University of Southern Queensland

Faculty of Engineering & Surveying

Design of a Wireless Acquisition System for a Digital Stethoscope

A dissertation submitted by

Justin Miller

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Electrical and Electronic Engineering)

Submitted: October, 2010

Abstract

The auscultation of the heart with a stethoscope is one of the most common methods employed by physicians to diagnose cardiovascular and respiratory illnesses. Phonocardiography refers to the technique of acquiring and recording of heart sound signals. The emergence of teleheath and electronic stethoscope technology has opened new opportunities for rural and regional medical services including the remote screening of heart murmurs.

This dissertation investigates the design and implementation of a wireless data acquisition module to capture auscultation sounds from an electronic stethoscope, and sets the foundation for further research into the area of remote auscultation diagnosis and non-invasive techniques for diagnosing abnormalities.

Methods to detect activity in the signal are evaluated for the suppression of ambient noise and adaptive gain control. Several well known noise reduction techniques for signals acquired from a single source are studied and evaluated. A PI controller is developed to control the gain of the input stage to account for attenuation of the heart and respiratory sounds caused by volume effects (i.e. absorption) of the human body.

The acquisition module is controlled by a 16bit dsPic digital signal controller which samples auscultation signals from a digital stethoscope and streams the auscultation signals to the host over a wireless Bluetooth connection. The signal and power supply is isolated for compliance with the international standards for medical devices (IEC 60601-1). A Windows application incorporating a Bluetooth client was developed to receive incoming data packets from the acquisition module and display the signal graphically.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, 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 Engineering and Surveying 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.

Prof F Bullen

Dean

Faculty of Engineering and Surveying

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.

Justin Miller

0050027832

Signature

Date

Acknowledgments

I would like to express my sincere gratitude to Associate Professor John Leis for his guidance and encouragement throughout the course of this project. Due thanks must go to the Faculty of Engineering and Surveying at the University of Southern Queensland for the resourcing of this project.

Last but not least, I would like to thank my wife, Soo Kooi, and my children, Joshua, Theodore and Elizabeth. This project would be impossible without their patience and support.

Justin Miller

University of Southern Queensland

October 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The auscultation of the heart with a stethoscope is one of the most common methods employed by physicians to diagnose cardiovascular and respiratory illnesses. With the growing acceptance of teleheath (remote diagnosis) and electronic stethoscope technologies, the acquisition and graphical display of heart and lung sounds may prove to be beneficial for rural and regional medical services.

The benefits of a wireless stethoscope are numerous: Heart and lung sounds can be transferred to a PC, laptop or mobile phone further further analysis without cables. The patient and practitioner are free to move without hindrance and are safe from potentially fatal voltage sources that may be present on a device that is not properly isolated.

Phonocardiography provides a graphical visualisation of auscultation signals, allowing clinical observation of heart sounds that are characterised by frequencies outside the normal range of human hearing (Tilkian and Conover, 2001). The time-frequency analysis of auscultation signals has been proven to be a powerful diagnosis tool for the segmentation of heart and lung sound components and identification of abnormal heart sounds including systolic murmurs and ventricular septal defects.

The key objective of this project is to capture heart sounds from an electronic stethoscope and transmit the data to a desktop PC for display.

The purpose of this report is to review background research in the field of signal processing when applied to phonocardiographic records. This includes: (i) An understanding of the sounds generated by the heart, (ii) methods of controlling the signal gain before it is sampled, (iii) an evaluation of common de-noising techniques and (iv) an evaluation of time-frequency transforms for the display of phonocardiographic signals in the time-frequency domain.

1.1 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 reviews past and current research in the field of heart sounds and data acquisition.

Chapter 3 investigates the signal processing aspects of the project.

Chapter 4 discusses the hardware and firmware design of the wireless acquisition module.

Chapter 5 discusses the software design of the host application.

Chapter 6 discusses the test and implementation stage of the project.

Chapter 7 concludes the dissertation and suggests further work in the area of remote auscultation diagnosis.

Chapter 2

Literature Review

2.1 Introduction

The auscultation of the heart is one of the most common methods employed by physicians to diagnose cardiovascular and respiratory illnesses. The most common auscultative tool is the stethoscope. An experienced physician can diagnose a wide range of cardiovascular abnormalities including mitral stenosis and systolic murmurs, however many abnormalities are commonly missed due to an inability to apply selective listening to the various components of the heart beat, or a natural inability to detect frequencies outside the normal range of human hearing. Segmentation of the various heart sound components, including components that indicate an abnormality, can be difficult to achieve if they occur simultaneously or close apart (Tilkian and Conover, 2001).

Phonocardiography provides a graphical visualisation of auscultatory signals, allowing clinical observation of heart sounds characterised by frequencies outside the normal range of human hearing. With the application of a Short-Time Fourier Transform or Continuous Wavelet Transform, heart sounds are represented in the time-frequency domain, hence allowing heart sound components to be readily identified. As such, the phonocardiogram is not only a useful diagnosis tool for the experience clinician, but also a valuable learning tool for trainee medical staff (ibid).

The phonocardiogram also facilitates a screening process to rule out innocent murmurs before referring the patient to a cardiologist for expensive echocardiography.

This review of literature will cover the foundations of bioacustics pertaining to the cardiovascular system, the capture of auscultative signals in a noisy environment and evaluate methods of transforming heart sounds into the time-frequency domain for analysis.

2.2 Acoustic Properties of the Heart

2.2.1 Cardiac Cycle

The cardiac cycle can be defined as as the synchronized activity of the atria and the ventricles. During the atrail and ventricular diastole: (i) Venous (deoxygenated) blood enters the right atrium through the superior and inferior venae cavae.(ii) Blood flows into the right ventricle through the tricupsid valve. (iii) Arterial (oxygenated) blood flows from the lung into the left atrium. (iv) The left ventricle is filled with the atrerial blood through the mitral blood (Tilkian and Conover, 2001).

During the atrial systole phase, the atria begins to contract towards the end of the ventricular diastole. During the ventricular systole phase: (i) Venous blood moves through the pulmonary artery from the right ventricle to the lungs for oxidation. (ii) Arterial blood passes through the aorta from the left ventricle to the circulatory system (ibid).

The human heart consists of four valves to ensure that blood flows in only one direction through the circulatory system. The mitral and tricuspid valves, commonly referred to as the atrioventricular valves, guard the entrance from the atria to the ventricles. The semilunar valves (aortic and pulmonic valves) prevent blood from flowing back into the ventricles from the aorta and pulmonary arteries (ibid).

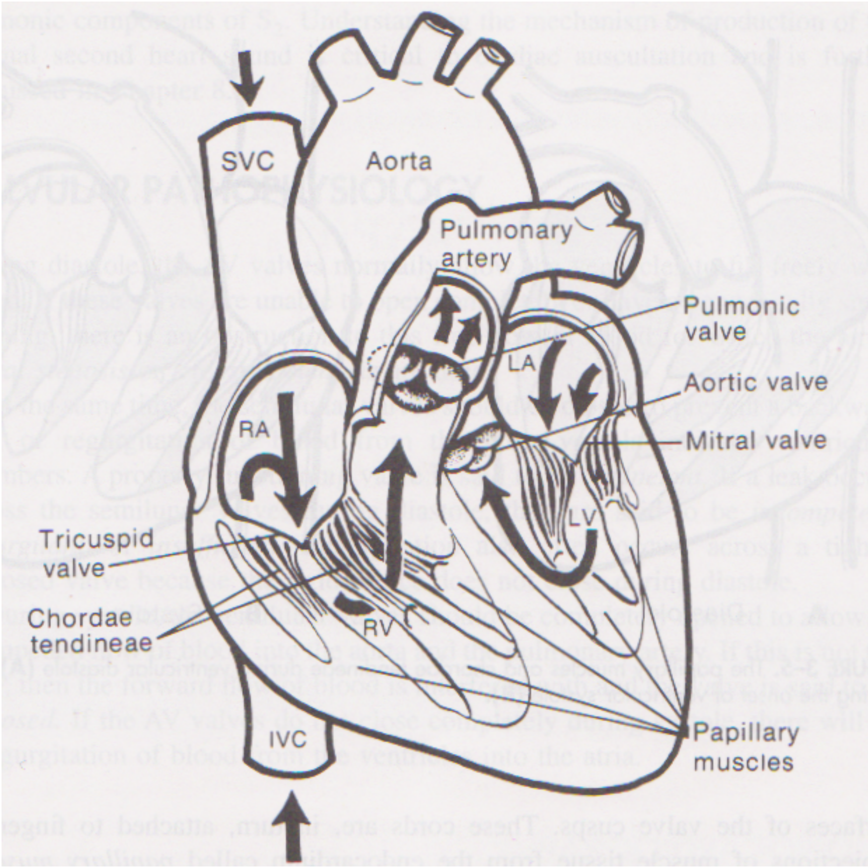


Figure 2.1: The blood flow through the four valves of the heart. Source: Tilkian and Conover (2001)

2.2.2 Heart Sounds

The first heart sound (S1) is caused by the closure of the atrioventricular valves - first the mitral valve followed shortly by the tricuspid valve. The closure of the aortic valve closure, closely followed by the pulmonary valve closure, causes the second heart sound (S2). The first and second heart sounds occur within a frequency range of 20Hz to 175Hz (Tilkian and Conover 2001). Rangayyan and Lehner (1987) however discovered that S1 contained peaks in low frequency range (10-50Hz) and and medium frequency range (50-140Hz), whilst S2 was found to contain peaks in a lower frequency range (10 to 80Hz), medium-frequency range (80-200Hz) and high-frequency range (220-400Hz).

The third (S3) and fourth (S4) heart sounds are the result of passive ventricular filling (early diastole) and active ventricular filling sound (late diastole) respectively. The third and fourth heart sounds occur between 20Hz and 70Hz. The presence of S3

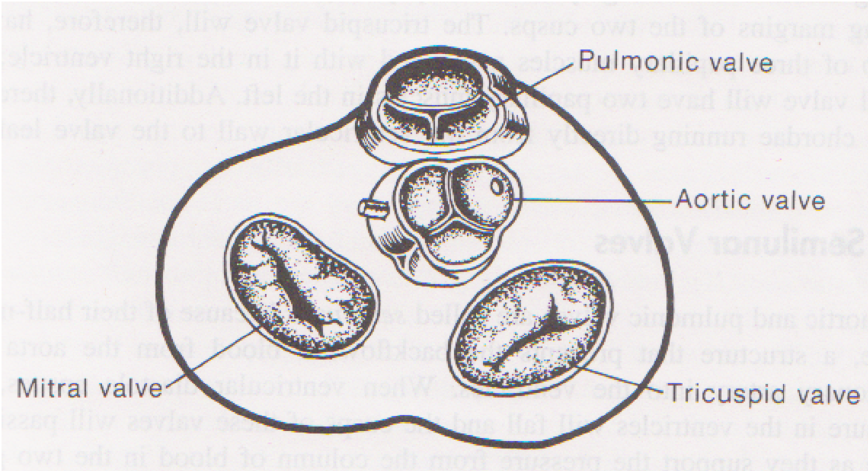


Figure 2.2: The four valves of the heart. Source: Tilkian and Conover (2001)

and S4 may suggest heart abnormalities, and therefore should be examined carefully (Tilkian and Conover 2001).

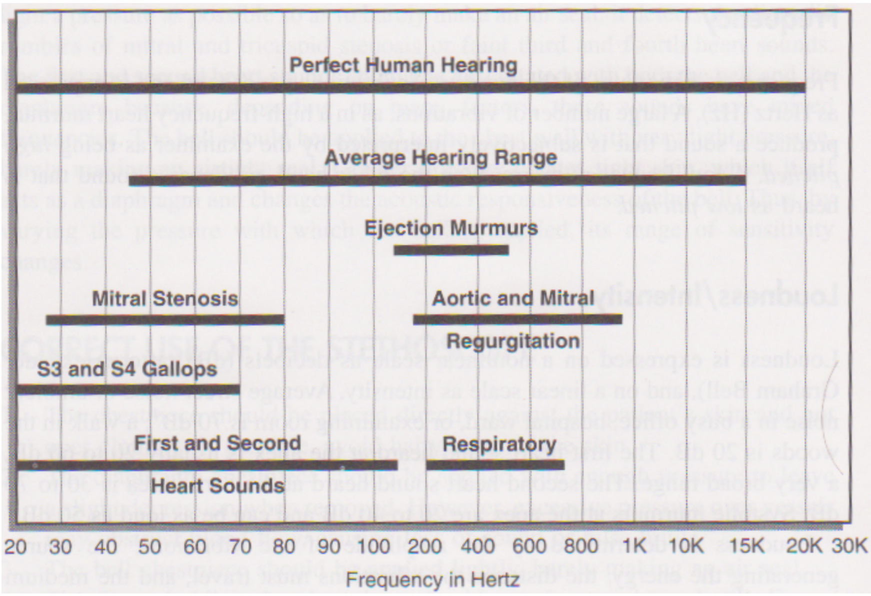


Figure 2.3: Frequencies of common heart and respiratory sounds. Source:Tilkian and Conover (2001)

There are various other heart sounds that may indicate an abnormality. Such heart sounds include clicks, pops and ejection sounds. Ejection sounds may be caused by a diseased aortic and pulmonary valve. An abnormal or stenosed mitral or tricuspid valve may result in an opening snap or click (Tilkian and Conover 2001). Rangayyan and Lehner (1987) discovered that some murmurs can occur at frequencies up to 600Hz.

2.3 Time-Frequency Analysis

Frequency analysis is an important component of phonocardiographic diagnosis. Research has found that large intensity murmurs can overlap with the first and second heart sounds (Liang et al, 1997), therefore a time-domain representation of the phonocardiographic signal alone is inadequate for diagnosis. Analysis of the heart sounds in the frequency domain can be accomplished by performing a Fourier Transform over a segment of the heart sounds.

The Discrete Fourier Transform (DFT) is defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{jn\omega_k} \tag{2.1}$$

where

$$w_k = \frac{2\pi k}{N}$$

Time information is lost as a consequence of transforming the signal into the frequency domain. The Fourier Transform is therefore useful for analysing stationary signals (signals that do not vary over time), but inadequate for the study of a signal that contains non-stationary characteristics (Lee et al, 1999). An adaptation of the Fourier Transform, the Short-Time Fourier Transform (STFT), computes the time-varying frequency of the signal by calculating the Fourier Transform over a series of short, overlapping segments of the signal (Vikhe et al, 2009). The time information is derived from the location of the current segment (window) under analysis (Lee et al, 1999). To reduce the effects of spectral leakage, each segment is passed through an appropriate window function (Obaidat and Matalgah, 1992).

The Discrete Short-Time Fourier Transform is defined by

$$X(k,w) = \sum_{n=0}^{N-1} w(n-k)x(n)e^{jn\omega_k} \tag{2.2}$$

where

$$w_k = \frac{2\pi k}{N} \tag{2.3}$$

Lee et al (1999) found that the STFT was constrained by strict limitations on the time-frequency resolution. The resolution is set by the length of the window. Thus, higher frequency components are displayed with equal precision as lower frequency components. Increasing the length of the window will increase the frequency resolution, but at the same time decrease the time resolution of the signal (Obaidat and Matalgah, 1992).

An alternative to the Short-Time Fourier Transform is the Continuous Wavelet Transform. Whilst the Fourier transform uses a sinusoidal wave to analyse the signal, the Wavelet Transform transforms a time-domain signal into the time-frequency domain with wavelets of finite energy. Unser and Aldroubi (1996) analogised the wavelet transform as a function of correlation of which maximum output occurs when the input signal most resembles the analysis template (mother wavelet).

The mother wavelet is defined by:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})dt \tag{2.4}$$

where a is the scaling parameter and b is the shifting parameter.

The continuous wavelet transform is defined as:

$$W(a,b) = \int_{-\infty}^{\infty} x(t)\psi_{a,b}(t)dt \tag{2.5}$$

The time spread is proportional to the scaling parameter a, whereas a is inversely proportional to the frequency. Thus, the wavelet transform exhibits localisation in time whereby higher frequency components are accurately displayed on the time axis.

The time interval between the closure of the aortic and pulmonary heart valve can be measured to test for a heart condition known as pulmonic stenosis (Vikhe et al, 2009). Vikhe et al discovered that it was impossible to determine the time period between the closure of the aortic and pulmonary heart valves during the second heart sound with the short time Fourier transform. In contrast, the time localisation of the wavelet transform enabled an accurate time measure between the closure of the aortic and pulmonary heart valves.

