprocessor such as dsPIC processor

All the above may be suited to the task, but because of the range of capabilities and prices of various hardware components, it is better to delay this selection until more is understood and known.

Testing a range of hardware with different ADC resolutions and sample frequencies would be:

- expensive
- time consuming
- difficult to replicate the same experimental conditions
- uncertainty of errors and anomalies

Therefore MATLAB was chosen as a hardware emulator for development and testing so that there would be no hardware limitation or constraints. This will facilitate obtaining better general specifications of the target implementation environment.

3.2.1 Lock-in Amplifier in MATLAB

MATLAB was used to develop and test the digital lock-in amplifier algorithms by varying specific parameters to simulate the restrictions of a reasonable priced microprocessor and DSP's. In this way the project was able to test whether the use of DSP and microprocessors was reasonable in terms of it accuracy and noise immunity. That is, is this a good way to go?

One of the outcomes of this project was to determine the required microprocessor specifications suitable to implement a lock-in amplifier.

MATLAB allows us to identify the specifications using MATLAB's inherit benefits which are:

- low cost (availability)
- reduce hardware setup and test time
- consistent test conditions
- the ability to change simulation variables easily
- reduce errors where the source of the error would not be easily determined
- good representation of the data
- implement large volume of test iterations
- uses a C like programing language which is reasonable easy to transfer to real world microprocessors

The limitations of the MATLAB program include:

- The gap between the pseudo representation of hardware architecture (not even emulation) and the implementation in physical hardware will result in lot of issues which will not be resolved in this project.
- MATLAB will be generally slower than a physical implementation. This, however, is not necessarily a bad thing whilst it will increase the iteration of testing the real world implementation will be expected to be faster.

Based on the analysis of the results from the MATLAB program, the microprocessor specifications can be suggested. The MATLAB program was used to determine the microprocessors ADC resolution and sample frequency, other microprocessor specifications that are to be taken into consideration will include:

- price
- size
- type of software language used on the microprocessor
- development systems or programmers required

The implementation of a digital lock-in amplifier using a microprocessor or DSP needs to have minimum number of components. Therefore the aim is to maximise the microprocessors ability to produce the reference signal internally and its inbuilt analogue to digital converter (ADC). This drove the project to the use of a dual-phase lock-in amplifier.

3.3 Constraints

Test for increased noise immunity. In the research literature revealed that increased noise immunity could be obtained by using 2^{nd} harmonics. (Refer 2.6 above) This project uses simpler algorithms at this stage, so as to minimise costs in specified hardware. It was felt that if the outcomes showed that the noise immunity was insufficient, then implementation of a more complicated algorithm to use 2^{nd} harmonics may be warranted.

Thus the 2^{nd} harmonic was not used in this project but considered worth investigating in a subsequent project, if required.

Implementation in physical hardware. This project specifically avoided implementation in physical hardware for the reasons given above. (Ref 3.2.1)

Time. The memory capacity of the programs host computer resulted in it not being able to run enough random noise samples to produce an exact representation of a Gaussian noise.

ADC simulation limitations. For simplicity the MATLAB program did not simulate the ADC sample frequency fully. It did not simulate the sampling of a near continuous waveform and keeping the reference signal at a constant amount of samples for each sample frequency change. Instead, an experimental signal was produced to match the number of the samples in the reference signals for each ADC sample frequency change;

Signal Amplitude. The simulated experimental signal amplitude was not changed to reflect how the physical experiment amplitude would change. The simulated experimental signal amplitude was set at 1.8 volts to maintain a level of constancy throughout the simulations.

3.4 Constants

Noise. To see how the lock-in amplifier would perform under different SNR conditions, it was planned to develop and apply simulated white noise to the experimental signal.

Phase. It was planned to implement a dual-phase lock-in amplifier because it reduced the amount of code required. The time required for development of code to simulate a phase shifter, and possibly a phase locked loop (PLL), would seriously compromise the project. The simulated phase delay of the experimental signal therefore can be set at an arbitrary constant.

2% acceptable output. It was important to identify a point where the project could determine that a true positive signal was 'seen'. The literature search (Ref 2.6 above) suggested that a few percent of the signal was acceptable. The project supervisor nominated 2% as the point of determining true positive results.

Reference frequency. The project supervisor identified 2000 Hz to be used as the reference frequency. This was based on a physical experiment currently running in the laboratory.

3.5 Algorithm Considerations

Algorithm development is the core of the project and is discussed more fully in subsequent sections of this document (Ref Chapter 4 below). The following general comments apply regarding algorithm development.

A review of the current literature was used to determine the equations to be used for the digital lock-in amplifier. The review also investigated the principles behind the equations and how they are used to perform the lock-in function. Derivations using the lock-in amplifier equations were written out by hand to form a better understanding of their operation (Ref 2.5).

An understanding of analogue lock-in amplifiers was first developed to form a knowledge foundation of the lock-in amplifier. This lead to investigating the different variations (Ref 2.4) of digital lock-in amplifiers. Digital lock-in amplifiers which use digital signal processes (DSP) and dual-phase multipliers were mainly considered.

3.5.1 Lock-in Amplifier Algorithms in MATLAB

A MATLAB program was written such that each of the parameters that simulate the microprocessor specifications could be easily changed to rerun the program for the different parameters.

Specifically, the lock-in amplifier algorithms to function correctly, and provide adequate outputs, the simulated environment requires certain components to be present or simulated. The following lists these components:

Input for the simulated 'experiment'

- Reference signal (constant amplitude and frequency at a constant phase)
- Simulated experimental signal
- Noise injection

Output from simulated 'experiment'/ input to simulated lock-in amplifier

- Derived SNR from experimental output
- ADC sample frequency simulation

Simulated components

- ADC bit resolution simulation
- Low pass filter
- Lock-in amplifier amplitude output

Output of simulated lock-in amplifier

- Average error squared output (σ^2)

Maximum flexibility of parameters was chosen to drive the simulation so as to obtain a maximum understanding of the requirements of the future implementation hardware.

3.5.2 Outputs of this Study

This project has a number of specified objectives which are listed below, but there are a number of other areas of understanding that are desirable and useful for determining the feasibility and specification of a future hardware implementation.

Specified Project Objectives

1. Determine the set of algorithms to be implemented for the lock-in, including the reference signal multiplication, low pass filtering, and the separate phase-locked loop.
2. Evaluate the performance of the algorithms as implemented, and show the signal recovery performance for various parameter settings at different SNR levels.
3. Evaluate the performance in MATLAB using sampled real-world signals.
4. Investigate suitable processor architectures and development systems for an embedded lock-in amplifier.

Desirable information to be gained from this project

- Is the implantation of lock-in amplifiers on a DSP or microprocessor feasible? Or is the technology just not there yet?
- What is the minimum ADC bit resolution that will give ‘acceptable’ results?
- Is there an optimal ADC bit resolution. Or is higher resolution better? Is the ‘quality’ linear?
- How does the ADC sample frequency effect the result?
- Does Integration time affect the results? If so, how?
- Is there a relationship between sample frequency, bit resolution and integration time? Can one be traded off for the other?

3.6 Microprocessor Selection

Based on the analysis of the results from the MATLAB program the microprocessor specifications can be suggested. The MATLAB program was used to determine the suitable ADC resolution and sample frequency, other microprocessor specifications.

These specifications can be then used to determine suitable microprocessor or DSP hardware and final choices can be determined taking into consideration:

- price
- size
- type of software language used on the microprocessor
- development systems or programmers required

The actual selection of hardware and the implementation are not seen as part of this project.

Chapter 4 Algorithm Development

4.1 Chapter Overview

This chapter will discuss the implementation of the lock-in amplifier algorithms in MATLAB. The chapter will follow the development of the MATLAB program from a basic lock-in implementation to simulate a range of microprocessor specifications. The plots produced during the development of the program were used as a visual check will also be discussed. The processing of an external sampled signal will be discussed.

The final main MATLAB program which performed the microprocessor specification simulation is listed in the Appendix B.

4.2 Algorithms Variations

The following subsections demonstrate the progression of the lock-in amplifier algorithms implemented in a MATLAB program.

A version of the program using a square wave reference was tested. The main program (Appendix B) can easily be changed to use a square reference signal. The outputs for the square wave reference version were similar to the outputs of the sinusoidal reference wave version. For this reason the results were not duplicated using the square wave reference.

The values of the SNR, average error^2 and amplitude out for each ADC bit resolution were stored for future plotting. The MATLAB program used to produce the plots from the data produced by the final main MATLAB program is listed in the Appendix B.

4.2.1 Lock-in Amplifier in MATLAB

The start of the MATLAB program was written to create the experimental signal and the two reference signals. These signals were made into matrices that had a time period of 0.1 seconds in length at a constant frequency of 2 kHz.

To simulate the increase or decrease in ADC sample frequency the amount of values per cycle for each signal matrix were increased or decreased. This was done while maintaining the correct frequency and the same time period. The sample frequency was set to 80 kHz as a starting figure.

The experimental signal matrix (signal) was made to be a sinusoidal cosine waveform with a phase delay of $\pi/10$ radians and an amplitude of 1.8 volts.

The first reference signal matrix was made to be a sinusoidal cosine waveform with zero phase shift. The second reference signal matrix was made to be a sinusoidal cosine waveform with a phase delay of 90 degrees (which becomes a sine wave). Both reference signal matrices have an amplitude of 1 volt. The signal and the two reference signals were plotted to see if their phases are correct shown in Figure 4-1.

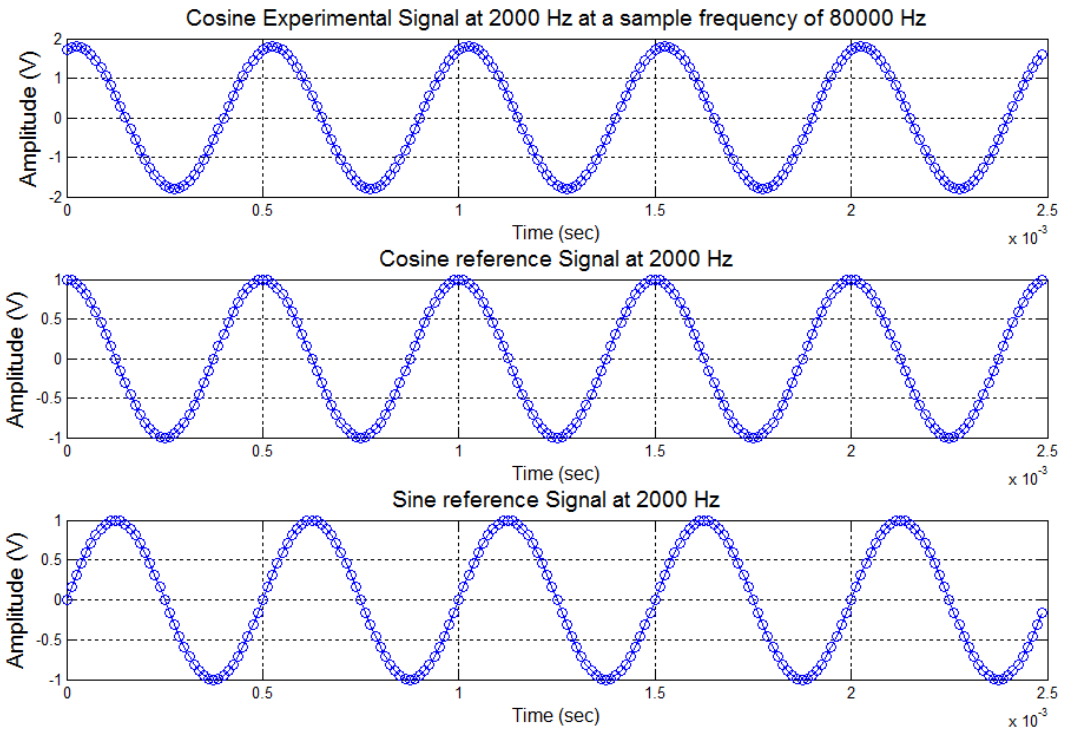

Figure 4-1: Plot of signal and reference signals

The lock-in amplifier equations were programed next. The two reference multiplication stages were programed using the equations in section 2.5.

The signal matrix is multiplied element wise by the cosine reference signal matrix as well the signal matrix is multiplied element wise by the sine reference signal matrix. This forms two separate matrices which are equivalent to equations (2.5.6) and (2.5.10) (refer to 2.5). Figure 4-2 and Figure 4-3 below show the signal and the reference signals with the result of the multiplication below each pair.

The two output matrices from the multiplication stage require filtering. A moving average low pass filter was written and tested in the MATLAB program but it took a lot of processing time so it was removed.