2.4 First Heart Sound Detection

The observed PCG signal can be modelled as:

$$S(n) = F(n) + C(n) + N(n) \tag{2.6}$$

Where F(n) denotes the fundamental components of the heart sound, S1 and S2, C(n) represents a mixture of other heart sound components and N(n) represents noise. The analyse of heart sounds for diagnostic purposes is therefore dependent on adequate segmentation of the heart sound. Accurate segmentation of the heart sounds greatly simplifies the identification of abnormal heart sounds from the cardiac cycle (Wang et al, 2005).

Malarvili et al (2003) demonstrated a simple method of segmenting the heart sound components by correlating the instantaneous energy of the patients ECG signal with the heart sound signal. The segmentation worked under the premise that the opening and closing of cardiac valves are preceded by electrical events of the cardiac cycle.

Iwata et al (1980) introduced a method to detect the first and second heart sounds by spectral analysis. The spectral parameters are extracted from a linear prediction process. An ECG reference is used to aid the selection of the spectral peaks for analysis by limiting the range of tracking .

Since ECG signal analysis is outside the scope of this dissertation, these methods will not be considered.

Liang et al (1997) introduced a time-domain technique of heart sound segmentation that derived an envelope from the Shannon energy principle. The envelope is filtered twice, forward and time-reversed, to remove phase-distortion and delay. After filtering, the signal is normalised to the absolute maximum amplitude of the signal envelope. The average Shannon energy is then calculated over contiguous blocks with the following formula:

$$E_s(t) = \frac{-1}{N} \sum_{i=1}^{N} x^2(i) \cdot log x^2(i) \tag{2.7}$$

Where N is the number of samples in the contiguous block segment.

The average Shannon energy is then normalised with the following equation:

$$P_a(t) = \frac{E_s(t) - \bar{E}_s(t)}{\sigma(E_s(t)} \tag{2.8}$$

Where $\bar{E}_s(t)$ and $\sigma(E_s(t)$ is the mean and standard deviation of the average Shannon energy, respectively.

The output is represented by a series peaks that correspond to the first and second heart sound, other heart sound components, and noise. A threshold is temporarily applied to remove peaks caused by low-level noise. Liang et al added a rule based algorithm to reject extra peaks caused by noise (eg speech) and recover weak heart sound components that are below the threshold. Identification of S1 and S2 follows by identifying the respective systolic and diastolic periods with the assumption that the systolic period is constant whereas the diastolic period is variable.

Liang et al continued their research on heart segmentation by developing an algorithm which used discrete wavelet decomposition and reconstruction to extract the signal within frequency bands that correspond to the first and second heart sounds. The heart sound signal was applied to fifth-level discrete wavelet transform to obtain the 1st to 5th detail coefficients as well as the 4th and 5th approximation coefficients. The signal was reconstructed with a filter bank consisting of 6th order Duabechies filters. The Shannon energy envelogram from Liang et al's earlier research was applied to the reconstructed detail and approximation frequency bands to determine the peaks that correspond to S1 and S2.

Liang et al argued that the discrete wavelet decomposition and reconstruction method offers greater immunity to previous time-domain and fixed-filter methods. Respiration noise was eliminated, however external environmental noise including speech and ambient noise caused errors during segmentation.

Wang et al (2005) argued that time-domain algorithms proposed by Liang et al and others are unreliable if the signal is contaminated by lung noises or environmental noise.

Wang et al developed an improved segmentation method by implementing a Wavelet de-noising algorithm using prior to reconstructing the coefficients applicable to the S1 and S2 frequency bands. Not unlike Liang's wavelet segmentation algorithm, the Shannon energy is calculated from the reconstructed signal and analysed to determine the peaks of S1 and S2.

2.5 Heart Rate Detection

A simple approach to determine the heart rate was outlined by Markandey (2009). The heart sound signal is first smoothed by a moving average filter defined by the following expression:

$$y(i) = \frac{1}{N} \sum_{j=0}^{N-1} x(i+j) \tag{2.9}$$

Where N is the order of the filter.

The first heart sound is then detected by calculating the maximum slope of the resulting waveform. The heart rate, HR, is calculated as:

$$HR = \frac{f_s \cdot 60}{n} \tag{2.10}$$

Wheref_s is the sampling frequency and n is the number of samples between two consecutive S1 events.

Markandey's algorithm applied a moving average to the heart rate to produce a stable heart rate figure suitable for display on a user interface.

2.6 Auto Gain Control

The purpose of an Automatic gain control (AGC) in the input sampling stage of a data acquisition circuit is to ensure that the input analogue waveform is accurately quantised by the analogue-to-digital (ADC) converter (Kang and Lidd, 1988). This is especially true in the case of linear quantisation. Weak input analogue signals result in a power signal-to-noise ratio due to quantisation noise (Young, 1995). Conversely, an AGC can attenuate high-amplitude signals to prevent saturating the ADC (ibid).

With reference to the propose phonocardiogram acquisition module, an input stage consisting of an AGC is necessary because: 1. The output characteristics of the electronic stethoscope are unknown; and; 2. Attenuation of sound due to the volume effects of the human body.

Kaniuas (2006) studied the attenuation of biosignals through the chest region. Volume effects (i.e. absorption) accounted for most of the attenuation, of which the three main causes of sound absorption in the chest were: (i) inner friction, (ii) thermal conduction and (iii) molecular relaxation. Each cause exhibited a different sound absorption coefficient. In an earlier paper, Kaniuas et al (2005) demonstrated a correlation between attenuation and the body mass index (BMI). A increased amplitude of auscultative signals were observed in patients with a higher BMI.

Kaniuas approximated the amplitude at the point of auscultation by the following equation:

$$p(r) = k \cdot \frac{p_0}{r} \cdot e^{-\alpha(r) \cdot r} \tag{2.11}$$

Where k is the constant, r = propagation distance, p0 is the sound pressure amplitude of the point source at r=0, and a(r) is the sound absorption coefficient as a function of r. Kaniuas et al (2005) observed the attenuation at likely regions of the chest to be examined by stethoscope:

It is therefore possible to conclude that a manual gain control would be inadequate for this application because the amplitude of the input signal would vary significantly during the auscultation.

A conventional analogue AGC system consists of a variable gain amplifier, a fixed gain amplifier, a signal detector, low pass filter and difference amplifier in a feedback loop (Steber, 1988). The gain of the variable gain amplifier is determined by the difference amplifier which compares the output of the signal with a reference voltage.

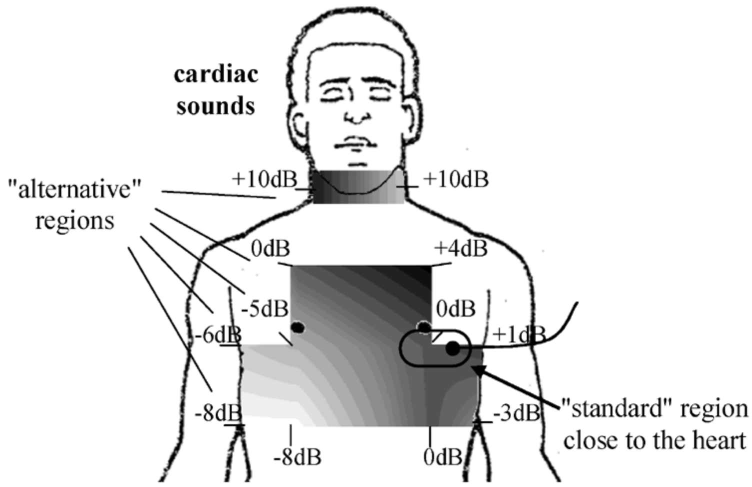


Figure 2.4: Attenuation of the human body due to volume effects Source: Kaniuas et al (2005)

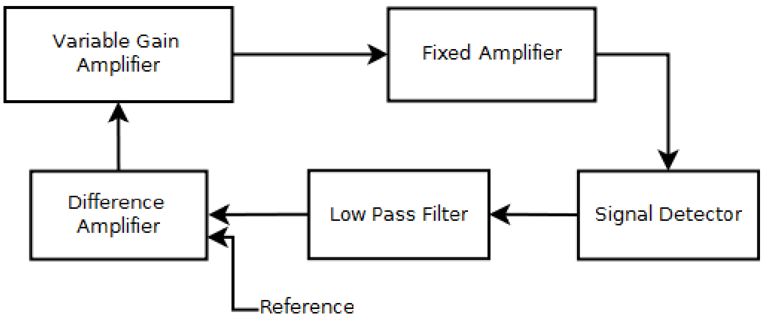


Figure 2.5: Typical Analogue AGC System

Although Steber noted that analogue AGC systems are low-cost and easily implemented in hardware, Steber identified a number of problems with the analogue AGC system: (i) Analog AGC systems tend to have a poor transient response because of the analogue filter components in the control loop. (ii) Undesirable distortion due to overloading can occur because the gain is a function of average amplitude rather than peak amplitude.

Another constraint, according to Kang and Lidd (1984), is that a gain-change within the analysis window of a time-varying waveform can introduce errors into a system involving analysis and synthesis. This not only creates problem for the de-noising stage of the phonocardiogram acquisition module if a transform/thresholding algorithm is applied to the signal, but also for any subsequent analysis of the heart sounds.

Using digital signal processing techniques, Steber implemented an automatic gain control system that was equivalent to the analog system. The signal detector was implemented by extracting the positive half-cycles from the waveform. If any part of the half-cycle is above the noise-floor threshold, the signal would be multiplied by a gain factor to increase the amplitude of the signal to the maximum value. The noise-floor threshold is pre-determined by the characteristics of the input signal.

Kang and Lidd (1984) introduced a automatic gain control (AGC) algorithm based on a LPC encoder that used low-band energy estimation for voice activity detection. The algorithm used a probably density function to compute the mean value of the low-band energy of the signal to smooth out fluctuations caused by sudden changes in loudness, leakage of higher frequency components and ambient-noise. The error signal of the control loop was calculated by subtracting the mean of the low-band energy from a reference level.The gain would then be incrementally adjusted by an incremental gain with a non-linear relationship to the error.

One of the advantages of Kang and Lidd's AGC algorithm is that gain is not adjusted during unvoiced (non-speech) periods. Commenting on the auto gain control algorithm, Kang and Lidd noted that steady state was achieved within a few seconds and remained stable with unnecessary gain recalculations.

Archibald (2008) built upon the research of Steber and Kang et al by adding a proportional-integral (PI) controller for detecting voice activity. One distinguishing aspect of Archibald's auto gain control system is that it incorporates an adaptive noise detection algorithm, whereas the methods proposed by Steber and Kang et al required a LPC encoder to detect speech and non-speech periods. The adaptive noise detection algorithm utilises a PI controller to estimate the noise floor level for voice activity detection.

Stationary noise is determined by computing the variation of signal energy within an envelope. A flat variation indicates stationary noise. In contrast, an envelop with a high variation of signal energy indicates a period of voice/activity. In the event of a non-voice period, the gain is set to 0. Otherwise, the gain is calculated by:

$$G = \frac{DesiredAmplitude}{PeakAmplitude} \tag{2.12}$$

Which is a rather simplistic method to correct the gain. On commenting on the proposed algorithm, Archibald noted that the quality of the output signal was dependant on the rate of gain change. Audible zipper noise will occur if the gain change is too fast, whereas noise amplification and clipping can occur if the gain change is too slow. This behaviour observed by Archibald corresponds to the transient response of under-damp and over-damped second order systems respectively (Nise, 2000).

Archibald's algorithm could be improved by applying a proportional-integral-derivative (PID) controller to the amplitude error calculated by:

$$Error = DesiredAmplitude - PeakAmplitude \tag{2.13}$$

Ogata (1995) expressed the continuous PID controller as:

$$m(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{dt}\right] \tag{2.14}$$

The discrete PID controller can be represented as a difference equation of:

$$m_k = q_0e_k + q_1e_{k-1} + q_2e_{k-2} + m_{k-1} \tag{2.15}$$

Where coefficients $q_0 = K\left(\frac{T_d}{T}\right)$, $q_1 = -K\left(1 + \frac{2T_d}{T} = \frac{T}{T_i}\right)$ and $q_2 = \frac{KT_d}{T}$ for a rectangular approximation of integration. Coefficients q_0, q_1 and q_2 can be determined experimentally,and/or computed with a maximum descent algorithm (Aigner et al).

2.7 Performance Characteristics of Voice Activity Detection

Beritelli et al (2002) identified several parameters which characterised the performance of a voice activity detection algorithm:

- Front End Clipping (FEC): Clipping introduced in passing from noise to speech activity.

- Mid Speech Clipping (MSC): Clipping due to speech misclassified as noise.
- OVER: Noise interpreted as speech due to a late detection of the transition from active to silence.
- Noise Detected as Speech (NDS): Noise interpreted as speech within a silence period.

To reduce the probability of front end and mid speech clipping from occuring, Woo et al (2000) proposed the formation of a hysteresis based on the estimated noise floor.

2.8 Ambient Noise Cancellation

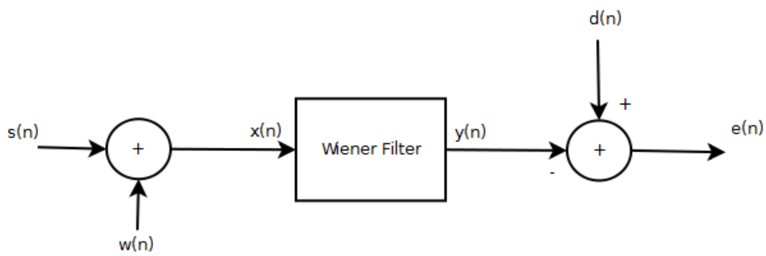
The quality of heart sounds acquired by a phonographic device can be impaired by internal (bodily) and external noise sources. Leading causes of noise include respiration sounds, movement of the patient, movements of the stethoscope (shear noise) and external environmental noises (Varady, 2001).

Grumet (1993) identified numerous external environmental noise sources in a clinical environment. Call buttons, telemetric monitoring systems, electronic intravenous machines, patient-activity monitors and personal movement are typical examples of environmental noises that were identified by Grumet. Grumet measured an average noise level of 67db in acute care admission and general medical wards at night. A study into noise pollution in a hospital setting by Cabrera and Lee (2000) supported this observation in a separate study that found the noise levels in a typical urban hospital would often exceed 55db.

Whilst much of research into noise within a clinical setting focuses on the psychological impact on patients, the cancellation of ambient noise is of utmost importance to phonocardiography which requires a signal with a relatively high signal to noise ration for accurate analysis and diagnosis (Zhoa, 2005).

A conventional method of noise cancellation is to apply a fixed FIR or IIR filter to the signal input. However a fixed filter will not eliminate all ambient noise from the signal because ambient noise can be caused by various sources at various frequencies and signal intensities (Liang et al , 1997). Another difficulty is presented by the process of selecting the frequency range of the fixed frequency band without degrading the useful heart sounds components in signal (Varady, 2001). A superior de-noising technique would involve the use of an adaptive filter as the impulse response of an adaptive filter is adjusted automatically to operate under changing conditions and minimise the signal error (Widrow et al, 1975).

The Weiner filter is an optimal filtering method that suppresses noise without degrading the useful components of the signal (Widrow et al, 1975). The impulse response of a Weiner filter is designed so that the output closely approximates the characteristics of the expected signal (Proakis and Manolakis, 1996). Consider the following model:



From above model, the error can be mathematically represented as:

$$e(n) = d(n) - y(n) \tag{2.16}$$

An ideal filter will reduce the mean-square error to zero.

The FIR Weiner filter of length M can be defined as

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \tag{2.17}$$

Where $h(k)$ represents the coefficients of the filter.

The objective of the Weiner filter is to minimise error. From equation (2.16), the mean square error is :

$$\varepsilon_M = E|e(n)|^2 = E|d(n) - \sum_{k=0}^{M-1} h(k)x(n-k)|^2 \tag{2.18}$$

Minimization of ε_M yields:

$$\sum_{k=0}^{M-1} h(k)\gamma_{xx}(l-k) = \gamma_d x(k) \tag{2.19}$$

Where γ_{xx} is the autocorrelation of the input signal and $\gamma_d x(k)$ is the cross-correlation between the expected and input signals, that is $E\left[d(n)x^*(n-k)\right]$ (Proakis and Manolakis, 1996).

Equation (2.19) can be expressed as:

$$\Gamma_M h_M = \gamma_d \tag{2.20}$$

Where $\Gamma_M h_M$ is a M x M dimension Toeplitz matrix comprising of the autocorrelation of the input signal and γ_d is the cross-correlation vector of the expected and input signals (Proakis and Manolakis, 1996).