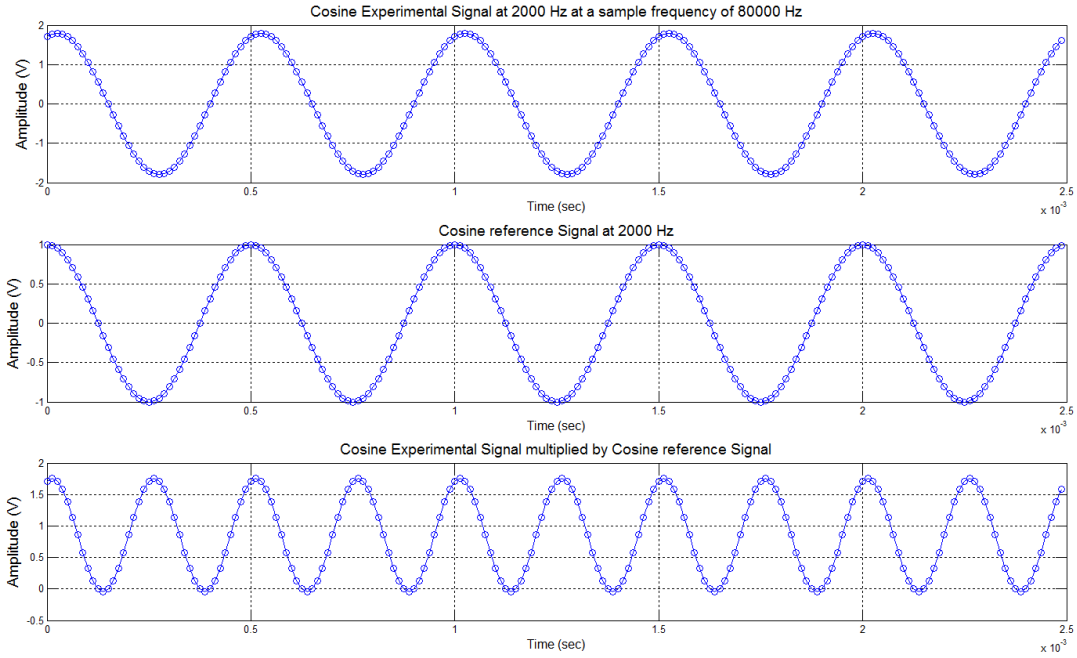

Figure 4-2: Result of multiplication of signal and reference cosine signal

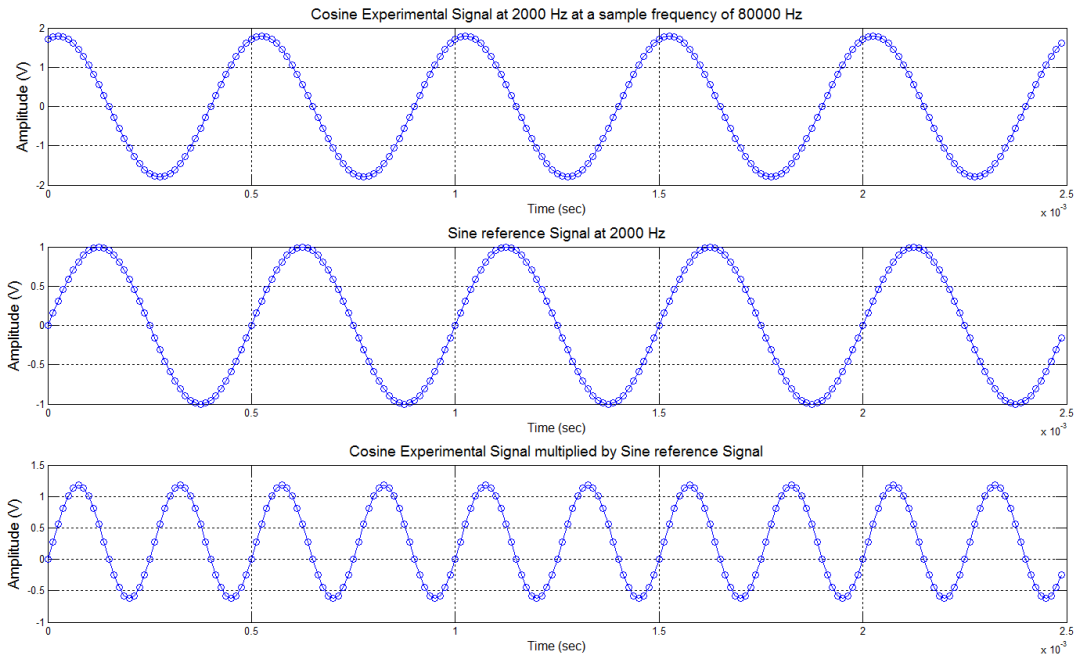

Figure 4-3: Result of multiplication of signal and reference sine signal

The moving average low pass filter was replaced by a low pass filter which used MATLABs 'mean' function. The mean function found the DC value of the multiplication stage matrices. The output of the filters produced a single value for each of the X and Y channel outputs. The X and Y channel outputs are then used to find the amplitude of the signal and phase difference between the signal and the reference signal.

The outputs of the lock-in amplifier algorithms are two values, one for the amplitude signal and one for the phase of the signal, for the set integration time period.

The MATLAB program for the final correctly functioning lock-in amplifier algorithms (without added noise and ADC resolution) is listed in the Appendix B.

The preceding implemented the lock-in amplifier algorithms. To test the lock-in amplifiers performance the amplitude and phase of the output were compared with the input amplitude and phase to see if they were equal. The input and the output should be equal because there is no noise added to the signal. Table 4-1 shows the lock-in amplifier input and output values.

Table 4-1: Table of input and output values for amplitude and phase for zero noise

Variable	Input	Output
Amplitude	1.800	1.800
Phase delay of signal	$\frac{\pi}{10} = (0.31416)$	0.31416
Noise	0.00	SNR: Inf dB

The phase output results were not displayed in future program versions after it was seen that the algorithms were functioning correctly, as it had no effect on the amplitude output. Furthermore the phase was held constant for all following simulations (so would not change). Table 4-2 below shows that the output of the lock-in amplifier is not effected by the phase delay of the signal.

Table 4-2: MATLAB program output for various signal phase delays

Phase delay of signal		Amplitude	
Input (radians)	Output (radians)	Input (V)	Output (V)
$\frac{\pi}{10} = (0.31416)$	0.31416	1.800	1.800
$\pi = (3.1416)$	3.1416	1.800	1.800
$2\pi = (0.0000)$	-2.8267e-014 ≈ 0	1.800	1.800
$\frac{4\pi}{3} = (4.18 - 2\pi = -2.094)$	-2.0944	1.800	1.800
$\frac{25\pi}{87} = (0.902756)$	0.90276	1.800	1.800

4.2.2 SNR of Input Signal

After the lock-in amplifier algorithms functioned correctly by outputting the input values noise was added to the signal to test the lock-in amplifier noise rejection performance. The noise was added using MATLABs ‘randn’ function which produces a random number with a Gaussian distribution. The random noise was made into a matrix the same length as the signal so that it could later be added to the signal. This random matrix was then multiplied by a noise level multiplication factor. The multiplication factor was a range of increasing incremental values to simulate the level of noise increasing in the ‘experiment’. The multiplication factor range will be discussed later in this section. Figure 4-4 show the signal with no noise and the signal with two increasing noise multiplication factors.

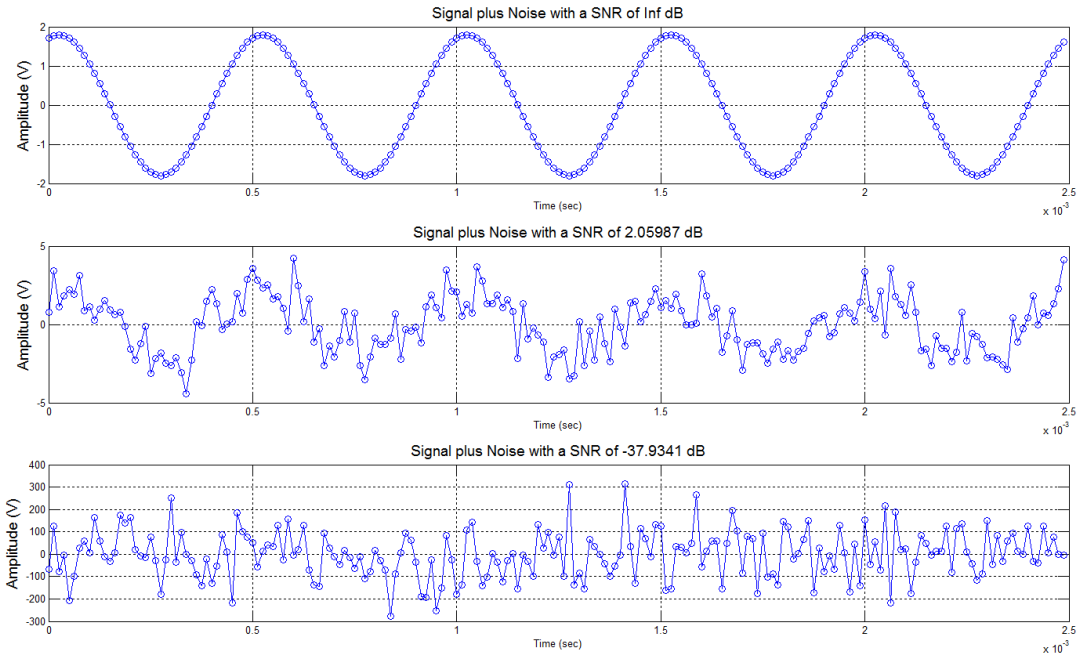

Figure 4-4: Noise add to the signal at SNR of Inf dB, 2 dB and -37.9 dB.

The signal to noise ratio (SNR) of the signal was used to determine the level of noise immunity at which the lock-in amplifier could still produce an acceptable output. The SNR of the signal was calculated using the signal power and noise power using equation 2.6.1 in section 2.6.

The amplitude of the lock-in amplifier output was plotted against the SNR for the range of noise multiplication factors. It was expected that as the noise increased that the amplitude would increase and the output would not be acceptable. The question was then raised 'How far above the input amplitude (1.8 V) could the output get before it was deemed unacceptable?' A value of two percent above the input amplitude was chosen as discussed previously (Ref 3.4). The two percent error was marked on the output amplitude plot to give a visual representation and the value of the SNR at this point was also displayed in the title of the plot.

The first version of code used a noise multiplication factor range from 0.1 to 1000 with 100 steps spaced linearly over the range. This large range was used because the lock-in amplifier algorithm performance was unknown. Figure 4-5 shows the approximate output range of the lock-in amplifier, seen to the left of the plot the values below -40 dB do not need to be calculated. Therefore the range of noise multiplication factors can be reduced to save computation time.

The error between the input and the output was calculated to monitor the output error as the noise level increased. This only produced one number for each level noise in the range, the value of this would change each time the program was run due to the different random noise. It was recommended by the project supervisor to run 1000 sets of random noise. This amount of random samples was tried but could not be run successfully on the host computer at that time (it kept on crashing). A new host computer was purchased, this computer was able to run the 1000 sets successfully. The 1000 sets of random noises were averaged so that a more appropriate effect of the Gaussian noise was achieved. So 1000 sets of random noises were averaged and squared to produce an average error squared (σ^2) for each noise level. This σ^2 is the square of the average of the error between the input amplitude and the output amplitude for the set of random numbers. A straight line approximation was applied to the average error^2 plot which can be found in Appendix C.

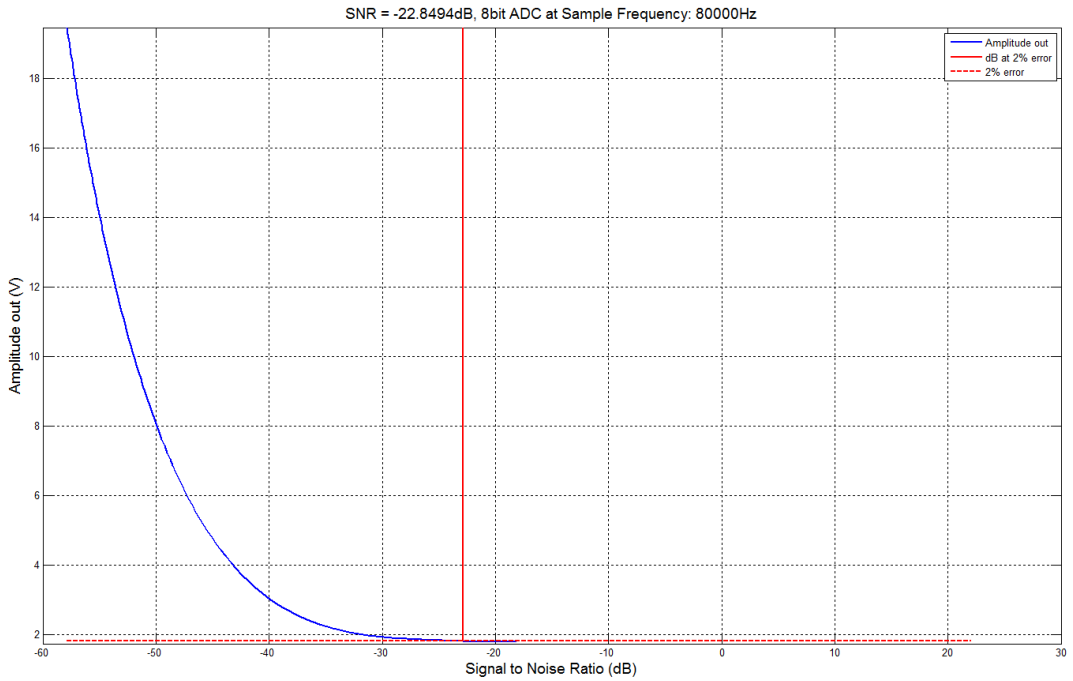

Figure 4-5: Amplitude out using linear spacing for a range from 0.1 to 1000

The noise multiplication factor range was reduced to a range of 0.1 to 100 to save on program run time plot can be seen in the Appendix C.

This average error squared was plot against the SNR of the input signal this relationship can be seen in the Appendix C.

The multiplication factor was then changed to a logarithmic spacing because the average error squared was plotted on a logarithmic y axis. The logarithmic scale visually produced a straight line and the logarithmic spacing of the noise produced an evenly spaced line on the average error squared plot shown in Figure 4-6. The noise multiplication factor using the logarithmic spacing was first run from 0.1 to 1000 with 100 steps. After the program was run this logarithmic noise multiplication factor range produced a slightly better result than the linear spacing, because the logarithmic spacing provided more data points at the acceptable output amplitude.

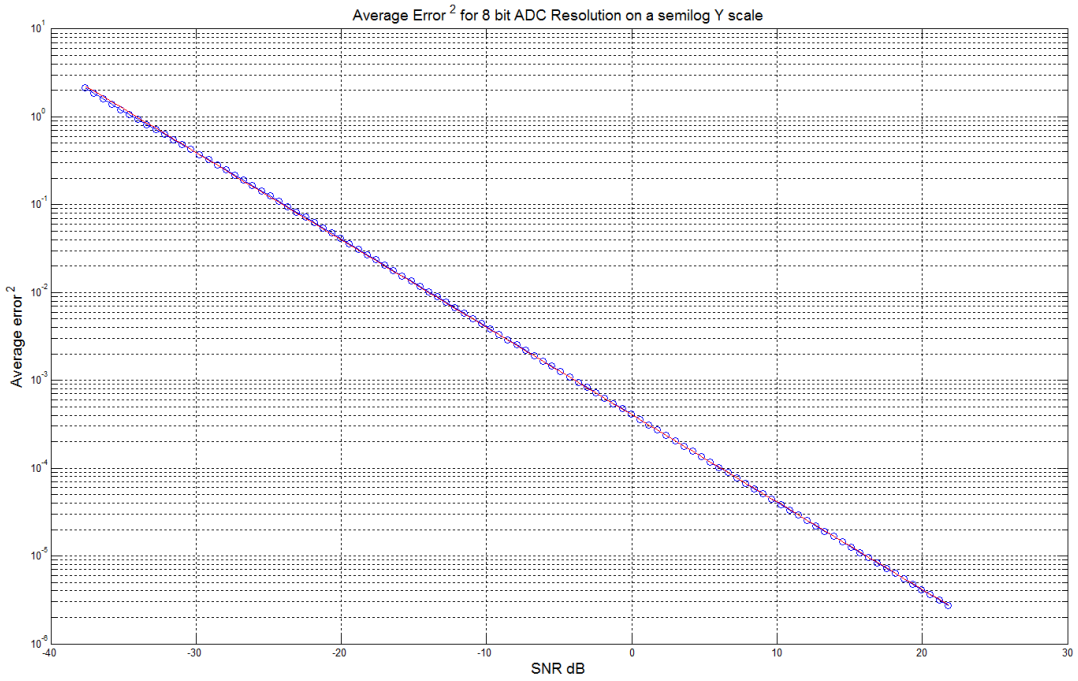

Figure 4-6: Average error^2 using logarithmic spacing on a logarithmic y axis

The noise multiplication factor using the logarithmic spacing was reduced to 0.1 to 100 with 100 steps. The acceptable output using the new range occurred within this 0.1 to 100 range.

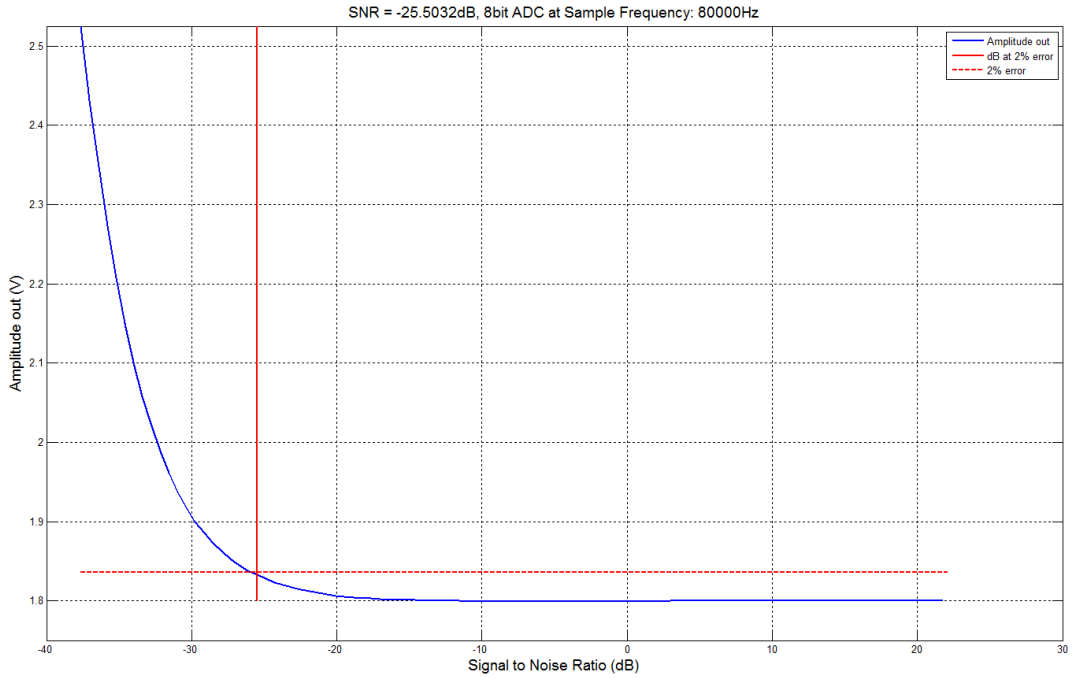

Figure 4-7: Amplitude out using logarithmic spacing for a range from 0.1 to 100

4.2.3 Simulating a Microprocessors ADC

Signal quantisation was used to simulate a range of analogue to digital converter (ADC) bit resolutions. A range of two bit to sixteen bit was used. The quantisation was performed on the signal matrix after the noise had been added to it. The signal matrix was converted to a sixteen bit integer then quantised to the required bit resolution. Then the signal matrix was converted back to double perdition format.

An outer loop was added to the program to determine the effect of changing the ADC bit resolution from 2 bit to 16 bit. While testing the quantisation by visually checking the quantised signal plot (Figure 4-8) the noise was set to zero.

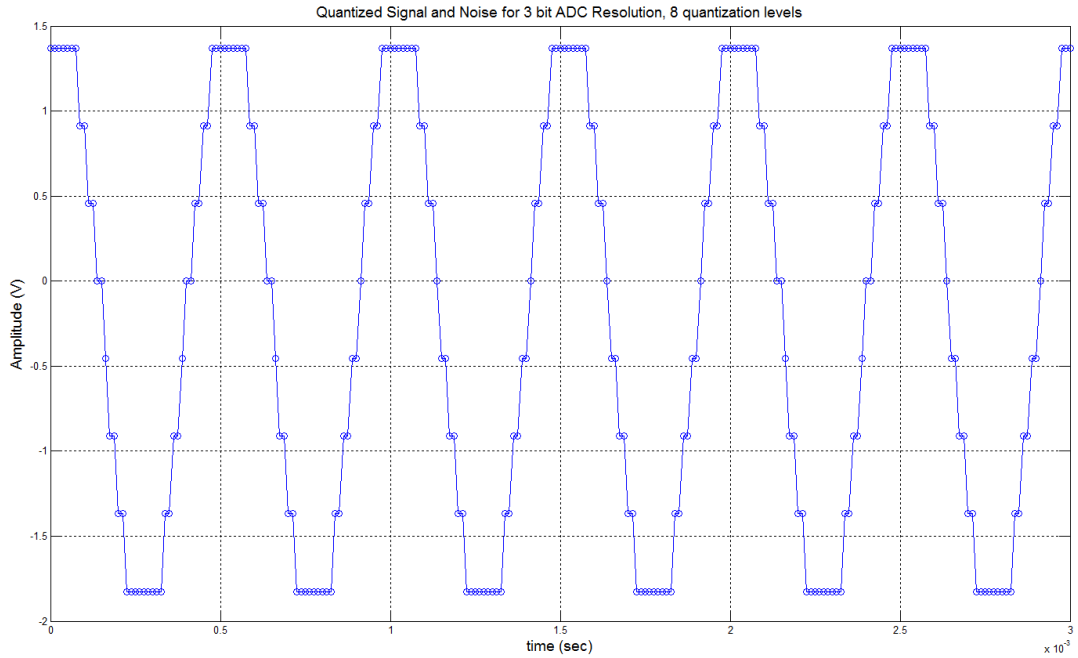

Figure 4-8: Experimental signal quantised to 3 bit with zero added noise.

A loop was made to find specific SNR values and their corresponding average error^2 value from the straight line approximation matrix. This formed the main plot to compare the different ADC resolutions. The average error^2 for each decibel value from 20dB to -30dB in 5dB steps was stored for each ADC bit resolution increase. Figure 4-9 shows the relationship between the average error^2 and the increased ADC bit resolution.

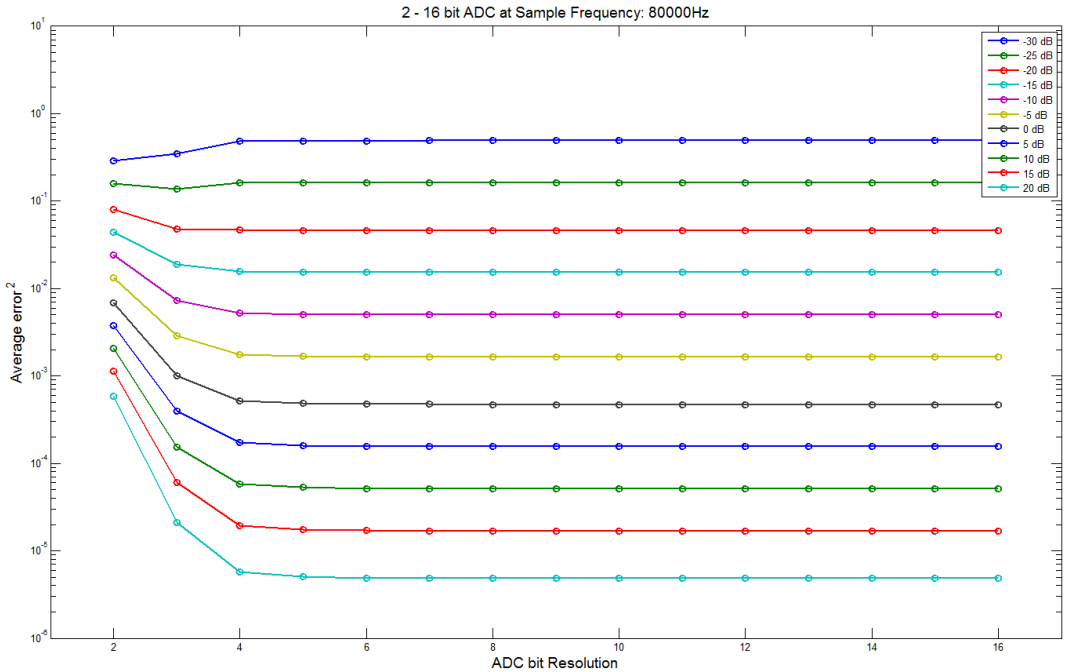

Figure 4-9: Average error^2 compared to ADC resolution for different SNR levels.