To solve for the optimal filter coefficients, h_M:

$$h_M = \Gamma_M^{-1}\gamma_d \tag{2.21}$$

The above equation takes the form of a series of Yule-Walker equations which can be efficiently solved by the Levinson-Durbin algorithm (Proakis and Manolakis, 1996).

One disadvantage of the Wiener filter proposed above, according to Widrow et al (1975), is that it requires a "priori" knowledge of the expected signal characteristics. However, this information can be recorded in a noise free environment and stored in memory.

Boll (1979) proposed a subtractive noise suppression algorithm that obtained the noise spectrum during periods of inactivity. The signal is buffered into contiguous frames and windowed to eliminate spectral leakage. The fast Fourier transform (FFT) is applied to the signal and averaged over successive frames. During periods of non-speech activity, the noise floor level, also known as the bias, is estimated.

The bias is then subtracted from the the magnitude of the spectrum of speech periods. Values with a negative magnitude are set to zero (Boll defines this process as half-wave rectification). Boll proposes an additional step that involves selecting the minimum

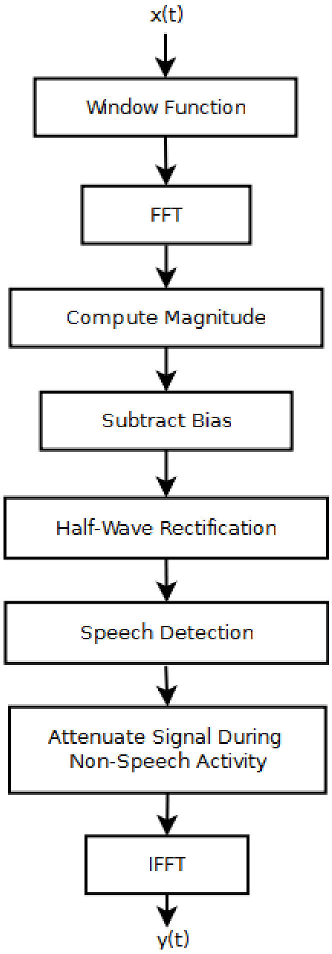


Figure 2.6: Flow chart of subtractive noise suppression algorithm. Source: Boll (1979)

magnitude value from three adjacent frames where the magnitude is less than the noise floor level calculated in an earlier step. The signal is attenuated during periods of non-speech activity, and finally transformed back into the time domain with an inverse fast Fourier transform (IFFT) function.

Boll's spectral subtraction algorithm presents a problem in relation to phonocardiographic signals. Heart sounds are non-stationary signals (Iwata, 1980), thus averaging the magnitude may cause temporal smearing of short transitory sounds (Boll, 1979). Scarlat (1996) noted that the power spectral method proposed by Boll produced unnatural artefacts described as "musical noise"

Varady (2001) introduced a method of de-noising phonocardiographic signals with an adaptive wavelet filter. Varady's filter decomposed the heart sounds and a noise reference into coefficients with the discrete wavelet transform. A rule based algorithm was applied to perform cross-channel cancellation of noise whereby the wavelet coefficients of the noise reference were subtracted from the wavelet coefficients of the heart sounds. Adaptive thresholding was applied to the resulting coefficients to remove residual noise. The signal is reconstructed with the inverse wavelet transform.

Varady's algorithm assumes the presence of a second transducer to provide the noise reference, however the noise can be extracted from non-speech (or non-heart beat) sections as demonstrated by Archibald (2008) and Boll (1979).

Zhao (2005) studied a form of wavelet shrinkage that derives the threshold function from the Stein Unbiased Risk Estimate (SURE). The Stein Unbiased Risk Estimate is a statistical function that adaptively optimises the threshold levels used to remove noise from the signal. Zhao's wavelet shrinkage method makes the assumption that the energy of the useful components will be concentrated in a few coefficients of the wavelet transform, whereas noise will be uniformly distributed. Therefore, Zhao's algorithm is expected to be effective against ambient environmental noise (eg air conditioning) but not so effective against non-stationary noise such as speech or crying.

2.9 Signal Encoding Techniques

In order to transmit the heart sounds wirelessly, the heart sounds must be encoded in a digital format that is resilient to a high noise environment. The two encoding methods covered in this section are (i) Pulse code modulation (PCM) and (ii) Continuously variable slope delta modulation (CVSD).

Pulse code modulation (PCM) is a common technique for converting analogue signals into a binary code for transmission.The amplitude of each sample is represented by a word of data. A significant disadvantage of PCM encoding is the higher bandwidth requirements compared to single-bit word encodings such as CVSD (Young, 1994). Despite the simplicity of PCM, Prabhu et al (2006) argues that PCM is more prone to interference than CSVD.

Continuously variable slope delta modulation (CVSD) is an adaptation of adaptive delta modulation. A sample is represented by a single bit which refers to change in the amplitude of the signal. CVSD encoders typically require a higher sample rate due to the reduction of bits per sample (Prabhu et al 2006).

2.10 Conclusions

The continuous wavelet transform was judged to be the best transform for representing phonocardiographic signals in the time-frequency domain due to the superior resolution properties over the short time Fourier transform. An automatic gain control algorithm based on the PI controller was deemed to be superior to alternate techniques examined by this review. An adaptive filter will filter noise far effectively than a fixed filter, however an empirical approach is required to select the best filter given the constraints of the hardware.

Chapter 3

Signal Processing

3.1 Chapter Overview

This chapter reviews fairly important signal processing principles that are applicable to this project. Topics including voice activity detection, automatic gain control and noise reduction will be investigated in this chapter.

3.2 Voice Activity Detection

3.2.1 Introduction

Voice activity detection (VAD) is the process of classifying "silent" and "voiced" periods in a signal. In this research project, voice activity detection is applied to signal obtained from a digital stethoscope. Thus, "voiced" periods refer not to human speech, but rather to auscultation noises (eg heart beats). As "silent" periods often contain ambient noise, it is possible to perform a spectral estimation of noise during the silent periods. Thus voice activity detection forms an integral component of the noise reduction algorithms discussed later in this chapter.

This chapter will evaluate and discuss the performance of a number of techniques to detect activity in a signal, namely:

- Energy Method
- Entropy Method

Each method follows a similar process:

1. Segment the signal into frames of 64. 2. Calculate the energy or entropy of the signal 3. Smooth the value with a moving average. 4. Determine if the value is greater than the estimated noise floor. If it is greater, the segment is a heart sound. If the value is lower than the noise floor, the segment is silence. 5. The noise floor is estimated from the silent segments of the signal.

3.2.2 Energy of Signal Method

Energy is often used as a measure of activity in a signal. A periodic signal over a finite time is said to have high energy, whereas a silent period will have significantly low energy.

The energy of a discrete-time signal can be found by:

$$E = \sum_{i=0}^{N-1} |x(i)|^2 \tag{3.1}$$

Alternatively, in accordance with Parseval's Theorem, the energy of a discrete signal may also be found from the frequency domain by:

$$E = \frac{1N}{\sum}{}^{N-1}_{k=0} |X(k)|^2 \tag{3.2}$$

The signal is divided into short segments of 64 samples. Segments of high energy indicates activity, whereas low energy indicates silence.

The standard deviation of signal energy can also be used as a foundation to a VAD algorithm. Segments showing a high degree of standard deviation indicate a periodic signal with significant variation (e.g. a heart beat) The standard deviation of signal energy is calculated by:

$$E = \sqrt{\sum_{i=0}^{N-1} |x(i) - E_{mean}|^2} \tag{3.3}$$

However experimentation with standard deviation method did not indicate any significant performance boosts over the energy method.

3.2.3 Spectral Entropy Method

The entropy of a random sequence is a measure of the unpredictability, or disorganisation, of a sequence. A signal consisting of white noise is inherently unpredictable, therefore the entropy is high. The periodic nature of a heart sound is more organised, therefore the entropy approaches zero. This hypothesis can be applied to signal processing for the detection of useful sounds (eg heart sounds) in a noisy signal.

Shannon's equation to calculate entropy in a sequence is:

$$E = \sum_{i=0}^{N-1} p(i) \cdot log(p(i)) \tag{3.4}$$

where p is the probability density function (PDF) of a signal.

The PDF can be estimated from the power spectral density (PSD) of the signal:

(3.5)

where X is the fourier transform of the signal.

Figure 3.1 shows the design of the entropy based VAD used by this project.

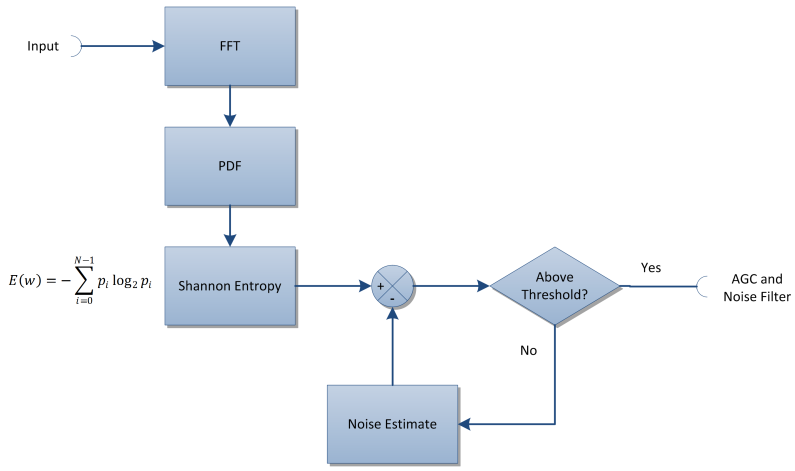


Figure 3.1: Spectral Entropy Method of Detecting Signal Activity

3.2.4 Adaptive Thresholding

Noise is seldom constant. The amplitude of noise may change abruptly by simply turning on or turning off an air-conditioner. A robust voice activity detector algorithm must take into account variability of noise. Algorithms with a fixed threshold designed for less noisy environments may incorrectly detect noise as activity when operated in a noisy environment. Conversely, an algorithm designed for noisy environments may detect weak signals as noise. Therefore, the threshold must adapt to the noise levels in any given environment.

The noise threshold is estimated by calculating the entropy of the signal during non-active segments in between heart beats. In the event of a continuous auscultation sound (eg gallop rhythm), the signal is estimated before the stethoscope is applied to the chest and between measurements. The noise threshold is "smoothed" by a function that is analogous to a PI controller. For the entropy method:

$$N(t) = (1 - 0.999) * E(t) + 0.999 * N(t - 1); \tag{3.6}$$

Where N = noise and E = entropy of the current segment.

From this value, a hysteresis is formed by calculating a separate threshold for noise and voice segments. The hysteresis allows for the transition phase of the heart sound (i.e inactivity to activity, and vice versa) to complete before changing state. The hysteresis also provides a degree of tolerance for short variations in entropy during the active and inactive segments.

Setting the range of the hysteresis function is an art in itself. If Ts is too high (remembering that a high entropy indicates predictability of the signal), each heart sound will be truncated as the amplitude drops below the threshold. However this behaviour occurred rarely during testing, due to the PI behaviour of the threshold estimation algorithm. On the other hand, if Tn is set too low, the transition stage from non-activity to activity is detected late, resulting front end clipping (FEC), as defined by Beritelli et al(2002), is introduced.

A signal completely absent from noise will have a noise floor of zero, thus the entire signal will be a "voiced" segment.

3.2.5 Attenuation of Non-Envelope Segments

To improve the perceptive audio quality of the signal and ultimately increase the signal-to-noise ratio, non-voice (noise only) segments are removed from the signal.

3.2.6 Test Method

The three methods discussed in this section were tested for resiliency to error against 2 test signals as follows:

- Clean normal heart beat signal
- Normal heart beat signal with additive white Gaussian noise
- Normal heart beat signal superimposed with ambient hospital noise

The clean normal heart beat is clearly segmented into the two normal heart sounds, S1 and S2.

A small amount of Additive White Gaussian Noise (AWGN) was added to the clean signal to model ambient noise. A real sample acquired from a hospital is then superimposed with the signal to investigate the performance of the VAD in a real environment.

3.2.7 Test Results

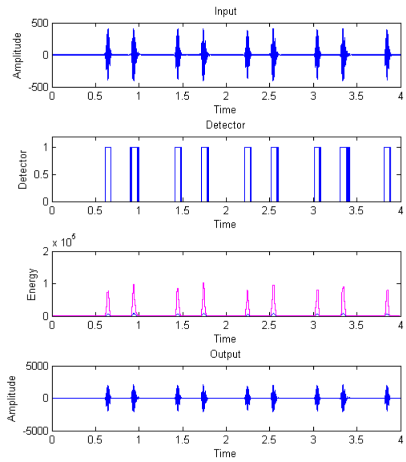


Figure 3.2: VAD Test - Energy of a Clean Signal

Figure 3.2 shows the performance of the energy method when applied to a clean signal. As expected, the heart beats are detected as periods of activity.

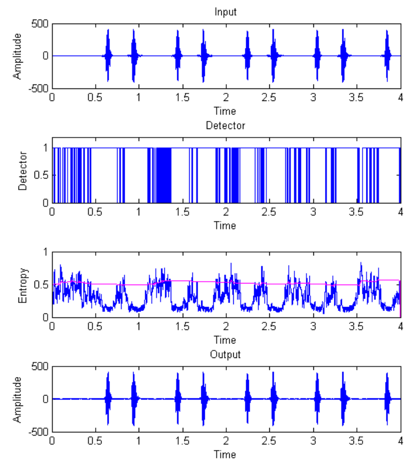


Figure 3.3: VAD Test - Entropy of a Clean Signal

Figure 3.3 was a little unexpected at first glance. Most of the signal is detected as active periods. The adaptive threshold is set low due to the absence of ”disorganised” signals, such as white noise.

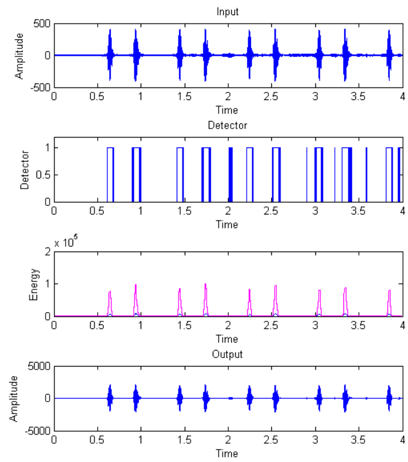


Figure 3.4: VAD Test - Energy of a Signal with Real Ambient Noise

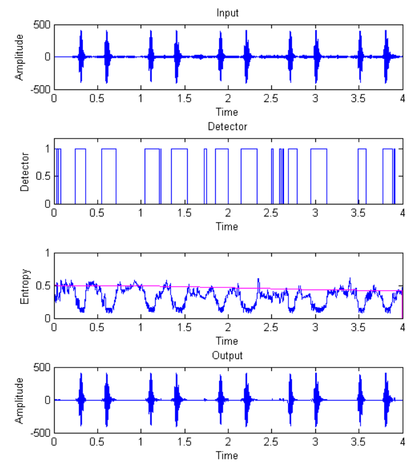


Figure 3.5: VAD Test - Entropy of a Signal with Real Ambient Noise

The performance of the energy and entropy methods for real noise are shown in figures **??** and 3.5 respectively. Both methods performance comparitively well.

The entropy method offers greater immunity to additive white Gaussian noise (Figure 3.7) than the energy method (Figure 3.6).,

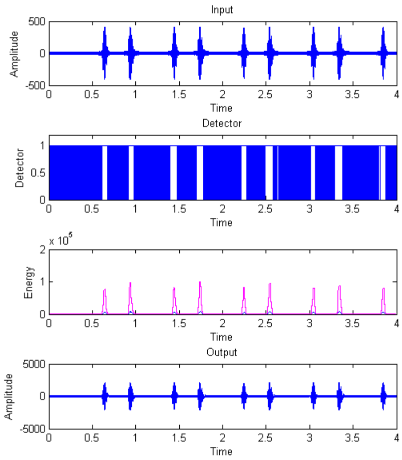


Figure 3.6: VAD Test - Energy of a Signal with AWGN

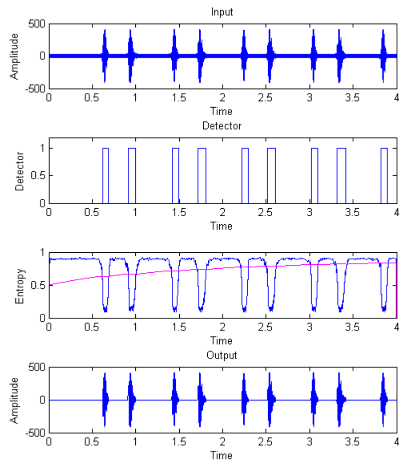


Figure 3.7: VAD Test - Entropy of a Signal with AWGN

3.2.8 Conclusion

Entropy based method was deemed most suitable due to its resilience to noise. The noise threshold was adjusted using a PI controller. The entropy was smoothed with a moving average filter. Non-voice segments are attenuated to improve SNR.