4.2.4 MATLAB 'seed' Function

After the ADC bit resolution loop was added it was noticed that the results were inconsistent for each time the program was run and for different ADC resolutions. The '*seed*' option of MATLABs *randn* function was used so that each bit resolution change would have the same set of random noises. This was so that the difference between each result was only caused by the ADC bit resolution changing. The plot of amplitude out without the 'seed' option used is shown below in Figure 4-10.

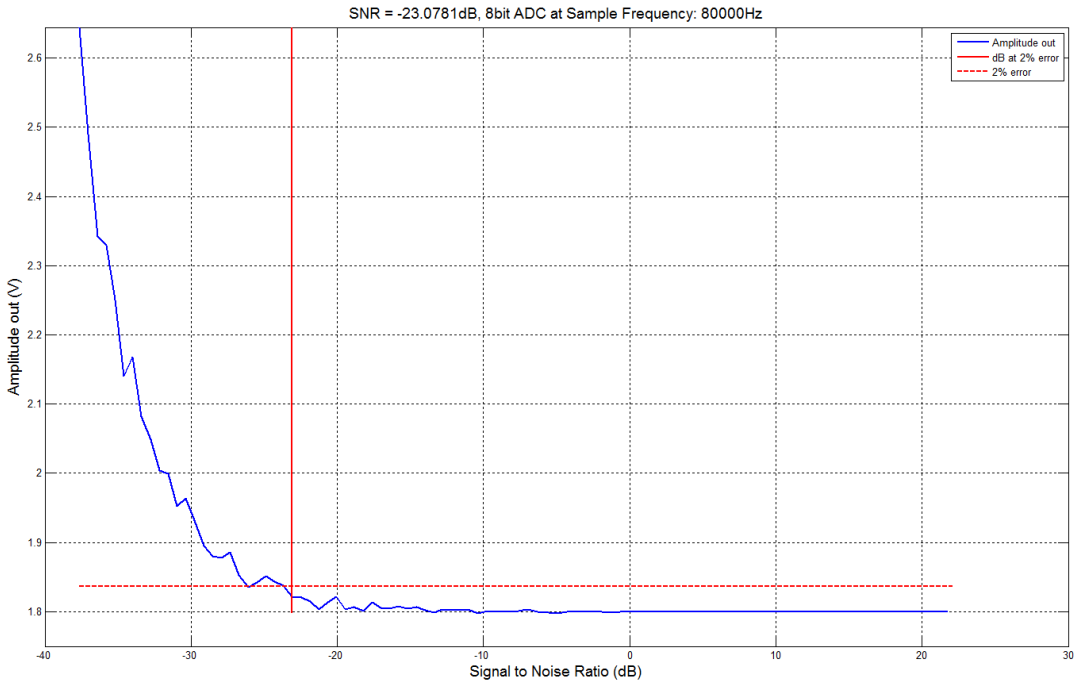

Figure 4-10: Amplitude out when no 'seed' option was used in the randn function

4.3 Results of Final Program

The final program was run with different sample frequencies, which were changed manually because of the time taken to run the program was too long and memory intensive. The final MATLAB program used to simulate the ADC specifications can be found in the Appendix B.

Critical information discovered during the programs development that improved the results were the use of the 'seed' option in the *randn* function and the number of random noises sets being averaged. The variable which controlled the amount of averaged random noise sets showed that a higher number (1000-5000) resulted in a smoother amplitude out curve.

The SNR level at the 2% error threshold fluctuated up and down as the sample frequency increased at a 1000averaged sets of random numbers and below.

With 1000 averaged sets of random numbers from a range of multiplication factors from 0.1 and 100. The SNR for the each ADC bit resolutions change over a range of ADC sample frequencies is displayed in Table 4-3.

Table 4-3: Signal to noise ratio (dB) at the 2% error output for 1000 sets of random noise.

fs	ADC bit Resolutions														
(kHz)	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
128	19	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25
80	19	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23
64	18	-24	-24	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25	-25
32	18	18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18
16	16	16	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20
8	16	16	-13	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12

The SNR level at the 2% error threshold decreased consecutively as the sample frequency increased at 5000 averaged sets of random numbers.

With 5000 averaged sets of random numbers from a range of multiplication factors from 1 and 31.6 the SNR for the each ADC bit resolutions change over a range of ADC sample frequencies is displayed in Table 4-4.

Table 4-4: Signal to noise ratio (dB) at the 2% error output 5000 sets of random noise.

fs (kHz)	ADC bit Resolutions														
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
128	-1.9	-26	-26	-26	-26	-26	-26	-26	-26	-26	-26	-26	-26	-26	-26
80	-17	-25	-25	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23
64	-15	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23	-23
40	-16	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20	-20
32	-15	-19	-19	-19	-19	-19	-19	-19	-19	-19	-19	-19	-19	-19	-19
16	-15	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18
8	-14	-14	-14	-14	-14	-14	-14	-14	-14	-14	-14	-14	-14	-14	-14

It is thought that the average of 1000 sets of random noises did not represent the Gaussian distribution effectively but the 5000 sets seemed to according to Table 4-4.

Using a high number (5000) of averaged random noise sets (without the 'seed') showed that the simulation using the 'seed' function for 1000 random noise sets is close to the values received using a high number of sets as shown in Table 4-5.

Table 4-5: No seed function used for 5000 sets of random noise.

fs (kHz)	ADC bit Resolutions														
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
80	-15	-23	-23	-23	-25	-25	-24	-23	-24	-23	-23	-24	-24	-23	-23

The average of the average error^2 decreased as the sample frequency increased.

The Table 4-6 below shows the average of the average error^2 for 5000 averaged sets of random numbers from a range of multiplication factors from 1 to 31.6.

Table 4-6: Average error squared at the 2% error output ($x10^{-3}$) for 5000 sets of random noise.

fs (kHz)	ADC bit Resolutions														
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
128	35	25	24	23	23	23	23	23	23	23	23	23	23	23	23
80	52	40	38	38	37	37	37	37	37	37	37	37	37	37	37
64	66	51	48	48	47	47	47	47	47	47	47	47	47	47	47
32	125	100	95	94	94	94	94	94	94	94	94	94	94	94	94
16	228	192	183	181	181	180	180	180	180	180	180	180	180	180	180
8	421	362	350	346	345	345	345	345	345	345	345	345	345	345	345

When the MATLAB lock-in amplifier algorithm was run using a one second integration time the results were improved by approximately -10 dB as shown is Figure 4-11. A full data set using a one second integration time was not able to be produced due to the host computer limitations.

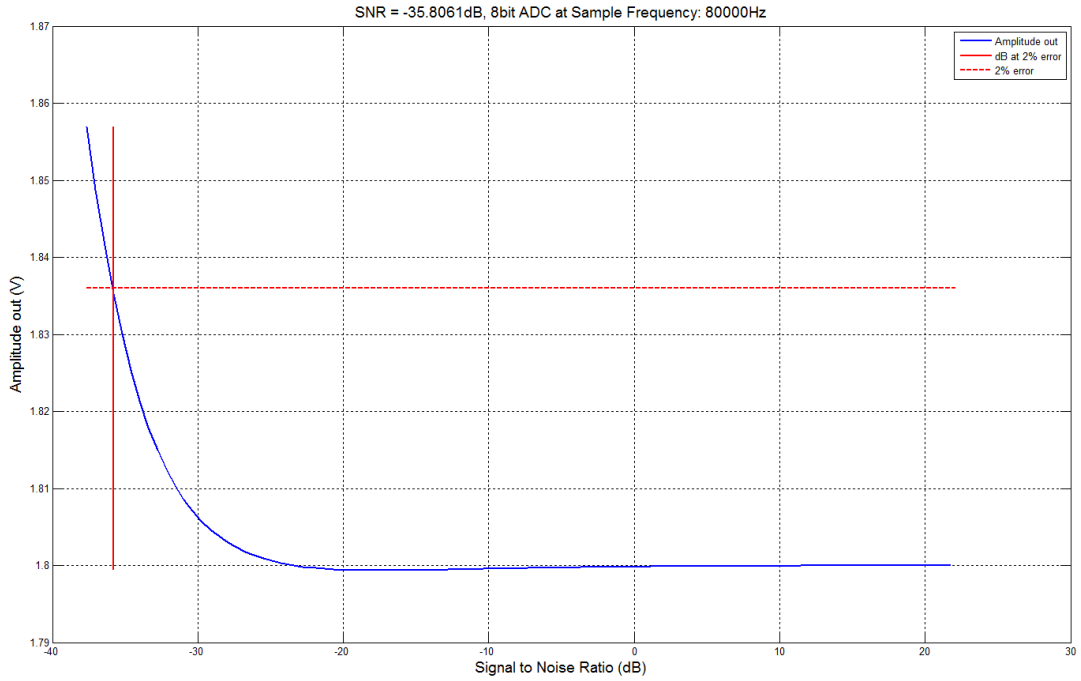

Figure 4-11: Amplitude out compared to SNR for an integration time of 1 s

4.4 Input Real Signal

The experiment included a waveform generator which produced the cosine and sine reference signals. The cosine waveform was used as a reference signal for a LED driver which pulsed the infrared LED. The light from the LED travelled through a gas chamber and received by a photo detector. The signal out of the photo detector is the experimental signal which would be feed into the lock-in amplifier. The experimental signal, the cosine and sine reference signals were sampled using a USB Adlink data acquisition unit. These three signals were also displayed on an oscilloscope for visual representation, a screen shot of this is included in the Appendix C. The sampled data from the acquisition unit was stored on a laptop for later analysis in MATLAB.

The sampled data was imported into MATLAB for processing with the lock-in amplifier algorithms. The sample was loaded into MATLAB using a premade program (supplied by the project supervisor) which placed the data into matrices. A new program was written which used the lock-in algorithms without the ADC and noise loops. Different integration times were used to see the variation this had on the results of the lock-in algorithms.

The sampled signal MATLAB program is listed in Appendix B.

The plot in Figure 4-12 is of the sampled experimental signal and the reference signals with gas present, the sampled signals without gas are in Appendix C. The product of the signal multiplication stages for the gassed sample are shown in Figure 4-13.

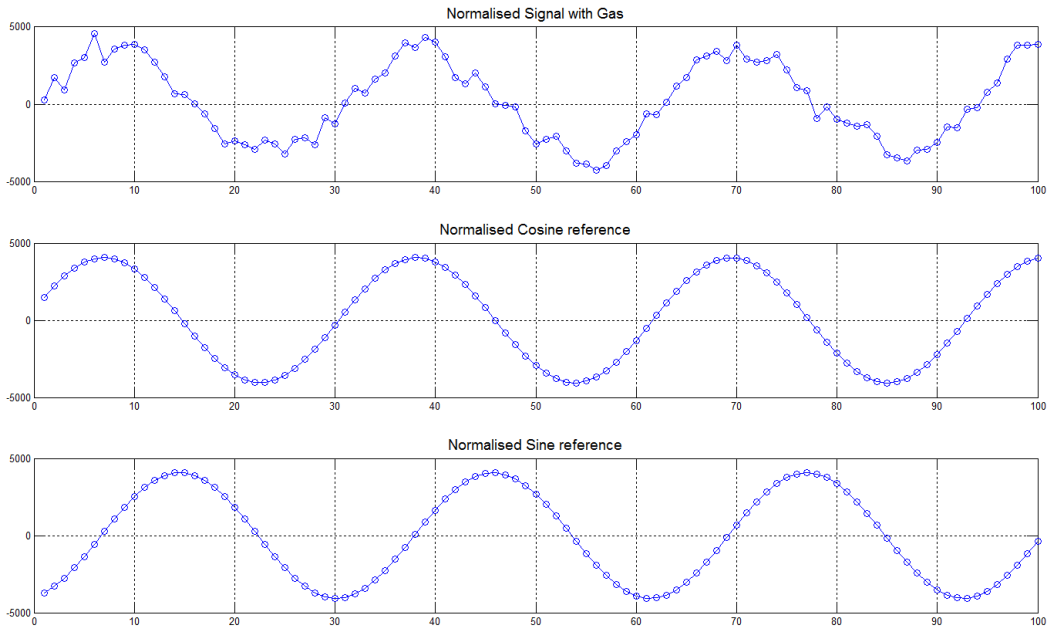

Figure 4-12: Samples signal with gas present.

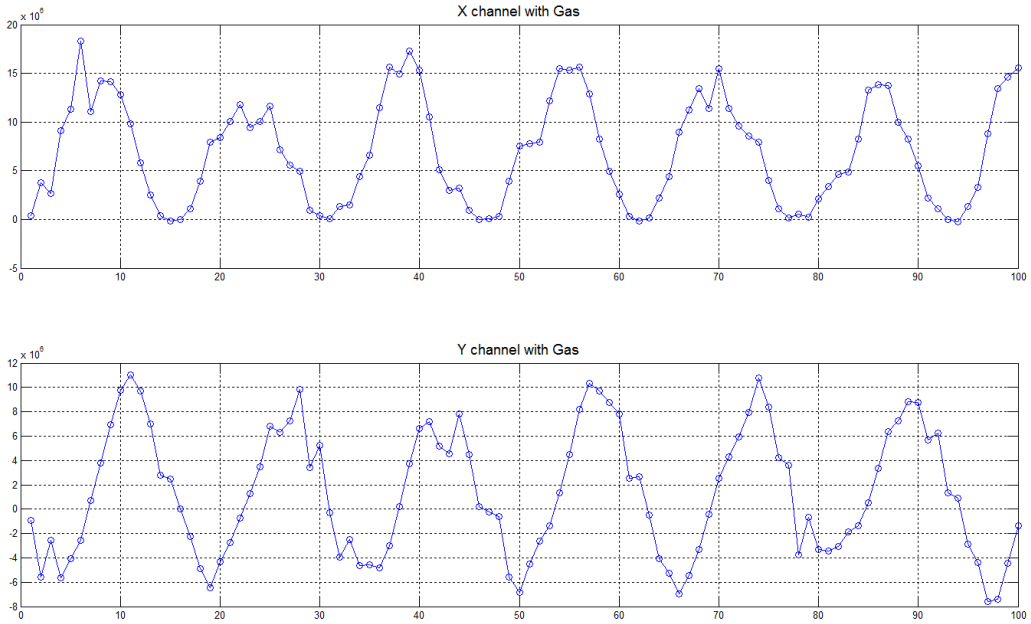

Figure 4-13: X and Y channels before LPF

Table 4-7 shows the output of the lock-in amplifier algorithms for an integration time of 0.16 s. The output amplitude of the gassed sample was 3.6% reduced from the non-gassed sample.

Table 4-7: Lock-in amplifier algorithm outputs for a sampled signal.

	No Gas Present	Gas Present
Amplitude	1.382	1.332
Phase	1.3235 radians	1.3235 radians

Chapter 5 Analysis

5.1 Chapter Overview

For clarity the graphs in this analysis section are a representation only. They have been generalised from the results. Please refer to the results for the exact relationships.

5.2 Analysis of Results and Plots

ADC resolution: The Figure 5-1 below shows that the noise immunity increased as the ADC resolution increased. Good results were obtained from using a 6 bit ADC resolution. ADC resolutions above 8 bit showed no discernable increase in noise immunity.

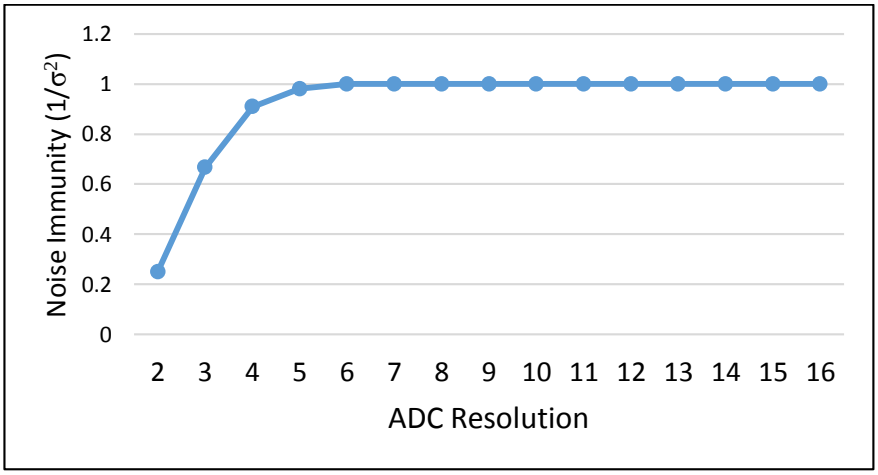

Figure 5-1: Noise Immunity compared to ADC Resolution

ADC sample frequency: An increase in the sampling frequency improved the noise immunity. For a doubling for the sampling frequency the noise immunity improved by approximately 3 dB as shown in Figure 5-2.

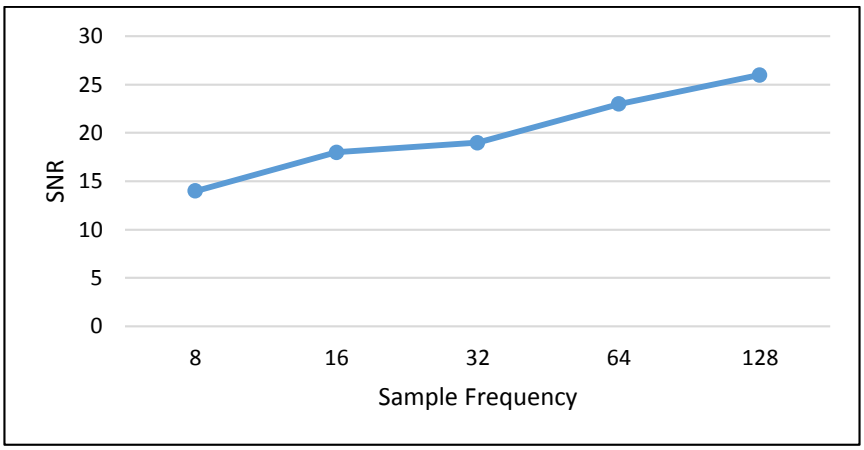

Figure 5-2: SNR compared to sample frequency

Integration time: An increase in integration time increased the noise immunity. Increasing the integration time from 0.1 s to 1 s increased noise immunity by 10 dB. As Figure 5-3 shows this relationship between SNR and integration time is not linear.

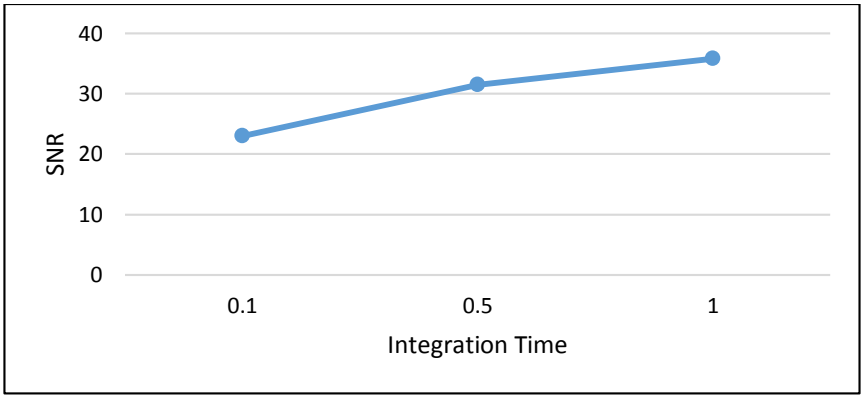

Figure 5-3: SNR compared to Integration time

Therefore a combination of specifications produces an acceptable result. Increasing the specifications improves the noise immunity of the lock-in amplifier.

5.3 Real Sampled Signal

The lock-in amplifier algorithms were tested with a real sampled signal from the output of an experiment involving an infrared gas sensor. Two tests results were analysed one without gas and the other with gas present. The results showed that the phase output did not change with the addition of the gas. The amplitude out of the gassed sample was reduced in comparison with the non-gassed sample.

The lock-in amplifier algorithms were able to retrieve the signal and phase of the sampled signal. The lock-in amplifier algorithms were not tested with a sampled signal containing high noise levels.

The performance of the lock-in amplifier algorithms under high noise level conditions is a task for future hardware development.

Chapter 6 Project Conclusions

6.1 Optimal Specifications

An inter-relationship existed between:

- ADC resolution.
- ADC sample frequency.
- Integration time.

ADC resolution: Good results were obtained from using a 6 bit ADC resolution. Higher bit resolutions produce marginally better results with no discernable improvement after 8 bit resolution. Actual implementation may depend on the cost of discrete ADC's at various resolutions.

ADC sample frequency: The microprocessor or the DSP will produce the ADC sample frequency as an on chip function

For a doubling for the sampling frequency:

- Noise immunity improved by 3 dB
- The average error[2] was halved

Therefore microprocessors and DSP's that have a higher clock speed will give better results however it is unknown if there is an upper limit to this improvement.

Integration time: An increase in integration time proved to increase the noise immunity. This was not fully tested because of the limitation of the host computer. However in minimal testing it was seen that an increase in integration time from 0.1 s to 1 s increased noise immunity by 10 dB. For applications were real time monitoring is not required increasing the integration time is advantageous.

Hardware should be chosen to optimise the integration function. It should be capable of 4 bit ADC (or better) and producing a sample frequency greater than 100 kHz. It should be noted that the dsPIC33F more than meets these requirements.

6.2 Opportunities for Further Study

The actual selection of hardware and the implementation are seen as the basis of a possible future project. Considerations should be given to dsPIC processes

Further study is required to identify the limit (if any) to improvement of the noise immunity by increasing the sampling frequency.

Further investigation of the increase in integration time would be considered profitable, this would only be feasible using a hardware implementation.

6.3 Conclusion Summary

The specifications determined by this project are within the specifications of current microprocessors and DSP's.

Hardware should be chosen to optimise the integration function. It should be capable of 6 bit ADC (or better) and producing a sample frequency greater than 100 kHz. It should be noted that the dsPIC33F more than meets these requirements.

There are further avenues of study available.

References

Aguirre, J., Medrano, N., Calvo, B. & Celma, S. (2011), 'Lock-in amplifier for portable sensing systems', Electronics Letters 47(21), 1172-1173.

Bengtsson, L. E. (2012), 'A microcontroller-based lock-in amplifier for sub-milliohm resistance measurements', Review of Scientific Instruments 83(7), 075103-075103-8.

Davies, R. & Meuli, G. (2010), Development of a digital lock-in amplifier for open-path light scattering measurement, in 'Industrial Electronics Applications (ISIEA), 2010 IEEE Symposium on', pp. 50-55.

Kim, E.-J., Park, H.-Y. & Kim, S. (2009), Low frequency clock synchronization technique for low signal to noise ratio (snr) signal recoery from noise environment, in 'Digital Signal Processing, 2009 16th International Conference on', pp. 1-4.

Li, G., Zhou, M., He, F. & Lin, L. (2011), 'A novel algorithm combining oversampling and digital lock-in amplifier of high speed and precision', Review of Scientific Instruments 82(9), 095106-095106-6.

Son, H.-H., Jung, I.-I., Hong, N.-P., Kim, D.-G. & Choi, Y. (2010), 'Signal detection technique utilising 'lock-in' architecture using $2\omega_c$ harmonic frequency for portable sensors', Electronics Letters 46(13), 891-892.

Vogelgesang, R 2004, Lock-in Amplifier Theory, viewed May 9, <http://traktoria.org/files/sonar/signal_processing/analog/LockIn.pdf>

Wenn, D 2007, 'Implementing Digital Lock-In Amplifier Using the dsPIC DSC', *Microchip Technology Inc. AN1115*, viewed 22 April 2013, <http://ww1.microchip.com/downloads/en/AppNotes/01115A.pdf>.

2013. [ONLINE] Available at: <http://www.thinksrs.com/downloads/PDFs/ApplicationNotes/AboutLIAs.pdf>, viewed 15 October 2013.

Appendix A

Project Specification

ENG 4111/2 Research Project

Project Specification

For: **Robert Skillington**

Topic: DSP-Based Lock-in Amplifier

Supervisors: John Leis

Sponsorship: Faculty of Health, Engineering & Sciences

Project Aim: To develop hardware and software to implement a lock-in amplifier for measurement systems. Using a Digital Signal Processor (DSP) or even a low-cost microcontroller. A part of the research aspect is to determine the suitability of certain processor architectures for this task.

Program: (Issue B, 26th March 2013)

1. Research the design and use of the Lock-in Amplifier, both analogue and digital.
2. Determine the set of algorithms to be implemented for the lock-in, including the reference signal multiplication, low pass filtering, and the separate phase-locked loop.
3. Evaluate the performance of the algorithms as implemented, and show the signal recovery performance for various parameter settings at different SNR levels.
4. Evaluate the performance in MATLAB using sampled real-world signals.
5. Investigate suitable processor architectures and development systems for an embedded lock-in amplifier.