3.3 Automatic Gain Control Algorithm

3.3.1 Design

The automatic gain control serves several important purposes, including the follow:

- BMI
- Amplify weak signals
- Attenuate signals to prevent clipping

The system design of the auto-gain algorithm proposed for this project is shown in figure **??**.

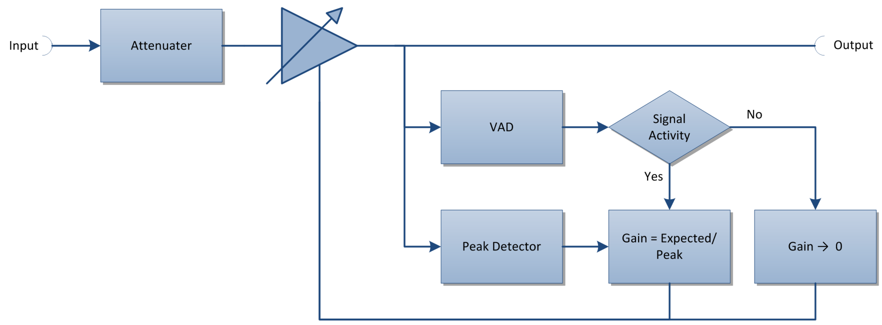


Figure 3.8: Automatic Gain Control Algorithm

3.3.2 Programmable Gain Amplifier (PGA) Controller

A hardware attenuator divides the input signal by 4. The signal is then applied to the input of a programmable gain amplifier which is controlled by the digital signal controller. The PGA selected for this project will amplify the input signal with a gain of 1, 2, 4, 5, 8, 10, 16 or 32.

The required gain is calculated by dividing the expected gain by the actual gain, and selecting the nearest value from table 3.1.

Gain		
Attenuation	PGA Gain	Total Gain
1/5	1	1/5
	2	2/5
	4	1
	5	1
	8	8/5
	10	2
	16	16/5
	32	32/5

Table 3.1: PGA Gain Settings

3.3.3 Transition State

To prevent abrupt sudden changes in gain during a transition state, the peak amplitude of the last 16 segments are stored in a sliding window. The maximum value is selected from the sliding window.

3.4 Spectral Subtraction Noise Cancellation

Adaptive Spectral Subtraction, as shown in figure 3.9 involves a statistical analysis of the signal to detect silent periods, of which the spectra of the ambient noise is calculated and stored in memory. During an active period, the FFT of the signal is determined and noise is subtracted. Further thresholding of the signal may be applied at this point. The signal is transformed back to the time domain, ideally in a de-noised state.

To evaluate the performance of the spectral subtraction algorithm, additive White Gaussian Noise was injected into a clean signal. The results are shown in figure 3.10.

It was discovered that the dsPIC signal controller selected for this project would not be powerful enough for a useful implementation of this algorithm. One possible alternative

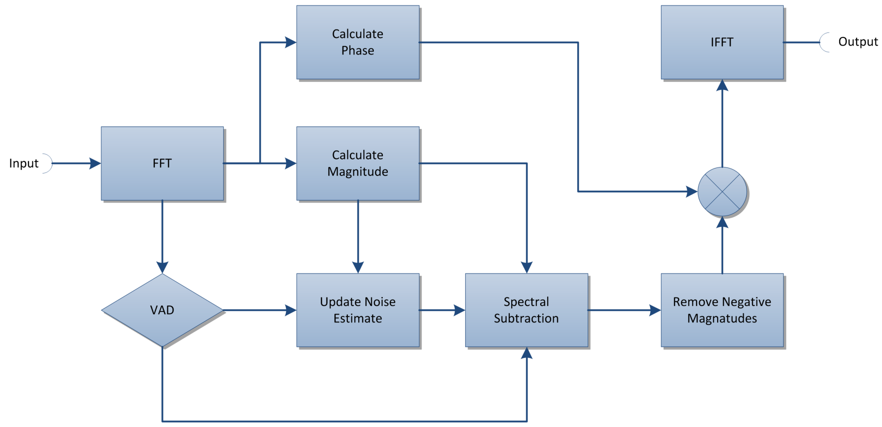


Figure 3.9: Spectral Subtraction

is to remove the noise at the host PC instead.

3.5 An Evaluation of Data Communication Protocols

3.5.1 Introduction

During the prototyping of the wireless acquisition module, it was discovered that the data transmitted from the onboard Bluetooth modem did not always arrive at the host. This section investigates a few simple methods of sending ”connectionless” packets over an unpredictable medium such as wireless, and methods of mitigating error due to data loss.

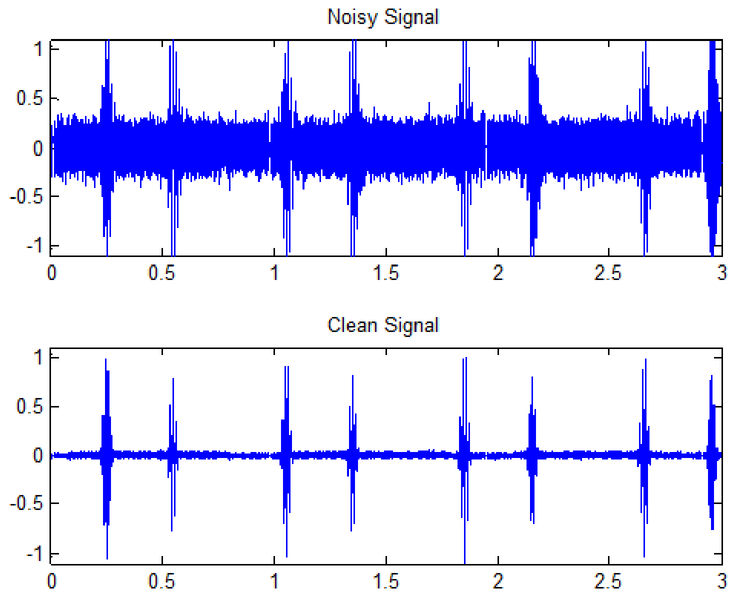


Figure 3.10: Spectral Subtraction

3.5.2 Raw PCM

This approach involved streaming raw PCM, without metadata (eg header), to the desktop PC. The Wave header would be prefixed to the stream for playback on the desktop storage. Data obtained from the 12 ADC was stored in two bytes, hi and low (the 4 highest significant bits were padded with 0s), and streamed to the PC one byte at a time.

This approach preserved the full integrity of the acquired signal, however without a method to synchronise the signal at the host, the stream was often read in the wrong sequence (eg lo byte from the previous packet read as hi, hi packet from the current packet read as lo) if a byte was dropped by the communication link.

3.5.3 Custom Data Structure

DLE, STX, LEN, DATA, DLE, ETX

Where LEN refers to the length of the packet, including control characters and DATA is the sampled auscultation signal of a variable length.

If DLE occurred during the data segment, another DLE would be prefixed to the data byte. The length of the packet was included to ensure the data integrity at the other end, since in theory, the frame could be of variable length. If a mismatch was detected, the packet would be dropped by the host and the sequence would be filled with 0s for length N.

The implementation of this method was not immune to error however. The presence of more than two consecutive DLE characters in the signal would cause errors at the receiver end.

3.5.4 uLaw and aLaw

The 12 bit signal would be up-scaled to 16 bit, and then companded to an 8 bit logarithmic value. Both standard preserve much of the signal.The benefit of this approach is that the data does not need to be encapsulated in a data packet. i.e. the companded data can be streamed raw. The data is already in a format that can be sent to a remote host via VOIP technology. The disadvantage is the inherent loss in reducing the bit resolution of a sampled signal.

3.5.5 Conclusion

Companding the signal into a logarithmic PCM value offered the least chance of error.

3.6 Chapter Summary

This chapter investigated voice activity detection, automatic gain control, noise reduction and data transmission protocols.

Chapter 4

Hardware and Firmware Design

4.1 Chapter Overview

This chapter covers the hardware design aspects of the project. A top-down approach was employed to design the circuit: Starting from the system level and ending with the schematics.

4.2 Specifications

The hardware was designed to fulfil the following requirements:

1. The input shall accept line level signals from a digital stethoscope.
2. The acquisition module shall operate from a single voltage source of 3.3V
3. Frequency components less than 1000Hz shall be sampled
4. The signal path shall be protected by galvanic isolation
5. The signal shall be transmitted to a PC over a wireless connection.

4.3 System Design

4.3.1 Acquisition Module

The system consists of the following modules:

- Input Stage and Automatic Gain Control
- Analogue-To-Digital Converter (including anti-aliasing filter)
- Signal Isolation and power isolation
- Digital Signal Controller
- Bluetooth Modem

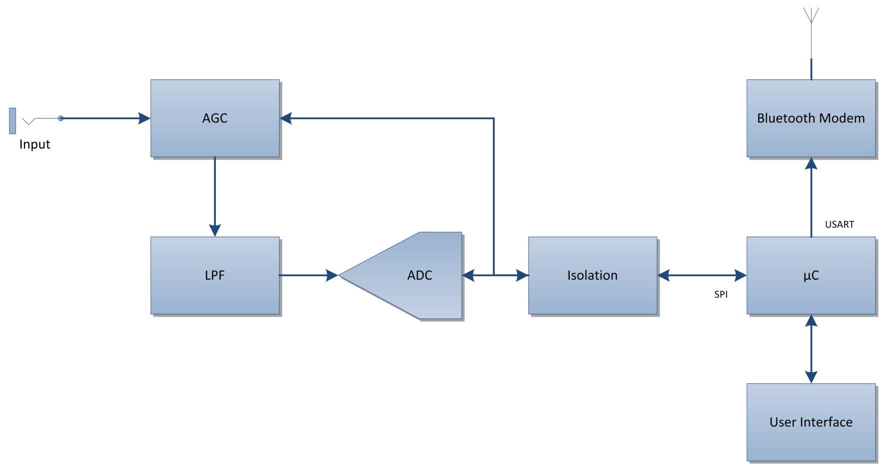


Figure 4.1: Hardware System Design

The schematics of the acquisition module are listed in Appendix C.

4.3.2 Receiver

The receiver side, as shown in figure 4.2, consists only of a Bluetooth modem and PC.

No hardware design was required at the receiver end. An "off-the-shelf" Bluetooth adapter was used to receive auscultation signals from the acquisition module.

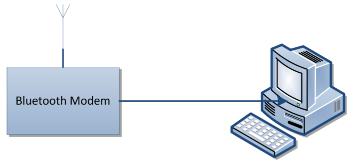


Figure 4.2: Hardware System Design

4.4 Automatic Gain Control

4.4.1 Input Stage and Fixed Attenuator

The output of the electronic stethoscope is capacitively coupled to the input stage of the wireless module to remove the DC component from the input. This was a necessary compromise because the external signal source references system ground, whereas the acquisition modules operates from a virtual ground referenced at Vdd/2 due to the single-supply operation of the circuit. When a voltage source that is referenced to system ground is connected to a op-amp stage that is referenced to virtual ground, a non-negligible DC offset exists at the input. The presence of this offset is problematic when the input was DC coupled because of the limited dynamic range available to the amplifiers (and analogue-to-digital converter) due to the low-voltage constraints of the circuit.

The solution is to AC couple the input, however this comes at a price. The decoupling capacitor in series with the resistance network of the attenuator forms a high pass filter which is undesirable as heart sounds contain valuable data at very low frequencies. A sufficiently large capacitor is therefore required to preserve as much of the low frequency components of the signal as practically possible. Given a Thevenin equivalent resistance of 1M for the attenuation circuit, a capacitor value of 0.22uF would provide a cut off frequency of 0.723Hz.

The attenuator is required for two reasons: 1. Whilst the line-level for consumer products is rated at -10db (0.316V RMS), the reference model of digital stethoscope used for design work was rated at 2.0V peak to peak. This exceeds the voltage swing of the op-amps, leading to clipping of the signal peaks. 2. The attenuation allows for better usage of the gains provided by the PGA. The signal can be attenuated when the gain is set less than 4, held constant at 4 and amplified at gains greater than 4.

The signal is attenuated by a resistive network and op amp configured to provide an approximate attenuation rate of 1:4 and an input impedance of approximately 1M Ohm. Line level outputs commonly have a very low output impedance (around 6-30 Ohms), therefore loading effects are negligible.

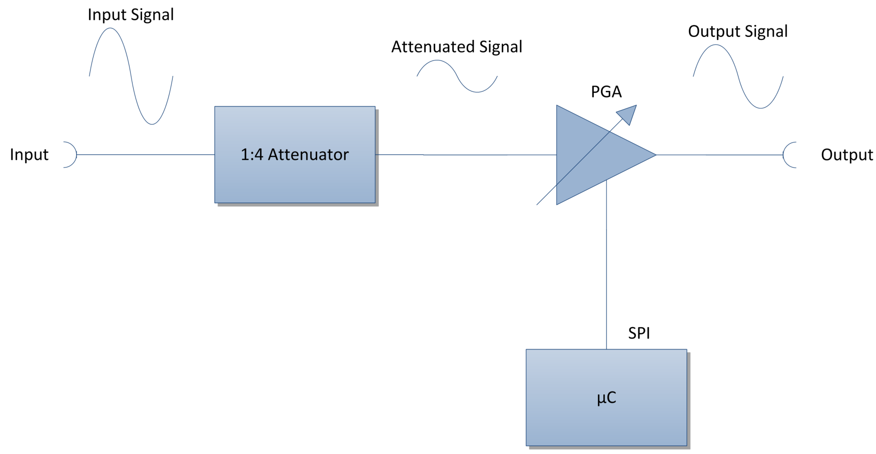


Figure 4.3: Design of Automatic Gain Control Hardware

Due to the low-voltage requirement of this circuit, an op-amp with rail-to-rail amplification was required to maximise the dynamic range of the circuit. The op-amp selected for this task was the MCP601, a CMOS op-amp especially designed for signal-rail applications.

The attenuator and AC coupled input stage was simulated by MICROCAP, as shown in 4.4.

4.4.2 Programmable Gain Amplifier (PGA)

The attenuated signal is amplified by a programmable gain amplifier (PGA) which is controlled by the microcontroller over an isolated SPI bus. The Microchip MCP6S21 was selected for this task. Like the MCP601 single supply op-amp, the MCP6S21 is designed to operate on a single supply and provides rail-to-rail input and outputs. The MCP6S21 also features a software shut down mode to conserve battery life which can be enabled by the SPI interface. The device is woken upon receiving a new gain

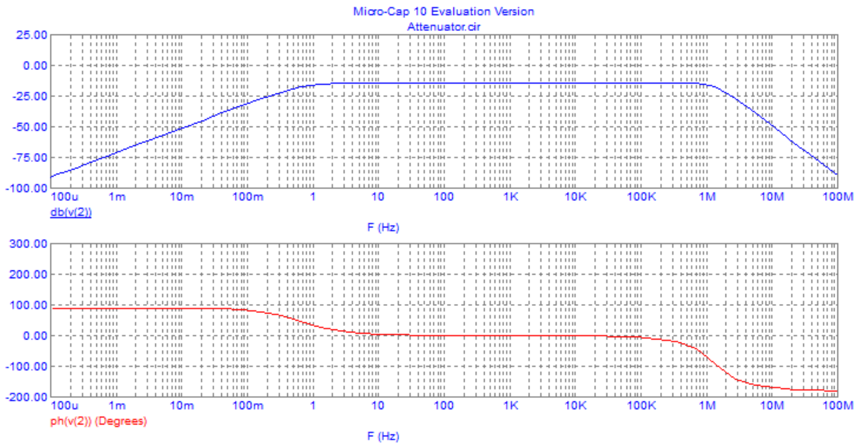


Figure 4.4: Frequency Response of Input Stage

instruction.

The PGA can amplify the signal at gains of 1, 2, 4, 5, 8, 10, 16 and 32. Thus, factoring in an approximate attenuation ratio of 1:4, the resulting signal will be attenuated/amplified by a factor of 0.25, 0.5, 1, 1.25, 2, 2.5, 4 and 8 respectively.

A test point is provided after the PGA for testing and validation.

4.4.3 SPI Interface

The PGA is controlled by a unidirectional SPI bus consisting of the clock, data-in and chip-select lines. Gain is set by first pulling the chip select low. This instructs the device to begin accepting serial data from the SPI bus. The chip-select will remain at the low logic state until the instruction and gain select bytes have been sent.

This is followed by setting the 'Write to Register' command bit (bit 7) of the instruction register, as shown by table 4.1. All other command bits are set to 0. The resulting instruction byte in hexadecimal is 0x40.

After the instruction register is set, the first (least significant) three bytes of the gain register, as represented by table 4.2, to a value that represents the desired gain.

Where XXX refer to the gain select bits shown in table 4.3.

Instruction Register							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	0	0	0	0	0	0

Table 4.1: MCP6S21 Instruction Register

Gain Register							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	X	X	X

Table 4.2: MCP6S21 Gain Register

Gain		
Gain	Setting (Decimal)	Setting (Binary)
2	0	000
4	1	001
5	2	010
8	3	011
10	4	100
16	5	101
32	6	111

Table 4.3: MCP6S21 Gain Select Bits

Finally, the chip-select is returned to the nominal high logic state to instruct the PGA to process the instruction and gain registers.

The PGA adjusts the gain when the two following conditions are met:

1. A 16 bit word, consisting of the instruction and gain bytes, is fully sent.
2. The chip select is released (i.e. pulled high)

4.5 Data Acquisition

4.5.1 Anti-Alias Filter

Elementary Shannon-Nyquist theorem states that the sampling frequency must be at least twice the highest frequency component of the sampled signal. The most common practice is to filter the signal with a low pass filter to remove frequency components above the Shannon-Nyquist frequency.

The first design performed anti-aliasing by a 2nd order low-pass Butterworth filter in a standard Sallen Key topology. The filter was designed with a cut off frequency of 1khz for an expected sampling rate of 2kHz. The frequency response is show in figure 4.5.

However, it was observed that the 2nd order filter did not provide an adequate roll-off for anti-aliasing purposes. As a result, frequencies well above 1kHz were sampled which introduced unwanted artefacts in the discrete signal due to the effects of aliasing.

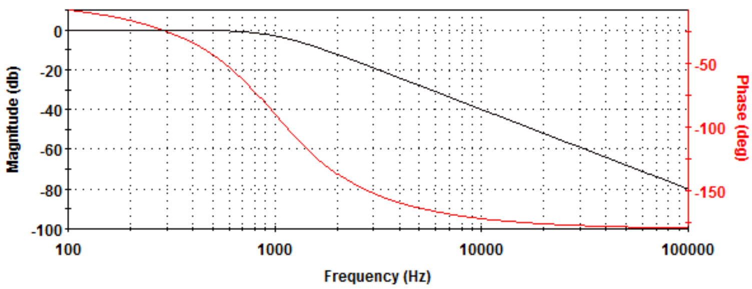


Figure 4.5: Frequency Response of a 2nd Order Butterworth Filter

The filter was redesigned for a Chebychev response which provided a sharper roll-off than the Butterworth (at the expense of a larger "ripple" in the passband). This provided only marginal improvement over the Butterworth filter, as seen in figure 4.6.

There were two obvious flaws with the design:

1. The transition region crossed over the Nyquist frequency.
2. The sampling frequency was two low.

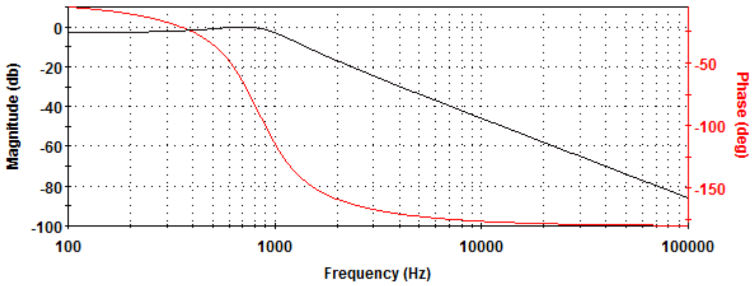


Figure 4.6: Frequency Response of a 2nd Order Chebychev Filter

The solution was to increase the order of the anti-aliasing filter and increase the sampling rate to take into account the non-ideal properties of a low-pass filter (namely, the roll-off within transition region).

The signal-to-noise ratio of an ideal analogue-to-digital converter can be calculated by:

$$SNR = (1.763 + 6.02b)dB \tag{4.1}$$

Where b = bit resolution.

An ideal 12 bit ADC will have a SNR of 74db. It is therefore desirable to design a low-pass filter with an attenuation of at least -74db at 1/2 the Nyquist frequency or lower. The final design consisted of an 8 pole Butterworth low-pass filter with a gain of -74db at 2905Hz as shown in figure 4.7. The sampling rate was increased to 8kHz.

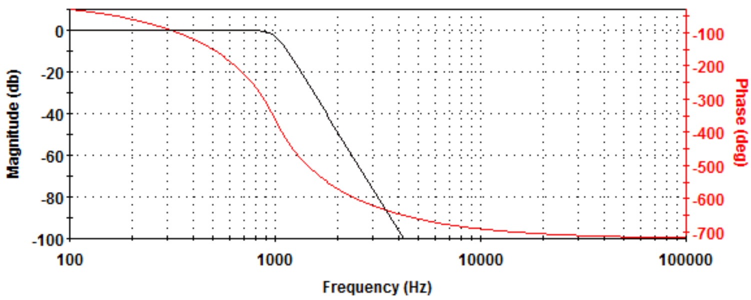


Figure 4.7: Frequency Response of a 8th Order Butterworth Filter

The implementation of the filter comprised of 4 op-amps in Sallen Key topology. Very few of the calculated resistor values were available commercially from local suppliers, therefore equivalent resistances were constructed with a trim technique as shown by table 4.4.

Resistor Values				
Desired	R1	R2	Final Resistance	Error (%)
9.76	10	390	9.75	0.102
21.5	22	910	21.48	0.090
10.7	12	100	10.71	0.134
15.8	22	56	15.79	0.032
7.68	8.2	120	7.68	0.059
3.65	3.9	56	3.65	0.107

Table 4.4: Trimmed resistor values used by the anti-aliasing filter

4.5.2 ADC

The signal is then sampled by a 12bit ADC and transferred to the microcontroller over the SPI bus. The reference voltage is tied to the Vdd (3.3V) rail so that Vdd/2 becomes the centre point.

4.5.3 SPI Interface

The conversion process is triggered by pulling the chip-select low. After the sample is acquired and converted, the ADC will stream the digitised sample to the microcontroller until all 12 bits are transferred. The ADC will continue to stream 0s until the chip-select is reset to high logic state. Low power mode is enabled when the chip-select is high.

4.6 Power Supply

4.6.1 Battery Management

The acquisition module was designed to operate on battery. The benefits of battery operation include true wireless functionality and isolation from mains power supply. A rechargeable single cell lithium-ion battery was considered ideal for this application given its superior energy to weight ratios and slow loss of charge when not in use. However the charge process for a lithium-ion batteries requires special monitoring and control, therefore battery management was implemented with the help of the Microchip MCP73812.

4.6.2 Voltage Regulation

The digital signal processor operates on a voltage of 3.3V, whereas the nominal supply voltage is 3.7V when operating from battery or 4.20V when powered by an external power source (eg wall adapter). Regulation is therefore required given a the DSC's specified maximum voltage of 3.6V. The relatively small margin of 0.4V requires the use of a low drop-out (LDO) regulator.

The MCP1700 satisfies this requirement, providing a stable output voltage of 3.3V with a typical overhead of 178mV. As per conventional linear regulator applications, the input and outputs are bypassed with a capacitor to reduce noise and improve stability of the regulator circuit.

4.6.3 Single Supply Design

As this will be a single supply (0-3.3V) circuit, a virtual ground was designed to provide a reference voltage to the op-amps. The op-amps selected for amplification and filtering support single rail operation, however because the input signal swings below 0V, the input must be biased at V/2. A simple solution would be to bias the inverting input of the op-amp with two resistors, as shown by figure 4.8.

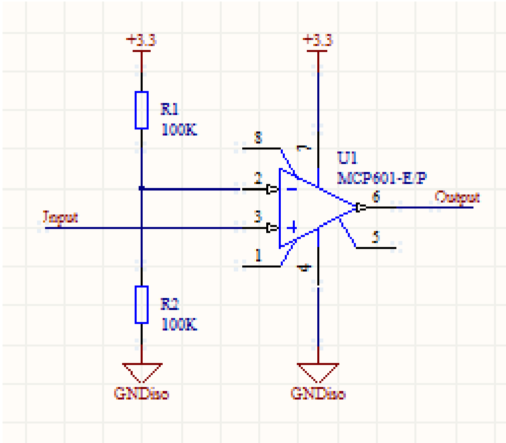


Figure 4.8: Pseudo-Ground

However this circuit is subject to small variances in input voltage, resister drift and mismatches in resistance. A far better option is to reference the the input to a virtual earth which is formed by an unity gain op amp, as shown by 4.9

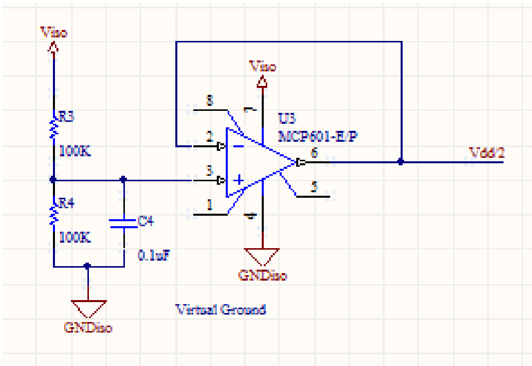


Figure 4.9: Virtual-Ground

The output voltage is fixed at V/2 and is not subject to the same constraints as the resistor bias circuit. The bypass capacitor is placed in parallel to the second resistor to filter Johnson noise caused by the relatively large resistances.

4.7 Isolation

4.7.1 Medical Standards

Roy et al discovered that currents as low as 100 mA can paralyse the respiratory system and cause the heart muscle to fibrillate (1976). Leakage current could originate from the mains earth conductor, from another external device, or from the patient via the applied part.

IEC 60601-1 defines a set of safety standards for electronic medical equipment. IEC 60601-1 standard was adopted by Australia under AS/NZ 3200.1 and may be used to support the electrical safety component of an application to register the device under the Australian Register of Therapeutic Goods (ARTG).

Any medical device that comes into physical contact with a patient is defined by the IEC 60601-1 as an applied part. The diaphragm of an electronic stethoscope is placed against the patient's chest, often for cardiac diagnosis. As such, an electronic stethoscope is categorised by the IEC as type BF applied part. Type BF medical devices must be separated from the earth to prevent dangerous leakage current flowing through the patient to ground (or vice versa).

Even though the acquisition module is completely isolated from the mains supply (eg powered by battery), there is still the risk of leakage current electrically coupled to the enclosure of the modules that must be mitigated by proper isolation techniques.

4.7.2 Signal Isolation

The IEC 60601 requires medical equipment to withstand an electrical fast transient of 1kV for input/output lines, and up to 2kV protection against surges.

In this design, the isolation occurs after the signal is sampled. In other words, the SPI bus and chip select lines are isolated, as opposed to the analogue signal input. Whilst isolating the analogue signal input is possible, the circuitry required is far from trivial given the non-linear characteristics of opto-coupler and transformer based isolators. The Writer investigated several analogue isolation amplifier solutions available on the market, however none were suitable for a battery powered project (for example, many required dual voltage rails of +/- 15V).

The SPI bus to the ADC and PGA, and the chip select lines, are isolated. Signal isolation is provided by the ADUM2400 digital isolator by Analog Devices. The ADUM2400 is fully compliant with the requirements of IEC 60601-1 and is certified for use in medical applications.

4.7.3 Power Supply Isolation

Signal path isolation provides little benefit unless the power supply is sufficiently isolated. The power supply is isolated by an isolated 3.3V DC-DC converter.

The NKE0303DC is compliant with Underwriters Laboratory (UL) to UL 60950, which specifies an identical distance through isolation (DTI) to IEC 60601. The DTI is the internal-clearance between conductors inside the isolation device. Protection up to 3kV is provided.

The device is a switch-mode converter operating at a typical switching frequency of 115kHz with a specified worse case ripple voltage of 80mV peak to peak. It is therefore necessary to filter the output to reduce the ripple voltage. As recommended by the data sheet, an LC filter was applied to the output in order to attenuate the ripple. The LC notch filter was designed a resonant frequency of 23.215kHz on the premise that the switching frequency of the DC-DC converter would be maintained well above this level. SPICE simulation confirmed that the rippled would drop to 3.9mV.

It should be noted that whilst the Author is confident that the DC-DC converter selected for this project meets most, if not all, of the requirements of IEC 60601, the device is not certified for medical equipment. The cost of a fully certified DC to DC converter was beyond the budget of the project. Production of this device is therefore not recommended without further revision to the power supply isolation of the circuit.

4.8 Digital Signal Controller

A digital signal controller is a variant of traditional microcontrollers that provide barrel shifters and multiply accumulators (MAC) which are used extensively in digital signal processing applications

The first step in the design process was to identify a digital signal controller that fulfilled a set of criteria, including:

- Low voltage (3.0-3.6V)
- UART for communications to the Bluetooth module
- SPI communication module to control the PGA and ADC.
- Low pin count (Desirable, but not mandatory for the prototype)

This narrowed the field down to two microcontrollers: Texas Instruments MSP430 family and Microchip dsPIC33 family. The dsPIC33 was chosen because it offered more RAM, faster processor speed and a hardware USART.

To simplify the circuit design, the microcontroller is clocked by an internal PLL at 80MHz (40 MIPS). An ISCP interface is provided for on-board firmware updates and debugging.

4.9 Bluetooth Modem

Wireless data communications will be provided by a Bluetooth module via the microcontrollers hardware USART. The BlueSMiRF Gold (see figure 4.10) was selected because it fully supports the service discovery (SDP) and RFCOMM protocols, and is easily controlled by an AT-like command set.

Figure 4.10: BlueSMiRF Gold Bluetooth Modem

4.10 User Interface

The user interface was kept simple, consisting only of an LED to display when data capture is in process, and a push button to enable and disable data capture.

Following conventional design principles, current to the LED is supplied by the microcontroller through a current limiting resistor. A GPIO pin was set aside for this purpose.

A simple momentary switch shorts a pull-up resistor to ground when it is pressed. The input pin is mapped to a interrupt which invokes a callback from the interrupt service routine (ISR).

4.10.1 Switch De-bouncing

A common problem with mechanical switches, shown in figure (**?**), is that the transition between on and off is rarely clean. The conductive contacts 'bounce' as they are moved to the on or off position. The mechanical oscillations caused by the bounces is also known as as 'switch bounce'.

One common solution involves placing a capacitor in parallel to the switch to damp the mechanical oscillations, as shown in (**?**). Slight variations in voltage due to the capacitor charging/discharging when the switch bounces are filtered by the hysteresis input of a Schmitt inverter.

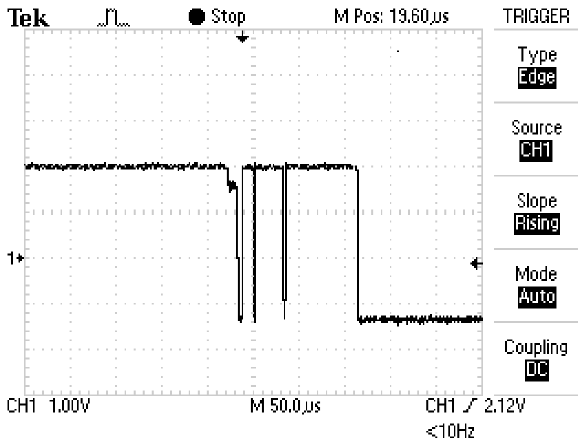


Figure 4.11: Mechanical switch 'bounce'. Source: http://www.labbookpages.co.uk/

To simplify the circuit design and reduce the number of components required, a hardware de-bouncing solution was not implement. Instead, a simple software de-bouncing algorithm was implemented to filter the transition between on and off.

The interrupt will call the de-bounce algorithm which performs the following:

1. Check the status of the switch input. If the switch input is low, increment the counter. If the switch is high, the switch has bounced - reset the counter.
2. Sleep for 1 millisecond
3. Repeat 10 times

The program will toggle the data capture mode if the input is held for 5 consecutive milliseconds.

4.11 Electromagnetic Compatibility

Featuring a switch mode DC-to-DC converter and type 1 radio device, the acquisition module is inherently a noisy device. Each IC is decoupled by a 0.1uF capacitor to filter electrically coupled noised on the supply rail. Separate ground lines are provided for analogue and digital devices.