As time and resources permit:

1. Design the hardware and implement the embedded lock-in amplifier.
2. Test its performance under various conditions.
3. Augment with a Phase-Locked Loop (PLL) so as to be able to use an external reference signal.

Agreed:

Student Name: Robert Skillington
Date:

Supervisor Name: John Leis
Date:

Examiner/Co-Examiner:
Date:

Appendix B

MATLAB program listing for LOCK-IN AMPLIFIER ADC simulation

ProjectDSPBasedLIA_2013_Final.m

```
% Robert Skillington, Project: DSP based Lock-in Amplifier 2013
% Lock-in Amplifier algorithm using Cosine or Square waveform for Reference signal
% This code uses logarithmic spacing for the added noise or linear spacing
% Simulates ADC bit resolution from 2 to 16 bit

clc;
clear all;
close all;

%% Signal parameters
tic
% sample frequencies used, 8,16,32,40,64,80,128 kHz
fs =80000;                      % Sampling Frequency (samples/second) >8000
Ar = 1;                         % Reference Amplitude
Ai = 1.8;                       % Input Amplitude
t = 0.1;                        % Integration Time in seconds
n = [0:round(t*fs)-1]'/fs;      % Time axis
fo = 2000;                      % Frequency of sinusoid
errIncr = 0.02;                 % percent error above input amplitude

%% Experimental signal

VsigCos = Ai*cos((n*2*pi*fo)-(pi/10)) ;% V signal Sine waveform equation
% to add the phase difference through the experiment

%% Cosine Reference signals & %% Square Reference signals Note: Ao(k) also
%% has to be changed for square reference.

% VrefSin = Ar*square(n*2*pi*fo);      % V reference Sine waveform equation
% VrefCos = Ar*square((n*2*pi*fo)+(pi/2)); % V reference Cosine waveform
% equation

VrefCos = Ar*cos(n*2*pi*fo);          % V reference Sine waveform equation
VrefSin = Ar*cos((n*2*pi*fo)-(pi/2));% V reference Cosine waveform equation

%% Plots of Experimental signal and reference signals and their multiplication

Vctest = VsigCos.*VrefCos; % Vctest to plot signal with no noise
Vstest = VsigCos.*VrefSin; % Vstest to plot signal with no noise

figure('units','normalized','outerposition',[0 0 1 1]); %Full screen Figure
figure(1)
subplot(3,1,1)
plot(n(1:ceil(1/fo*5*fs)),VsigCos(1:ceil(1/fo*5*fs)),'-o') % gives 5 cycles of the
waveform
title('Cosine Experimental Signal');
title(sprintf('Cosine Experimental Signal at %g Hz at a sample frequency of %g
Hz',fo,fs),'FontSize',14);
xlabel('Time (sec)','FontSize',12);
ylabel('Amplitude (V)','FontSize',14);
grid on

subplot(3,1,2)
plot(n(1:ceil(1/fo*5*fs)),VrefCos(1:ceil(1/fo*5*fs)),'-o')
title(sprintf('Cosine reference Signal at %g Hz',fo),'FontSize',14);
xlabel('Time (sec)','FontSize',12);
ylabel('Amplitude (V)','FontSize',14);
grid on
```

```
subplot(3,1,3)
plot(n(1:ceil(1/fo*5*fs)),Vctest(1:ceil(1/fo*5*fs)),'-o')
title('Cosine Experimental Signal multiplied by Cosine reference
Signal','FontSize',14);
xlabel('Time (sec)','FontSize',12);
ylabel('Amplitude (V)','FontSize',14);
grid on

figure('units','normalized','outerposition',[0 0 1 1]); % Full screen Figure
figure(2)
subplot(3,1,1)
plot(n(1:ceil(1/fo*5*fs)),VsigCos(1:ceil(1/fo*5*fs)),'-o')
title(sprintf('Cosine Experimental Signal at %g Hz at a sample frequency of %g
Hz',fo,fs),'FontSize',14);
xlabel('Time (sec)','FontSize',12);
ylabel('Amplitude (V)','FontSize',14);
grid on

subplot(3,1,2)
plot(n(1:ceil(1/fo*5*fs)),VrefSin(1:ceil(1/fo*5*fs)),'-o')
title(sprintf('Sine reference Signal at %g Hz',fo),'FontSize',14);
xlabel('Time (sec)','FontSize',12);
ylabel('Amplitude (V)','FontSize',14);
grid on

subplot(3,1,3)
plot(n(1:ceil(1/fo*5*fs)),Vstest(1:ceil(1/fo*5*fs)),'-o')
title('Cosine Experimental Signal multiplied by Sine reference
Signal','FontSize',14);
xlabel('Time (sec)','FontSize',12);
ylabel('Amplitude (V)','FontSize',14);
grid on

%% Main LIA with ADC resolution simulation

Aolimstore=[];   % Stores Ao max output below set threshold
Aerrlimstore=[]; % SNR at 2%

% set to 10,100,1000,5000 (time taken will increase start low)
kmax = 1000; % amount of random samples being averaged

adc = 2:16; % ADC resolution

for ADC = adc;

    aveSNRsin = [];     % initialising average of SNR   % aveSNRsqur = [];
    aveAerrsin = [];    % initialising average of error % aveAerrsqur = [];
    aveAo= [];          % initialising average of error % aveAosqur = [];

    stepn = -1; %stepn = 0.1; use with linspace
    %changes the range of Gaussian noise from the experiment
    x = logspace(stepn,2,100); % linspace(stepn,100,10/stepn);

    for a = x; %for a = for a = stepn;%  % noise added from 0.1 to 100

        for k = 1:kmax;

            %% Noise addition

            randn('seed', k+45);% sets the random starting at 45(no reason)

            Expnoise = a.*randn(length(n), 1); % Experimental noise

            npwr(k) = sum(Expnoise.^2);  % noise power

            sigpwr(k) = sum(VsigCos.^2); % signal power

            %% Input signal (Signal  + Noise)

            VsigCosandNoise = VsigCos + Expnoise;% adds noise to the signal

            %% Quantisation
```

```
            Mul = (2^15)/max(abs(VsigCosandNoise));  % scaling multiplier

            VsigCos16 = int16(VsigCosandNoise.*Mul); % convert to a signed 16 bit
intiger

            VsigCosQ16 = idivide(VsigCos16,2^(16-ADC), 'floor').*2^(16-ADC); %
Quantises

            VsigCosandNoiseQ = double(VsigCosQ16)./Mul; % Convert back to a double

            %% Noisy signal x reference signals

            Vs = VsigCosandNoiseQ.*VrefSin; % Vs is the multiplication of noisy input
signal and Sine reference
            Vc = VsigCosandNoiseQ.*VrefCos; % Vs is the multiplication of noisy input
signal and Cosine reference

            %% Filter for Vs and Vc to form I and Q (X and Y) channel
            %% output

            Ivs = mean(Vs); % mean of I
            Qvc = mean(Vc); % mean of Q

            %% Finds the Amplitude of output signal
            % For square Ref wave Ao = (pi/4)*2*sqrt((Ivs^2)+(Qvc^2));
            Ao(k) = 2*sqrt((Ivs^2)+(Qvc^2));

            %% Finds the Phase of signal

            Phase =  atan2(Ivs,Qvc);% OR atan(Qvc./Ivs)

            %% SNR

            SNR(k) = 10*log10((sigpwr)/(npwr)); % SNR of the output

            Aerr(k) = Ao(1,k)-Ai; % Error between the input and the output

        end

        aveSNRsin(end+1) = mean(SNR);       % Average of  SNR's for kmax sets of
noise
        aveAerrsin(end+1) = mean(Aerr.^2);  % Average of  error for kmax sets of
noise
        aveAo(end+1) = mean(Ao);            % Average of  Amplitude out for kmax sets
of noise

    end

    %% Ployfit straight line approximation for Average error squared
    % creates polynomial fit to the Average error squared values
    [p,s] = polyfit(aveSNRsin(1,2:end),log10(aveAerrsin(1,2:end)),1);
    polcap = polyval(p,aveSNRsin(1,2:end));
    Polcap10 = 10.^(polcap);

    %% Calculates the SNR at the 2% error set by 'errIncr'
    l=1;
    while aveAo(l)<= (Ai+(Ai*errIncr)); % acceptable output threshold
        Aolim = aveSNRsin(1,l);
        Aerrlim = aveAerrsin(1,l);
        l = l+1;
    end
    Aerrlimstore(1,end+1) = Aerrlim;
    Aolimstore(1,end+1) = Aolim; % Stores the SNR value at the acceptable output

    %% Variables stored for each ADC resolution

    SNRavebit(:,ADC) = aveSNRsin(1,2:end); % stores SNR
    Aerravebit(:,ADC) = aveAerrsin(1,2:end); % stores average error
    Aoavebit(:,ADC) = aveAo(1,2:end); % stores Amplitude out
    polcapbit(:,ADC) = Polcap10; % stores Polyfit straight line
```

```
    MeanAerravebit = mean(Aerravebit); % average of Average error^2

    fprintf('SNR = %2.4f dB\tat %2.0fbit ADC\tat Sample Frequency: %2.0f   Ave error:
%2.5f  k = %2.0f Time= %2.2fsec\n', Aolim, ADC, fs, MeanAerravebit(ADC),kmax,t);

end

%% Chooses specific SNRs for each ADC resolution

SNRdB = [20 15 10 5 0 -5 -10 -15 -20 -25 -30];

for dB = 1:length(SNRdB);
    for v = adc;
        nsnr=1;
        while length(SNRavebit)>=nsnr && SNRavebit(nsnr,v) >= SNRdB(1,dB);
            bitavesnr(dB,v) = nsnr+1;
            nsnr=nsnr+1;
            Avegerrstor(dB,v) = polcapbit(nsnr+1,v);
        end
    end
end
% Plot of Ave error^2 for SNRdB for ADC resolution range.
figure('units','normalized','outerposition',[0 0 1 1]); % Full screen Figure
semilogy(adc,flipud(Avegerrstor(:,adc)),'-o','LineWidth',2);% or use plot
title(sprintf('%g - %g bit ADC at Sample Frequency: %gHz', adc(1,1),adc(1,end),
fs),'FontSize',14);
ylabel('Average error ^2','FontSize',14);
xlabel('ADC bit Resolution','FontSize',14);
legend(fliplr({'20 dB','15 dB','10 dB','5 dB','0 dB','-5 dB','-10 dB','-15 dB','-20
dB','-25 dB','-30 dB'}))
xlim([1 adc(1,end)+1]);
ylim([10^-6 10^1]);

toc % show time taken
```

Program to Plot variables created by 'ProjectDSPBasedLIA_2013_Final.m'

ProjectDSPBasedLIA_2013_Final_Plots.m

```
%% Plots for different fs and bit resolutions to be used with
%% ProjectDSPBasedLIA_2013_Final.m

clc
close all

for ADC = adc(1,1):adc(end);

    figure('units','normalized','outerposition',[0 0 1 1]); % Full screen Figure
    subplot(2,2,1);
    semilogy(SNRavebit(:,ADC),Aerravebit(:,ADC),'-o',
SNRavebit(:,ADC),polcapbit(:,ADC),'r');%
    title(sprintf('Average Error ^2 for %g bit ADC Resolution on a semilog Y
scale',ADC),'FontSize',14);
    xlabel('SNR dB','FontSize',14);
    ylabel('Average error ^2','FontSize',14);
    ylim([10^-6 10^1]);
    xlim([-40 30]);
    grid on;

    %     figure('units','normalized','outerposition',[0 0 1 1]); % Full screen
Figure
    subplot(2,2,2);
    plot(SNRavebit(:,ADC),Aerravebit(:,ADC),'o');
    title(sprintf('Average Error ^2 for %g bit ADC Resolution',ADC),'FontSize',14);
    xlabel('SNR dB','FontSize',14);
    ylabel('Average error ^2','FontSize',14);
    %     ylim([0 max(aveAerrsin)]);
    xlim([-40 30]);
    grid on;

    %     figure('units','normalized','outerposition',[0 0 1 1]); % Full screen
Figure
    subplot(2,2,3);
    plot(n(1:500),VsigCosandNoiseQ(1:500),'-o');
    %     ylim([-5000 5000]);
    title(sprintf('Quantized Signal and Noise for %g bit ADC Resolution, %g
quantization levels',ADC,2^ADC),'FontSize',14);
    xlabel('time (sec)','FontSize',14);
    %     xlim([0 0.003]);
    %     ylabel('');
    grid on;

    l=1;
    while Aoavebit(l,ADC)<= (Ai+(Ai*errIncr)); % acceptable output threshold
        Aolim = SNRavebit(l,ADC);
        Aerrlim = Aerravebit(l,ADC);
        l = l+1;
    end
    Aerrlimstore(1,end+1) = Aerrlim;
    Aolimstore(1,end+1) = Aolim; % Stores the SNR value at the acceptable output

    %     figure('units','normalized','outerposition',[0 0 1 1]); % Full screen
Figure
    subplot(2,2,4);
    %plot(aveSNRsin(1,2:end),aveAo(1,2:end),'-',[min(aveSNRsin)
max(aveSNRsin)],[(Ai+(Ai*errIncr)),(Ai+(Ai*errIncr))],'--r','LineWidth', 2);
    plot(aveSNRsin(1,2:end),aveAo(1,2:end),'-',[Aolim Aolim],[min(aveAo)
max(aveAo)],'r',[min(aveSNRsin)
max(aveSNRsin)],[(Ai+(Ai*errIncr)),(Ai+(Ai*errIncr))],'--r','LineWidth', 2);
    title(sprintf('SNR = %gdB, %gbit ADC at Sample Frequency: %gHz',Aolim, ADC,
fs),'FontSize',14);
    xlabel('Signal to Noise Ratio (dB)','FontSize',14);
    ylabel('Amplitude out (V)','FontSize',14);
    legend('Amplitude out','dB at 2% error','2% error')% legend('Amplitude out','2%
error')%
    ylim([1.75 max(aveAo)]);
    %     xlim([-40 30]);
    grid on;
```

```
    fprintf('SNR = %2.4f dB\tat %2.0fbit ADC\tat Sample Frequency: %2.0f   Ave error:
%2.5f  k = %2.0f Time= %2.2fsec\n', Aolim, ADC, fs, MeanAerravebit(ADC),kmax,t);

end

SNRdB = [20 15 10 5 0 -5 -10 -15 -20 -25 -30];

for dB = 1:length(SNRdB);
    for v = adc;
        nsnr=1;
        while length(SNRavebit)>=nsnr && SNRavebit(nsnr,v) >= SNRdB(1,dB);
            bitavesnr(dB,v) = nsnr+1;
            nsnr=nsnr+1;
            Avegerrstor(dB,v) = polcapbit(nsnr+1,v);
        end
    end
end

figure('units','normalized','outerposition',[0 0 1 1]); % Full screen Figure
semilogy(adc,flipud(Avegerrstor(:,adc)),'-o','LineWidth',2);% or use plot
title(sprintf('%g - %g bit ADC at Sample Frequency: %gHz', adc(1,1),adc(1,end),
fs),'FontSize',14);
ylabel('Average error ^2','FontSize',14);
xlabel('ADC bit Resolution','FontSize',14);
legend(fliplr({'20 dB','15 dB','10 dB','5 dB','0 dB','-5 dB','-10 dB','-15 dB','-20
dB','-25 dB','-30 dB'}))
xlim([1 adc(1,end)+1]);
ylim([10^-6 10^1]);
```

Lock-in amplifier algorithms with no added noise or ADC simulation.

ProjectDSPBasedLIA_2013.m

```
% Robert Skillington Project DSP based Lock-in Amplifier 2013
% LIA algorithm using Square OR sine waveform for Reference signal

clc; clear all; close all;%

%%
tic

fs = 80000;                   % sampling frequency (samples/second) >4000
Ar = 1;                       % Reference Amplitude
Ai = 1.8;                     % Input Amplitude
n = [0:round(0.1*fs)-1]'/fs;  % Time axis
fo = 2000;                    % Frequency of sinusoid

%% Experement signal
 VsigCos = Ai*cos((n*2*pi*fo)-(pi/10)) ;%   % V signal Sine waveform equation to add
the phase difference through the experiment


%% Cosine Reference signals                  % Square Reference signals

VrefCos = Ar*cos(n*2*pi*fo); % Ar*square(n*2*pi*fo);    % V reference Sine waveform
equation
VrefSin = Ar*cos((n*2*pi*fo)-(pi/2));% Ar*square((n*2*pi*fo)+(pi/2));  % V reference
Cosine waveform equation


%% Vs and Vc (VsigSinandNoise x V reference Sine) and (VsigSinandNoise x V reference
Cosine)

Vs = VsigCos.*VrefSin; % Vs is the multiplication of noisy input signal and Sine
reference
Vc = VsigCos.*VrefCos; % Vs is the multiplication of noisy input signal and Cosine
reference

%% Filter for Vs and Vc to form I and Q (X and Y)

Ivs = mean(Vs); % mean of
Qvc = mean(Vc); % mean of


%% Finds the magnitude of signal

Ao = 2*sqrt((Ivs^2)+(Qvc^2));   % Ao = (pi/4)*2*sqrt((Ivs^2)+(Qvc^2));% for square
wave %LIA output Amplitude

fprintf('Ai: %1.10f \n',Ai)
fprintf('Ao: %1.10f \n',Ao)

%% Finds the Phase of signal

Phase = atan2(Ivs,Qvc); % atan(Qvc./Ivs);   % Phase difference from signal and
reference

fprintf('Phase: %1.5g radians \n',Phase);
```

Lock-in amplifier used on a real sampled signal.
LIA_Sampled_Gas_Signal.m

```
% Sampled signal Test

% Robert Skillington Project DSP based Lock-in Amplifier 2013
% LIA algorithm using square and sine waveform for Reference signal
% This code shows both an air and gas sample
% The sine and cosine voltage was 2.5V peak to peak
% The Cosine wave had a DC offset of 2.5V

clc; clear all; close all;%

sampl = 10000 ;

[DataMat NumChans SampleRate ScanRate NumScans Comment] = AdlinkReadFile('air.bin',
sampl);
[DataMatg NumChansg SampleRateg ScanRateg NumScansg Commentg] =
AdlinkReadFile('gas.bin', sampl);

samptime = sampl/ScanRate;
fprintf('Sample Time: %1.5g seconds \n',samptime);
xsigD = DataMat(:,1); % V signal Sine waveform equation to add the phase difference
through the experement
xCosD = DataMat(:,2); % V referance Cosine waveform equation   %  VrefSin =
Ar*square(n*wo)
xSinD = DataMat(:,3); % V referance Sine waveform equation %  VrefCos =
Ar*square((n*wo)+(pi/2))

figure(1)
subplot(3,1,1)
plot(xsigD(1:100),'-o')
title('Signal','FontSize',14);

grid on

subplot(3,1,2)
plot(xCosD(1:100),'-o')
title('Cosine reference','FontSize',14);
grid on

subplot(3,1,3)
plot(xSinD(1:100),'-o')
title('Sine reference','FontSize',14);
grid on

xsig = xsigD -mean(xsigD); % Normalises
xCos = xCosD -mean(xCosD);
xSin = xSinD -mean(xSinD);

figure(2)
subplot(3,1,1)
plot(xsig(1:100),'-o')
title('Normalised Signal','FontSize',14);
grid on

subplot(3,1,2)
plot(xCos(1:100),'-o')
title('Normalised Cosine reference','FontSize',14);
grid on

subplot(3,1,3)
plot(xSin(1:100),'-o')
title('Normalised Sine reference','FontSize',14);
grid on

Vc = xsig.*xCos; % Vs is the multiplication of noisy input signal and Cosine
reference
Vs = xsig.*xSin; % Vc is the multiplication of noisy input signal and sine reference

figure(3)
subplot(2,1,1)
plot(Vc(1:100),'-o')
title('X','FontSize',14);
grid on
```

```
subplot(2,1,2)
plot(Vs(1:100),'-o')
title('Y','FontSize',14);
grid on

%% Filter for Vs and Vc to form I and Q (X and Y)

Is = mean(Vs); % mean of
Qc = mean(Vc); % mean of


%% Finds the magnitude of signal

Ao = 2*sqrt((Is^2)+(Qc^2));   % LIA output Amplitude

fprintf('Ao: %1.2f \n',Ao)

%% Finds the Phase of signal

Phase = atan2(Qc,Is);% atan(Qc./Is);        % Phase difference from signal and
reference

fprintf('Phase: %1.5g radians \n',Phase);

%%
%% GASED Sample


xsigDg = DataMatg(:,1); % V signal Sine waveform equation to add the phase difference
through the experement
xCosDg = DataMatg(:,2); % V referance Cosine waveform equation   %  VrefSin =
Ar*square(n*wo)
xSinDg = DataMatg(:,3); % V referance Sine waveform equation %  VrefCos =
Ar*square((n*wo)+(pi/2))

figure(4)
subplot(3,1,1)
plot(xsigDg(1:100),'-o')
title('Signal with Gas','FontSize',14);
grid on

subplot(3,1,2)
plot(xCosDg(1:100),'-o')
title('Cosine reference','FontSize',14);
grid on

subplot(3,1,3)
plot(xSinDg(1:100),'-o')
title('Sine reference','FontSize',14);
grid on

xsigg = xsigDg -mean(xsigDg); % Normalises
xCosg = xCosDg -mean(xCosDg);
xSing = xSinDg -mean(xSinDg);

figure(5)
subplot(3,1,1)
plot(xsigg(1:100),'-o')
title('Normalised Signal with Gas','FontSize',14);
grid on

subplot(3,1,2)
plot(xCosg(1:100),'-o')
title('Normalised Cosine reference','FontSize',14);
grid on

subplot(3,1,3)
plot(xSing(1:100),'-o')
title('Normalised Sine reference','FontSize',14);
grid on

Vcg = xsigg.*xCosg; % Vs is the multiplication of noisy input signal and Cosine
reference
Vsg = xsigg.*xSing; % Vc is the multiplication of noisy input signal and sine
reference
```

```
figure(6)
subplot(2,1,1)
plot(Vcg(1:100),'-o')
title('X channel with Gas','FontSize',14);
grid on

subplot(2,1,2)
plot(Vsg(1:100),'-o')
title('Y channel with Gas','FontSize',14);
grid on

%% Filter for Vs and Vc to form I and Q (X and Y)

Isg = mean(Vsg); % mean of
Qcg = mean(Vcg); % mean of


%% Finds the magnitude of signal

Aog = 2*sqrt((Isg^2)+(Qcg^2));   % LIA output Amplitude

fprintf('Ao gas: %1.2f \n',Aog)

%% Finds the Phase of signal

Phaseg = atan2(Qc,Is);        % Phase difference from signal and reference pi/2-
2pi/25=1.319

fprintf('Phase gas: %1.5g radians \n',Phaseg);

per = Aog/Ao;
fprintf('Per change: %1.5g %%\n',per*100);
```