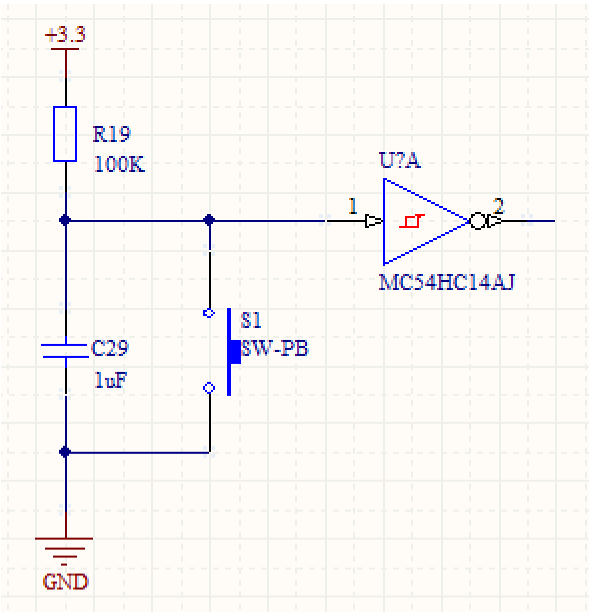


Figure 4.12: Hardware de-bouncing circuit

4.12 Chapter Summary

This chapter discussed the hardware design of the acquisition module, including the following topic:

- Input Stage and Automatic Gain Control
- Analogue-To-Digital Converter (including anti-aliasing filter)
- Signal Isolation and power isolation
- Digital Signal Controller
- Bluetooth Modem

Chapter 5

Software Design

5.1 Chapter Overview

The wireless acquisition module designed and implemented during this project would not be very useful without a means of receiving the data for further analysis. The software solution discussed in this chapter will capture the auscultation signal from any Bluetooth adapter that supports the RFCOMM protocol, display the signal on the screen and playback the signal to the PC's sound card.

5.2 System Design

A simplified

5.3 Data Capture

5.3.1 Bluetooth Interface

Microsoft Windows will detect the BlueSMiRF Gold Bluetooth modem when it is within range and automatically install the necessary drivers that will emulate an RS232 con-

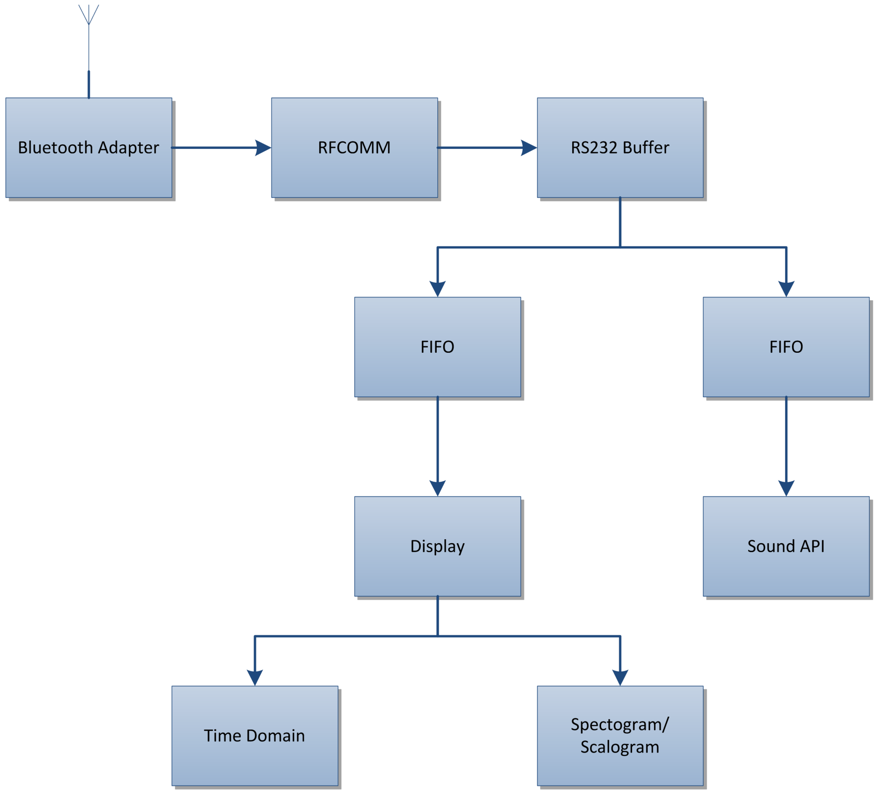


Figure 5.1: Data flow of host application

nection over the RFCOMM/SPP protocol, as shown by figure 5.2.

This simplifies development considerably, as the service-discovery (SDP) and low level logic-link control are performed by the operating system. Communication with the acquisition module can be achieved by opening a connection to a visualised serial port (eg COM3).

5.3.2 Asynchronous Serial Communication

Rather than poll the serial port buffer for data at fixed timed intervals, most modern programming languages provide a convenient event-driven component for asynchronous communications. That is, a new thread is created to poll the port in the background. This allows the program to perform computationally complex algorithms (eg Fast Fourier Transform, update graphs, etc) while the serial port waits for new data.

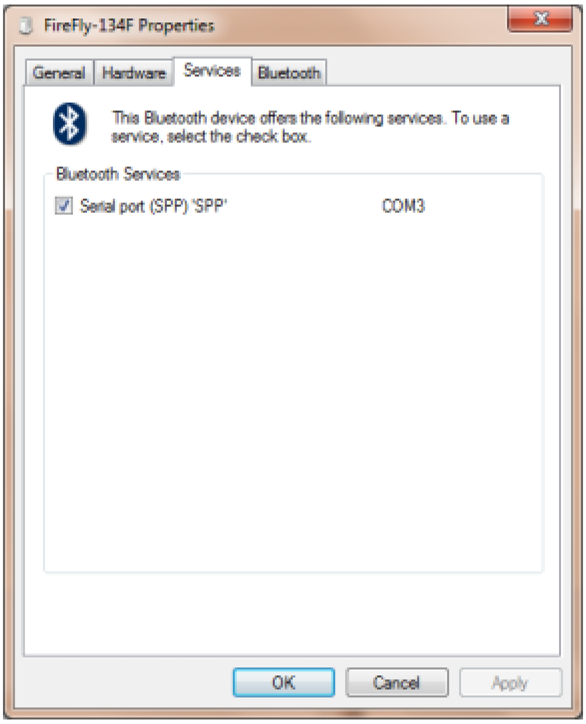


Figure 5.2: Bluetooth Properties in Windows 7

When new data arrives, the component buffers the data into a memory stream and raises an event . The data is then transferred into two local FIFO buffers from the memory stream. One buffer is used by the graphing module, while the other is used by the real time sound play back module.

One important design consideration is the need for thread synchronisation when reading and writing to the buffer. If two or more threads attempt to access the same memory space simultaneously, a race condition could occur leading to unpredictable values. One work around is to lock the buffer to prevent other threads from accessing while it is in use.

5.3.3 uLaw to PCM Conversion

To display and playback the heart sounds, the data was converted back to linear PCM. The function `int ulaw2linear(byte ulawbyte)` of the C# source reconstructs the companded sample into a 16 bit linear PCM value. Note that the information lost during the companding process can not be restored. The reconstructed signal will however retain most of the dynamic range of the original signal. The function `ulaw2linear` was ported to C# from Java code originally developed by Sun Microsystems (Now Oracle).

5.4 Realtime Sound Playback

The function `private void InitSound()` establishes a DirectSound playback device and creates a secondary buffer for double buffering the sound stream. A new thread is created to transfer data from the primary buffer (i.e. the FIFO data structure that is filled with data from the Bluetooth adapter) into the secondary buffer. The playback device then plays the contents of the secondary buffer.

5.5 Real Time Graphing

5.5.1 Time Domain Graph

The time domain representation of the auscultation signals, also known as a phonocardiogram, is updated at a fixed timed interval from the FIFO data structure that is populated by the serial port event handler.

5.5.2 Spectrogram (Short-Time Fourier Transform)

The Short-Time Fourier Transform provides a time-frequency representation of the signal in real time. The signal is segmented into frames of 256 words. The Fast Fourier Transform (FFT) is then calculated for each frame. The second half of the transform is removed as this data is not required. The magnitude of the transform is then displayed. The FFT algorithm is provided by the MathNet Numerics library.

5.5.3 Scalogram (Discrete Wavelet Transform)

Some experimentation was performed. Similar to the Short-Time Fourier Transform above, the signal was segmented into frames, except this time the size of the frame was 128 words. The discrete wavelet transform employing an array of Duebechies D4 filter banks was calculated for each frame. The magnitude of the DWT was displayed. The DWT was very similar to the STFT in many respects, however it provided better time-frequency resolution at higher frequencies.

The Daubechies D4 wavelet transform algorithm was ported to C# code from Java code developed by Ian Kaplan (2002).

5.6 Chapter Summary

This chapter outlined the solution used to stream data from an RFCOMM compliant Bluetooth adapter for desktop PCs and laptops. The signal can then be play backed through the system's sound card, and displayed in the time and frequency domains for further analysis by a trained professional.

Chapter 6

Implementation and Testing

6.1 Chapter Overview

This chapter deals largely with the implementation and testing of the wireless acquisition module and phonocardiogram PC application. The challenges encountered during the implementation stage are discussed in this chapter.

6.2 Hardware and Firmware

6.2.1 Breadboard

The hardware was partially constructed on breadboard, pictured in figure 6.1, primarily to learn more about the dsPic and Bluetooth modem, but also to test key components of the hardware including the anti-aliasing filter and SPI interface to the ADC. Although the constructed circuit occasionally sent useful data to the host (PC), the circuit was unstable. The stray capacitance of the breadboard and the high clock speed of the SPI bus (10MHz and greater) were a recipe for spurious oscillations.

Absent from this prototype were the isolation and auto-gain control stages.

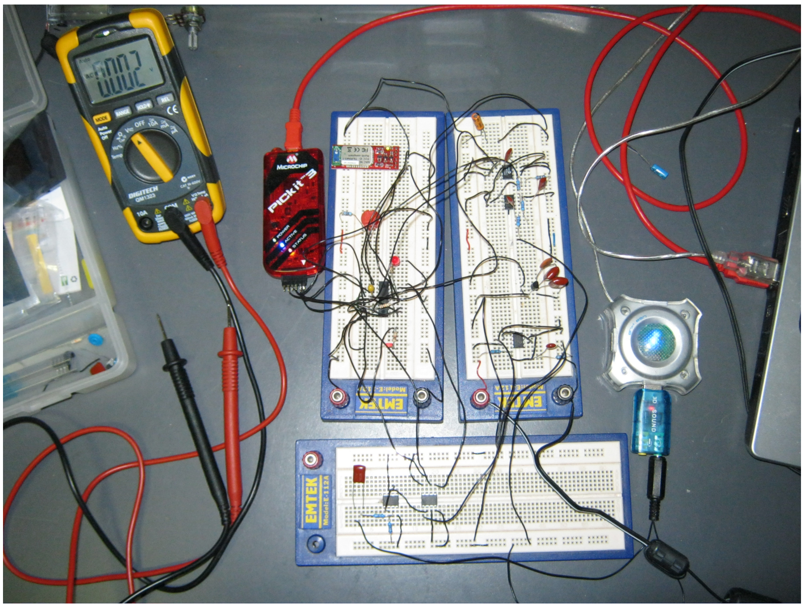


Figure 6.1: Breadboarded Prototype

6.2.2 Prototype PCB (Stripboard)

The parts were delicately transfered to a stripboard shown in figure6.2. The isolation, AGC and power supply components were added to this prototype to complete the circuit.

6.2.3 Firmware Development

The firmware was developed in C and compiled with the MPLAB C30 compiler. The code was edited, built, deployed and debugged in Microchip's MPLAB IDE.

The dsPic was programmed and debugged with the PicKit 3 In-Circuit Debugger, pictured in figure 6.4. The PicKit 3, when invoked by MPLAB, provided full debugging and flash ROM programming capabilities through the ICSP interface that was implemented as part of the hardware.

Due to the low pin count of the microcontroller, the alternate programming ports (PGED2 and PGED1) were used instead of the default programming ports to free up reprogrammable ports for the hardware serial modules (i.e. USART and SPI).

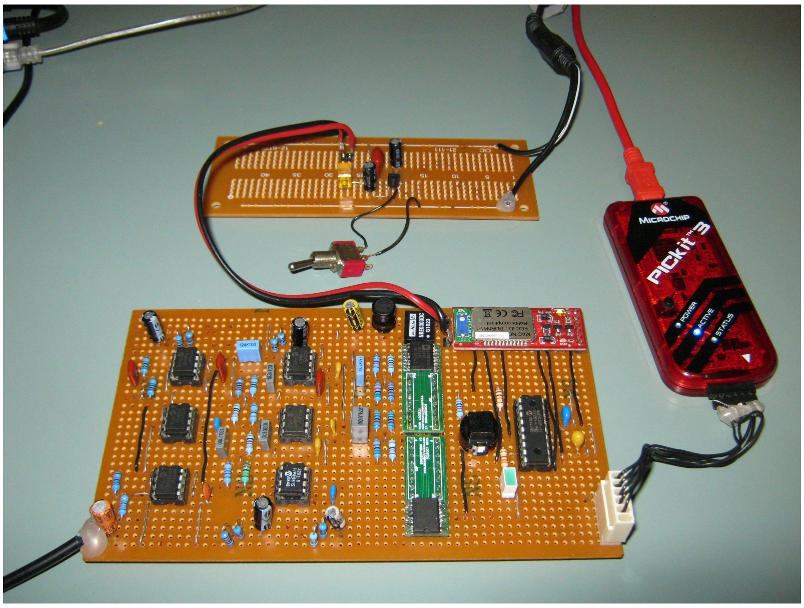


Figure 6.2: Completed prototype on prototype board

6.2.4 Serial Communications

The dsPic was interfaced to the Bluetooth modem with the hardware USART module. Ready-to-send (RTS) and clear-to-send (CTS) were configured for hardware flow control. Hardware flow control is not mandatory as it is possible to tie the RTS and CTS pins together at the bluetooth modem end, however they were retained to prevent data loss in the event that the internal buffer of the modem was full. The transmit data (TxD) pin at the dsPic end was connected to the receive data (RxD) pin at the modem end, and vice versa. The USART was configured for a baud rate of 115200.

6.2.5 Automatic Gain Control

Initial attempts to control the programmable gain amplifier (PGA) by the digital signal controller's internal SPI port failed. The troubleshooting process verified the correct timing of the chip select line. Although unorthodox, an AC measure from a digital multimeter confirmed the presence of the clock and data signals. Decreasing the clock speed and changing the SPI mode made no difference. Interestingly, the SPI ADC worked perfectly. The Writer did not have a logic analyser or oscilloscope with sufficient bandwidth to fully debug the SPI problems, so unfortunately the problem was never

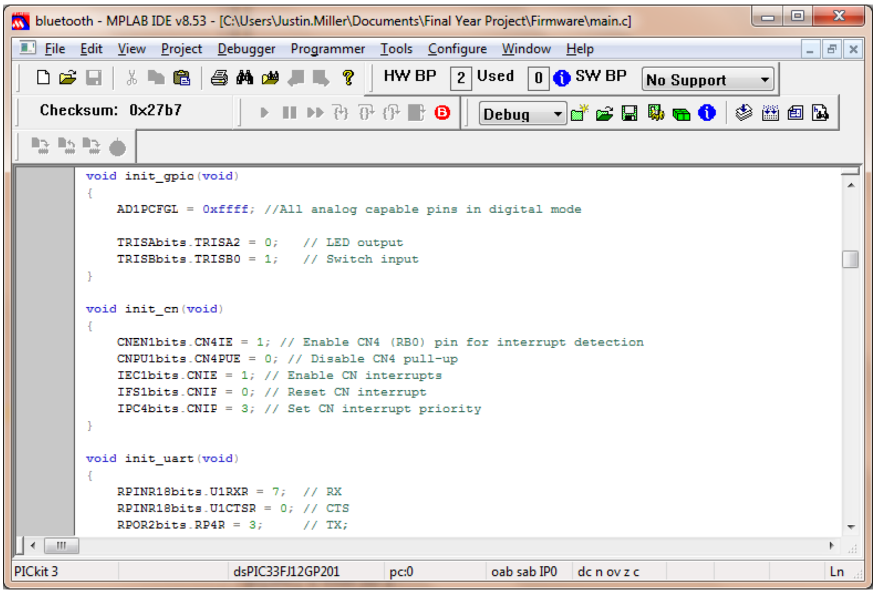


Figure 6.3: MPLAB Integrated Development Environment

Figure 6.4: PicKit 3 In-Circuit Debugger. Source: http://www.microchip.com

resolved.

However, an alternate solution was found. Emulating the SPI controller by "bit-banging" the appropriate ports solved the problem completely.

1. Temporarily disable the internal SPI controller. 2. Set the chip select for the PGA low. 3. Send the first bit (starting at the LSB) to the data-in line. 4. Set the clock high 5. Delay for 50 cycles 6. Set the clock low 7. Shift the data byte left 8. Repeat until the complete byte is fully sent. 9. Set the chip select high to set the gain. 10. Re-enable the internal SPI controller.

6.2.6 Analog to Digital Convertor

The SPI interface to the analogue to digital converter worked without any major difficulties. Although the ADC did not have a Data Input (DIN) pin, the SPI controller still required the program to write to the SPI buffer in order for the SPI controller to set the "data ready" register.

6.3 Software

The phonocardiogram application was developed with C# in Visual Studio 2010. Targeting the WinForms API, the application acquired data from the asynchronous serial port control, displayed the signal on the screen and directed the acquired sound to the sound card.

6.3.1 Graphing

The auscultation signals were displayed on-screen with a graphing component entitled PlotLab. PlotLab was designed for real time instrumentation and performed remarkably well for auscultation sounds.

The spectrogram requires further work, as evident by figure 6.5. The number of bins could be reduced to 2048 or beyond to provide greater resolution. While this would increase computational complexity of the FFT calculations, modern desktop PCs and laptops should have no difficulty handling the extra workload.

6.3.2 Sound Playback

The DirectSound API was used to play sound on the PC. The sound occasionally jittered, suggesting that the size of the secondary buffer was not large enough to account for minor delays in the stream. The artefacts introduced by companding the signal were also noticeable. A sound delay estimate at 2 seconds was also very noticeable. This

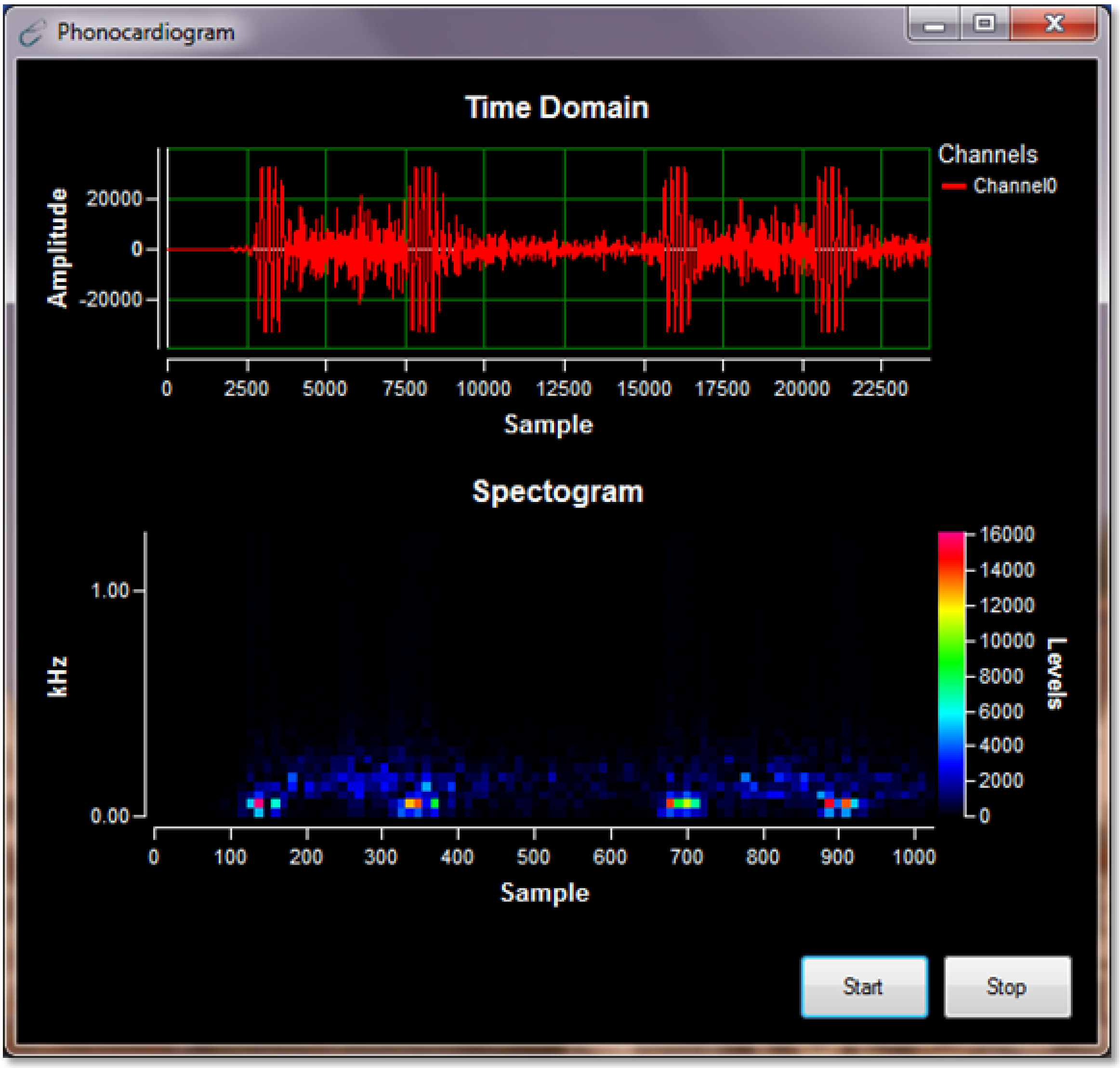


Figure 6.5: Phonocardiogram Application

delay could potentially cause echoes, requiring an echo cancellation filter if the loud speakers are turned on.

6.4 Chapter Summary

Some of the challenges encountered during the implementation of the wireless acquisition module and phonocardiogram PC application were discussed in this chapter. Some possible improvements include an increase in the number of bins for the short-time Fourier transform and an enlargement of the secondary buffer of the sound playback algorithm.

Chapter 7

Conclusions and Further Work

7.1 Achievement of Project Objectives

The following objectives have been addressed:

Literature Review Chapter 2 evaluated the characteristics of heart sounds and review research into various signal processing and acquisition methodologies.

Signal Processing Chapter 3 proposed and simulated algorithms for automatic gain control and noise reduction.

Hardware Design Chapter 4 discussed the hardware design of the wireless acquisition module.

Software Design Chapter 5 discussed the software design of phonocardiogram application that displayed ascultation sounds aquired by the wireless acquisition module.

7.2 Further Work

7.2.1 Server Side Signal Processing

The dsPic digital signal controller selected for this project was unfortunately too underpowered to perform any serious adaptive noise cancellation. While there are far more powerful products on the market designed for this tasks (eg dedicated DSP, FPGA, etc), the noice cancellation could be performed at the PC end where CPU resources are not a scarcity.

7.2.2 Telehealth

With the growing availability of fast internet services in rural areas, telehealth is becoming a practicle option for isolated patients and medical services. For this project to be practical in a telehealth context, futher research is required in the areas of:

- Real time stream of auscultation signals during a VOIP or video conference.
- Echo cancellation
- Real time data compression
- Add support for Google Health, Microsoft Vault and/or Federal Governments eHealth Initiative

7.2.3 Decision Based Segmentation of Heart Sound Components

The phonocardiogram presented by this project requires the expertise of a trained medical practioner to analyse. Segmentation of the heart sounds could lead to an aid for diagnosis, if for example, a heart murmour could be identified from a heart signal. This could be helpful as a training tool, or as a screening tool for experienced specialist. A decision based segmentation algorithm could rely on one or more concepts:

- ECG Reference

- Shannon Energy Principle
- Artificial neural network
- Wavelet Transform

References

Archibald, F. (2008), *Automatic Level Controller for Speech Signals Using PID Controllers*, Texas Instruments, Dallas, Texas.

Beritelli, F., Casale, S., Ruggeri, G. & Serrano, S. (2002), 'Performance evaluation and comparison of g.729/amr/fuzzy voice activity detectors', *Signal Processing Letters, IEEE* **9**(3), 85 –88.

Bovenzi, M. & Collareta, A. (1984), 'Collareta a. noise levels in a hospital.', *Industrial Health* **22**, 75 – 82.

Cabrera, I. N. & Lee, M. H. M. (2000), 'Reducing noise pollution in the hospital setting by establishing a department of sound: A survey of recent research on the effects of noise and music in health care', *Preventive Medicine* **30**(4), 339 – 345.

Grumet, G. W. (1993), 'Pandemonium in the Modern Hospital', *N Engl J Med* **328**(6), 433–437.

Huiying, L., Sakari, L. & Iiro, H. (1997), A heart sound segmentation algorithm using wavelet decomposition and reconstruction, *in* 'Engineering in Medicine and Biology Society, 1997. Proceedings of the 19th Annual International Conference of the IEEE', Vol. 4, pp. 1630 –1633 vol.4.

Iwata, A., Ishii, N., Suzumura, N. & Ikegaya, K. (1980), 'Algorithm for detecting the first and the second heart sounds by spectral tracking', *Medical and Biological Engineering and Computing* **18**(1).

Kang, G. & Lidd, M. (1984), Automatic gain control, *in* 'Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '84.', Vol. 9, pp. 120 – 123.

Kaniusas, E. (2006), 'Transmission of body sounds: an overview', *ULTRAGARSAS* **1**(56).

Kaniusas, E., Pfutzner, H. & Saletu, B. (2005), 'Acoustical signal properties for cardiac/respiratory activity and apneas', *Biomedical Engineering, IEEE Transactions on* **52**(11), 1812 –1822.

Lee, J. J., Lee, S. M., Kim, I. Y., Min, H. K. & Hong, S. H. (1999), Comparison between short time fourier and wavelet transform for feature extraction of heart sound, *in* 'TENCON 99. Proceedings of the IEEE Region 10 Conference', Vol. 2, pp. 1547 –1550 vol.2.

Liang, H., Lukkarinen, S. & Hartimo, I. (1997), Heart sound segmentation algorithm based on heart sound envelogram, *in* 'Computers in Cardiology 1997', pp. 105 –108.

Malarvili, M., Kamarulafizam, I., Hussain, S. & Helmi, D. (2003), Heart sound segmentation algorithm based on instantaneous energy of electrocardiogram, *in* 'Computers in Cardiology, 2003', pp. 327 – 330.

Markandey, V. (2009), *Digital Stethoscope Implementation on the TMS320VC5505 DSP Medical Development Kit (MDK)*, Texas Instruments, Dallas, Texas.

Nise, N. (2000), *Control Systems Engineering 3rd Edition*, John Wiley and Sons, New York, USA.

Noponen, A.-L., Lukkarinen, S., Angerla, A. & Sepponen, R. (2007), 'Phonospectrographic analysis of heart murmur in children', *BMC Pediatrics* **7**(1), 23.

Obaidat, M. & Matalgah, M. (1992), Performance of the short-time fourier transform and wavelet transform to phonocardiogram signal analysis, *in* 'SAC '92: Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing', ACM, New York, NY, USA, pp. 856–862.

Ogata, K. (1995), *Discrete-Time Control Systems - Second Edition*, Prentice Halll International, Inc., Englewood Cliffs, New Jersey.

Prabhu, C. & Reddi, P. (2006), *Bluetooth Technology and Its Applications with JAVA and J2ME*, Prentice Halll of India, India.

Proakis, J. & Manolakis, D. (1996), *Digital Signal Processing: Principles, Algorithms and Applications, 3rd Edition*, Prentice Halll International, Inc., USA.

Rangayyan, R. M. & Lehner, R. J. (1987), 'Phonocardiogram signal analysis: a review', *Crit Rev Biomed Eng* **1**(15), 211–236.

Scalart, P. & Filho, J. (1996), Speech enhancement based on a priori signal to noise estimation, *in* 'Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on', Vol. 2, pp. 629 –632 vol. 2.

Tilkian, A. & Conover, M. (2001), *Understand Heart Sounds and Murmurs*, W.B. Saunders Company, Philadelphia, Pennsylvania.

Unser, M. & Aldroubi, A. (1996), 'A review of wavelets in biomedical applications', *Proceedings of the IEEE* **84**(4), 626 –638.

Varady, P. (2001), Wavelet-based adaptive denoising of phonocardiographic records, *in* 'Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE', Vol. 2, pp. 1846 – 1849 vol.2.

Vikhe, P., Nehe, N. & Thool, V. (2009), Heart sound abnormality detection using short time fourier transform and continuous wavelet transform, *in* 'Emerging Trends in Engineering and Technology (ICETET), 2009 2nd International Conference on', pp. 50 –54.

Woo, K.-H., Yang, T.-Y., Park, K.-J. & Lee, C. (2000), 'Robust voice activity detection algorithm for estimating noise spectrum', *Electronics Letters* **36**(2), 180 –181.

Young, P. (1999), *Electronic Communication Techniques*, Prentice Halll International, Inc., Upper Saddle River, New Jersey.

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG4111/4112 Research Project
PROJECT SPECIFICATION

FOR: **Justin Sean Neil MILLER**

TOPIC: WIRELESS PHONOCARDIOGRAM AQUISITION SYSTEM

SUPERVISOR: Associate Professor John Leis

PROJECT AIM: To design and implement a wireless phonocardiogram acquisition and analysis system.

PROGRAMME: **(Issue A, 21 March 2010)**

1. Research information relating to the acoustic properties of the human heart.
2. Critically evaluate current methods of acquiring and analysing heart sounds for auscultative diagnosis.
3. Design and implement a phonocardiogram acquisition module to digitise heart sounds captured from an electronic stethoscope.
4. Design and implement a pre-processing filter to: (i) automatically adjust gain; and (ii) remove ambient noise from the signal.
5. Design and implement a wireless transceiver module to transmit heart sounds in real time.
6. Design and implement a phonocardiogram analysis application that shall receive heart sounds from the acquisition module and display a graphical representation of the signal.

As time permits:

7. Design and implement a web service to store heart sounds for remote diagnosis by an off-site practitioner.
8. Design and implement a teleconferencing client, or extend the functionality of an existing teleconferencing client by the means of a plug-in, to stream heart sound data over an internet connection.
9. Evaluate methods used to detect the first heart sound.
10. Implement first heart sound detection and design an algorithm to determine the heart rate.

Appendix B

Source Code

B.1 The testnrg.m Energy VAD Method

The function testnrg.m detects activity in a signal by calculating the energy of the signal and comparing this value to an adaptive threshold.