Appendix C

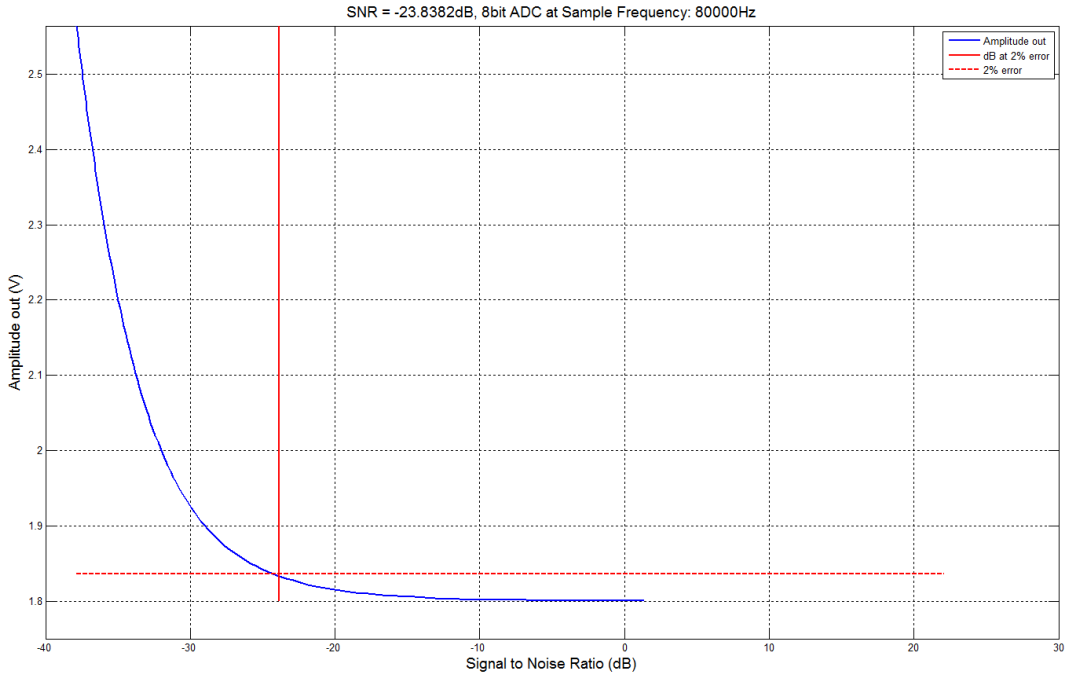

Figure C-1: Amplitude out using linear spacing for a range from 0.1 to 100

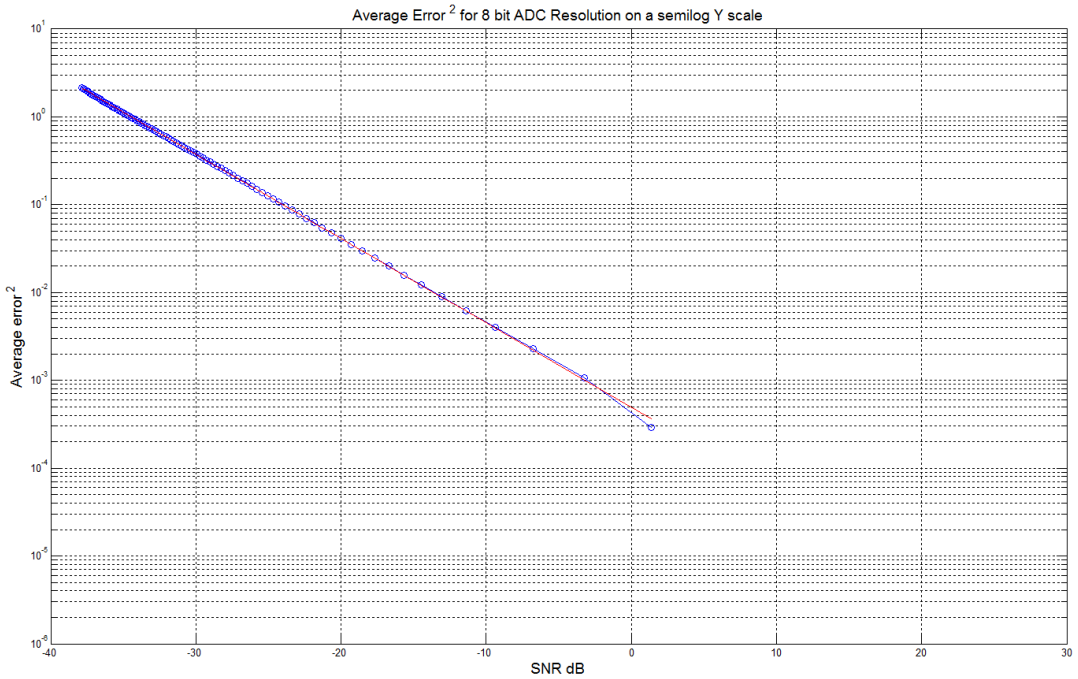

Figure C-2: Average error^2 using linear spacing on a logarithmic y axis

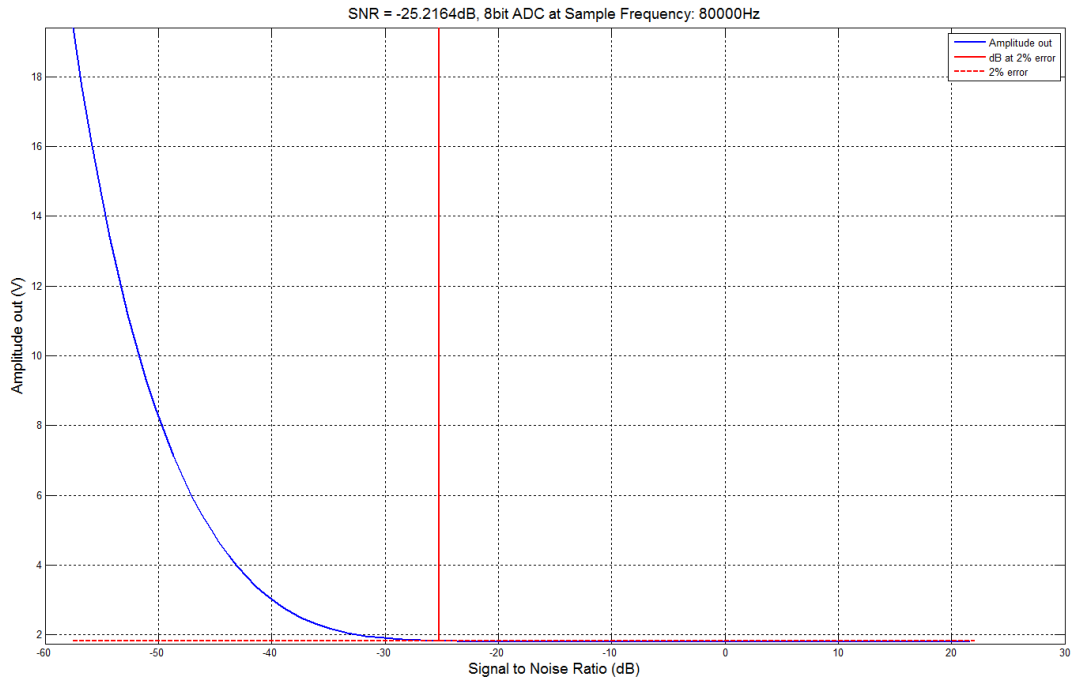

Figure C-3: Plot of amplitude out using a log spacing

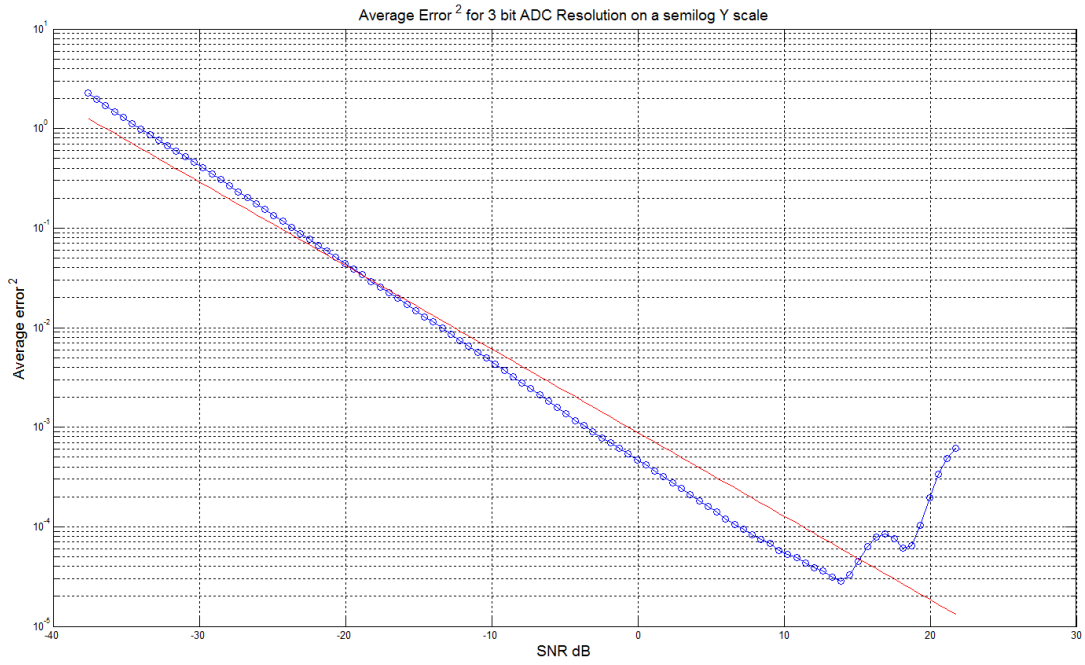

Figure C-4: Average error^2 with a straight line approximation for 3 bit ADC at sample frequency of 80 kHz

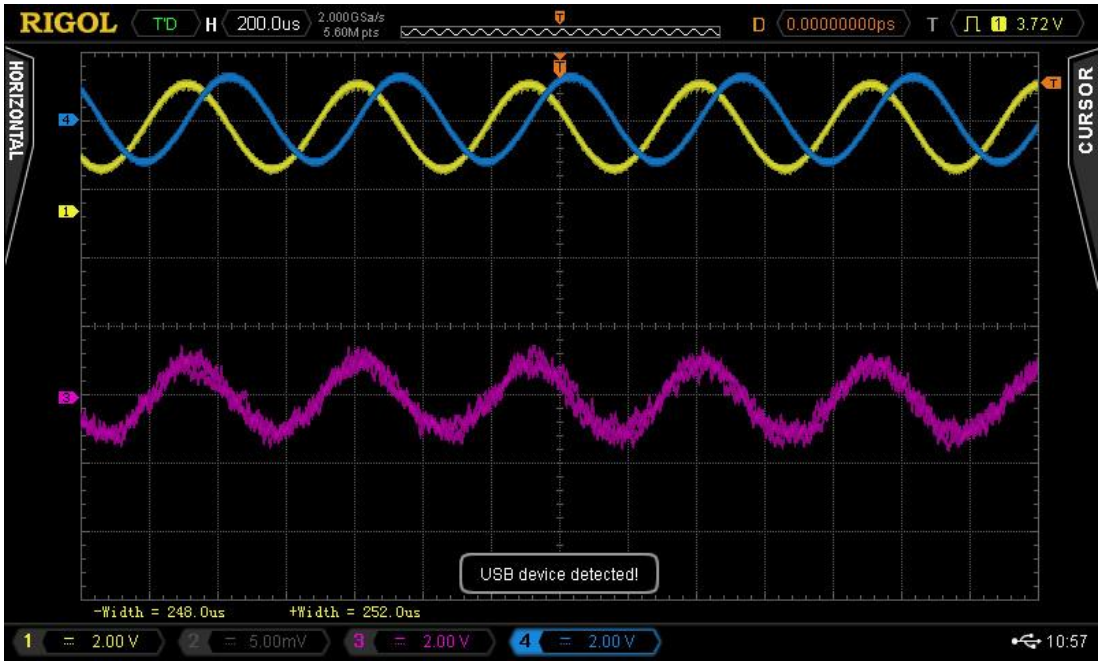

Figure C-5: An oscilloscope screen shot of a sampled gas experiment.

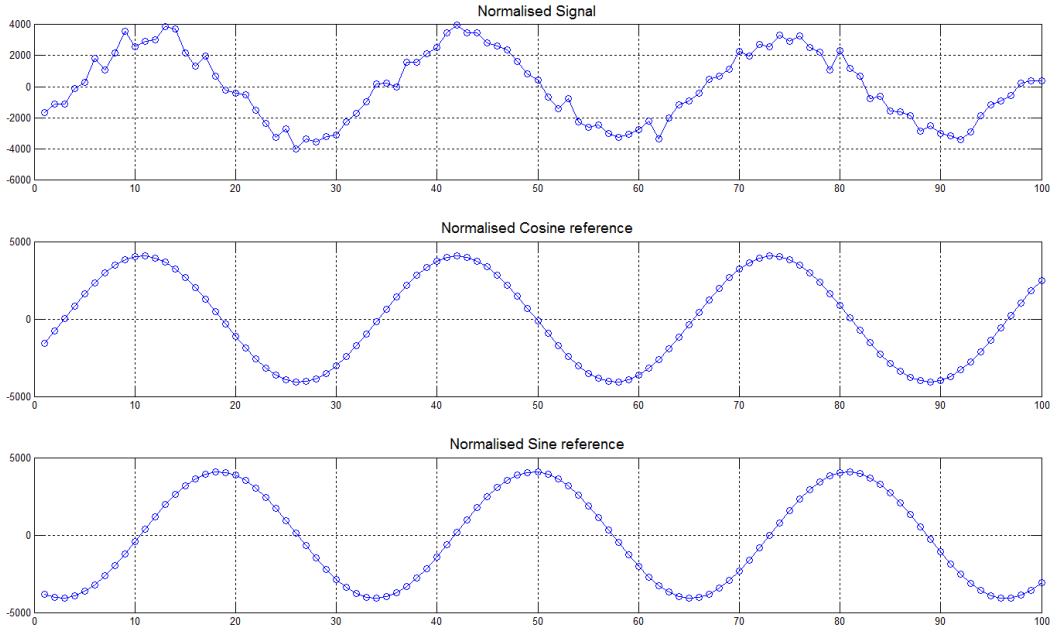

Figure C-6: Samples signal without gas present.