Listing B.1: Energy VAD Method.

```
clear all;
close all;

%noisyFilename = 'NoisyHeartSounds.wav';
noisyFilename = 'AmbientNoise2.wav';
% noisyFilename = 'NormalMono.wav';
originalFilename = 'NormalMono.wav';

[noisy, Fs] = wavread(noisyFilename);
[original, Fs2] = wavread(originalFilename);

noisy = noisy(1:Fs*4);
original = original(1:length(noisy));

refVoltage = 3.3;
resolution = 2^12;
adc = -1:2/(resolution -1):1;
expectedPeak = resolution / 2;
segLength =256;
numSegments = fix(length(noisy)/segLength);
frame = zeros(1, segLength);
wnd = rectwin(segLength);
silence = Fs * 0.25;
attenuation = 5;
gainSettings = [1, 2, 4, 5, 8, 10, 16, 32];

offset = 1;

clean = zeros(1,length(noisy));
energy = zeros(1,length(noisy));
detectArr = zeros(1, length(noisy));
old = zeros(1, length(noisy));
noise = zeros(1, length(noisy));
instGain = 1;
thisgain = 0;
lastGain = 0;

detector = 0;

E = 0;
Enoise0 = 0;
Enoise1 = 0;

for z = 1:length(clean)
   clean(z) = 0;
end

for s = 1:numSegments

    x = 0;
    for p = offset:segLength*s
      x = x + 1;
      attenuatedSig = noisy(p) / attenuation;
      tempVar = find_closest(attenuatedSig, adc);
```

```
            % Simulate ADC conversion
            frame(x) = find(tempVar == adc);

            % Convert to signed int
            frame(x) = frame(x) - expectedPeak;

        end
        % Calculate energy
        sumEnergy = 0;
        for c = 1:length(frame)
            sumEnergy = sumEnergy + abs(frame(c))^2;
        end

        E = sumEnergy/length(frame);

            Tn = Enoise0 + 180;
            Ts = Enoise0 + 120;

            if (detector == 0) && (E > Ts)
                detector = 1;

            elseif (detector == 1) && (E < Tn)
                detector = 0;
            end

            if (detector == 0)
                Enoise0 = 0.10 * Enoise1 + (1 - 0.999) * E;

                Enoise1 = Enoise0;
                thisgain = 0;

            else
                Enoise0 = 0.60 * Enoise1 + (1 - 0.96) * E;
                Enoise1 = Enoise0;

                thisgain = 5;

            end

        for p = 1:segLength
            old(offset+p-1) = frame(p);
            clean(offset+p-1) = frame(p) * thisgain;
            energy(offset+p-1) = E;
            noise(offset+p-1) = Enoise0;
            detectArr(offset+p-1) = detector;
        end
        offset = segLength*s + 1;

end

xaxis = 0:1/Fs:(length(clean)-1)/Fs;
subplot(4,1,1)
plot(xaxis,old);

title('Input');
xlabel('Time');
ylabel('Amplitude');

subplot(4,1,2)
plot(xaxis,detectArr);

title('Detector');
```

```
xlabel('Time');
ylabel('Detector');
ylim([0, 1.2]);

subplot(4,1,3)
plot(xaxis,noise);

xlabel('Time');
ylabel('Energy');
hold on
% %
plot(xaxis,energy, 'm');

subplot(4,1,4)
%hold on
plot(xaxis,clean);

title('Output');
xlabel('Time');
ylabel('Amplitude');

newSignal = clean / expectedPeak;
sigPowerdB = 10*log10(sum(original.^2)/length(original));
noisePowerdB = 10*log10(sum(noisy.^2)/length(noisy));
cleanPowerdB = 10*log10(sum(newSignal.^2)/length(newSignal));

snr = 10*log10(sum(original.^2) ./ sum(noisy.^2));
snrNew = 10*log10(sum(original.^2) ./ sum(newSignal.^2));
```

B.2 The `testent.m` Entropy VAD Method

The function `testent.m` detects activity in a signal by calculating the entropy of the signal and comparing this value to an adaptive threshold.

Listing B.2: Energy VAD Method.

```
clear all;
close all;

% [noisy, Fs] = wavread('NormalMono.wav');
[noisy, Fs] = wavread('AmbientNoise3.wav');
% [noisy, Fs] = wavread('NoisyHeartSounds.wav');

noisy = noisy(1:Fs*4);

refVoltage = 3.3;
resolution = 2^12;
adc = -1:2/(resolution -1):1;
expectedPeak = resolution / 2;
segLength =64;
numSegments = fix(length(noisy)/segLength);
frame = zeros(1, segLength);

silence = Fs * 0.25;
attenuation = 5;
```

```
gainSettings = [1, 2, 4, 5, 8, 10, 16, 32];

window = hamming(segLength);

offset = 1;

clean = zeros(1,length(noisy));
energy = zeros(1,length(noisy));
detectArr = zeros(1, length(noisy));
old = zeros(1, length(noisy));
noise = zeros(1, length(noisy));
instGain = 1;
thisgain = 0;
lastGain = 0;

detector = 0;

entHist = zeros(1, 8);
ptrHist = 0;

E = 0;
Enoise0 = 0;
Enoise1 = 0.5;

for z = 1:length(clean)
   clean(z) = 0;
end

for s = 1:numSegments

     x = 0;
     for p = offset:segLength*s
       x = x + 1;
       attenuatedSig = noisy(p) / attenuation;
       tempVar = find_closest(attenuatedSig, adc);

       % Simulate ADC conversion
       frame(x) = find(tempVar == adc);

       % Convert to signed int
       frame(x) = frame(x) - expectedPeak;
     end

     F = fft(frame) / length(frame);

     energyFreq = abs(F).^2;
     E = sum(energyFreq);
     PDF = energyFreq / E;

     Entropy = 0;
     for p=1:64
          if (PDF(p) > 0)
               Entropy = Entropy + (PDF(p) * log2(PDF(p)));
          else
               Entropy = Entropy + 0;
          end
     end

     Entropy = Entropy * (-1/log2(64));

     ptrHist = ptrHist + 1;
     if ptrHist > length(entHist)
        ptrHist = 1;
     end
```

```
    entHist(ptrHist) = Entropy;

        Tn = Enoise0 * 0.93;
        Ts = Enoise0 * 0.92;

        filtEnt = mean(entHist);

        if (detector == 0) && (filtEnt< Tn)
            detector = 1;

        elseif (detector == 1) && (filtEnt > Ts)
            detector = 0;
        end

        if (detector == 0)
            Enoise0 = 0.98 * Enoise1 + (1 - 0.98) * Entropy;

            Enoise1 = Enoise0;
            thisgain = 0;

        else
%             Enoise0 = 0.94 * Enoise1 + (1 - 0.92) * Entropy;

            Enoise1 = Enoise0;
            thisgain = 1;
        end

    for p = 1:segLength
        old(offset+p-1) = frame(p);
        clean(offset+p-1) = frame(p) * thisgain;
        energy(offset+p-1) = Enoise0;
        noise(offset+p-1) = filtEnt;
        detectArr(offset+p-1) = detector;
    end
    offset = segLength*s + 1;

end

xaxis = 0:1/Fs:(length(clean)-1)/Fs;
subplot(4,1,1)
plot(xaxis,old);

title('Input');
xlabel('Time');
ylabel('Amplitude');

subplot(4,1,2)
plot(xaxis,detectArr);

title('Detector');
xlabel('Time');
ylabel('Detector');
ylim([0, 1.2]);

subplot(4,1,3)
plot(xaxis,noise);

xlabel('Time');
ylabel('Entropy');
hold on
%%
plot(xaxis,energy, 'm');
```

```
subplot(4,1,4)
%hold on
plot(xaxis,clean);

title('Output');
xlabel('Time');
ylabel('Amplitude');

newSignal = clean / expectedPeak;
```

B.3 The main.c Firmware - Main Source File

The file `main.c` is where most of the functionality of the firmware resides.

Listing B.3: Firmware - Main Source File.

```
#include "main.h"
#include "uLaw.h"

_FOSCSEL(FNOSC_FRCPLL);
_FOSC(FCKSM_CSECMD & OSCIOFNC_ON   & POSCMD_NONE);
_FICD(JTAGEN_OFF & ICS_PGD2);
_FWDT(FWDTEN_OFF);

int main(void)
{

    init_pll();

        init_gpio();
        init_cn();
        init_timer1();
        init_spi();
        init_uart();
        open_uart();

        LATAbits.LATA3 = 1; // Disable ADC
    LATAbits.LATA4 = 1; // Disable PGA

        SetGain(4);                     // Set unity gain to begin with

    while(1)
        {
                LATAbits.LATA2 = active;
                if (buttonPress)
                {
                   DebounceSwitch();
                }

                if (pollADC)
                {
                        Read_ADC();
                        pollADC = ~pollADC;
                }


                if (symbolCount >= BUFFER_SIZE)
                {
                        SendBuffer();
                }
    }

        return 0;
}

void init_pll(void)
{
        // Setup internal clock for 80MHz/40MIPS
        //7.37/2=3.685*43=158.455/2=79.2275
        CLKDIVbits.PLLPRE=0; // PLLPRE (N2) 0=/2
```

```
        PLLFBD=41; //pll multiplier (M) = +2
        CLKDIVbits.PLLPOST=0;// PLLPOST (N1) 0=/2

        // Wait until the PLL is ready
        while(!OSCCONbits.LOCK);
}

void init_gpio(void)
{
        AD1PCFGL = 0xffff; //All analog capable pins in digital mode

        TRISAbits.TRISA2 = 0;    // LED output
        TRISBbits.TRISB0 = 1;    // Switch input
}

void init_cn(void)
{
        CNEN1bits.CN4IE = 1; // Enable CN4 (RB0) pin for interrupt detecti
        CNPU1bits.CN4PUE = 0; // Disable CN4 pull-up
        IEC1bits.CNIE = 1; // Enable CN interrupts
        IFS1bits.CNIF = 0; // Reset CN interrupt
        IPC4bits.CNIP = 3; // Set CN interrupt priority
}

void init_uart(void)
{
        RPINR18bits.U1RXR = 7;   // RX
        RPINR18bits.U1CTSR = 0;  // CTS
        RPOR2bits.RP4R = 3;      // TX;
        RPOR4bits.RP8R = 4;              // RTS

        // Setup UART
    U1BRG = 85;//86@80mhz, 85@79.xxx=115200
    U1MODE = 0; //clear mode register
        U1MODEbits.STSEL = 0; // 1-stop bit
        U1MODEbits.PDSEL = 0; // No Parity, 8-data bits
        U1MODEbits.ABAUD = 0; // Auto-Baud Disabled
        U1MODEbits.BRGH = 0;  // Low Speed mode
        U1MODEbits.BRGH = 1;  //use high precison baud generator

    U1STA = 0;  //clear status register
    IFS0bits.U1RXIF = 0;  //clear the receive flag
}

void init_spi(void)
{
    SPI1STAT = 0;                                        // Initialise SPI1Sta
    SPI1CON1 = 0;                                         // Initialise SPI1Co

        TRISAbits.TRISA3 = 0;    // CS for ADC
        TRISAbits.TRISA4 = 0;    // CS for PGA
        TRISBbits.TRISB9 = 0;    // CLK output
        TRISBbits.TRISB15 = 1;   // DI Input
        TRISBbits.TRISB14 = 0;   // DO Output

        RPOR4bits.RP9R = 8;      // Clock
        RPINR20bits.SDI1R = 15;                   // SDI input
        RPOR7bits.RP14R = 7;                               // SDO output
```

```
        SPI1STAT = 0b00000000; // Master sample data in middle, data xmt o
                                                // rising edge
//      SPI1CON1 = 0b00110000; // enable Master SPI, bus mode 1,1, FOSC/4
        SPI1CON1 = 0b101100000; // enable Master SPI, bus mode 1,1, FOSC/8

        SPI1STATbits.SPIROV = 0;                 // Clear SPIROV (SPI recie
        SPI1STATbits.SPIEN = 1;                  // Enable SPI
}

void init_timer1(void)
{
        T1CONbits.TON = 0; // Disable Timer
        T1CONbits.TCKPS = 0b10; // Select the PRESCALER
        T1CONbits.TCS = 0; // Internal clock
        T1CONbits.TGATE = 0; // Disable Gated Timer mode
        TMR1 = 0x00; // Make sure the timer is starting from zero
        PR1 = 61; // How long the timer should run before an interrupt (in

        IFS0bits.T1IF = 0; // Clear Timer1 Interrupt Flag
        IEC0bits.T1IE = 1; // Enable Timer1 interrupt

        T1CONbits.TON = 1; // Turn the interrupt on
}

void open_uart(void)
{
        U1MODEbits.UARTEN = 1;    // UART1 enabled
        U1STAbits.UTXEN = 1;        // UARTx transmitter enabled
}

void SendUART(unsigned char c)
{
     while(U1STAbits.UTXBF != 0);
     U1TXREG = c;
        while(U1STAbits.TRMT == 0);
}

void __attribute__ ((interrupt , no_auto_psv)) _CNInterrupt(void)
{
   // Insert ISR code here
   IFS1bits.CNIF = 0; // Clear CN interrupt

   buttonPress = ~buttonPress;
}

void __attribute__((__interrupt__)) __attribute__((no_auto_psv)) _T1Interr
{
        IFS0bits.T1IF = 0; // Clear Timer1 Interrupt Flag
        pollADC = active;
}

void Read_ADC()
{
        ADC_DATA adc;
        int value;

        SPI1STATbits.SPIROV = 0;
        LATAbits.LATA3 = 0; // Enable ADC

        // MSB
```

```
        SPI1BUF = 0x01;
        while (!SPI1STATbits.SPIRBF);
        adc.databyte[1] = SPI1BUF;

        // LSB
        SPI1BUF = 0x81;
        while (!SPI1STATbits.SPIRBF);
        LATAbits.LATA3 = 1; // Disable ADC
        adc.databyte[0] = SPI1BUF;

        adc.result >>= 1; // adjust composite integer for 12 valid bits
        adc.result &= 0x0FFF; // mask out upper nibble of integer

        // Scale to 16 bits
        adc.result = (adc.result << 4) ^ (adc.result >> 8);

        value = adc.result - 0x7FFF;

        buffer[symbolCount] = linear2ulaw(value);
        symbolCount++;
}

void SendBuffer()
{
        int i;

        for (i = 0; i < BUFFER_SIZE; i++)
        {
                if (buffer[i] == DLE)
                        SendUART(DLE);
                SendUART(buffer[i]);
        }

        symbolCount = 0;

}

void DebounceSwitch(void)
{
        unsigned char Switch_Count = 0;
        int i, j;

        // Disable interrupt
        CNEN1bits.CN4IE = 0;

        // Monitor switch input for 5 lows in a row to debounce
        for (i=0; i<10; i++)
        {
            if (PORTBbits.RB0 != 0)
            {
                Switch_Count++; // Pressed state detected
            }
                else
                {
                        Switch_Count = 0;
                }

                // Short delay
            for (j=0;j < 20000; j++)
                {
                }
```

```
        }

        // If the switch
        if (Switch_Count > 5)
        {
                active = ~active;
        }

        buttonPress = 0;

        // Re-enable interrupt
        CNEN1bits.CN4IE = 1;
}

void send_byte(unsigned char data) {
   int count;
   int i;

   for (count=0;count<8;count++)
   {

        if (data & 0x80)
         LATBbits.LATB14 = 1;
       else
         LATBbits.LATB14 = 0;

          LATBbits.LATB9 = 1;

          for (i =0; i< 50; i++);

          LATBbits.LATB9 = 0;
          data <<=1;

   }
}

void SetGain(int gain)
{
        int setting;

        // Translate the required gain into a valid PGA setting
        switch (gain)
        {
                case 1:
                        setting = 0b000;
                        break;
                case 2:
                        setting = 0b001;
                        break;
                case 4:
                        setting = 0b010;
                        break;
                case 5:
                        setting = 0b011;
                        break;
                case 8:
                        setting = 0b100;
                        break;
                case 10:
                        setting = 0b101;
                        break;
                case 16:
                        setting = 0b110;
```

```
                    break;
                case 32:
                    setting = 0b111;
                    break;
                default:
                    setting = 0;
        }

        SPI1STATbits.SPIEN = 0; // Disable SPI

        T1CONbits.TON = 0;              // Turn the interrupt off

    LATBbits.LATB9 = 0;
        LATAbits.LATA4 = 0;     // Enable PGA Chipselect

        send_byte(0x40);                // Send instruction
        send_byte(setting);             // Send setting

        LATAbits.LATA4 = 1;     // Disable PGA Chipselect

        T1CONbits.TON = 1;              // Turn the interrupt on
        SPI1STATbits.SPIEN = 1; // Re-enable SPI
}
```

B.4 The main.h Firmware - Main Header File

The file main.h is where the function declarations and global variables used by main.c can be found.

Listing B.4: Firmware - Main Header File.

```
#include <p33FJ12GP201.h>
#include <dsp.h>
#include "fft.h"
//#include "twiddleFactors.c"

#define DLE 0x10
#define STX 0x02
#define ETX 0x03
#define BUFFER_SIZE 64

unsigned char active = 0;
unsigned char buttonPress = 0;
unsigned char pollADC = 0;

typedef union
{
        char databyte[2];       // declare temp array for adc data
        unsigned int result; // declare integer for adc result
} ADC_DATA; // define union variable


/* Extern definitions */
extern fractcomplex sigCmpx[FFT_BLOCK_LENGTH]          /* Typically, the
*/
__attribute__ ((section (".ydata, data, ymemory"),      /* routine is a c
```

```
aligned (FFT_BLOCK_LENGTH * 2 *2)));
/* of an input signal. */

extern const fractcomplex twiddleFactors[FFT_BLOCK_LENGTH/2]    /* Twiddl
__attribute__ ((space(auto_psv), aligned (FFT_BLOCK_LENGTH*2)));

unsigned char buffer[BUFFER_SIZE];
unsigned char symbolCount = 0;

void init_pll(void);
void init_gpio(void);
void init_cn(void);
void init_uart(void);
void init_timer1(void);
void open_uart(void);
void init_spi(void);
void SendUART(unsigned char);
void Read_ADC();
void SendBuffer();
void DebounceSwitch(void);
void SetGain(int);
void __attribute__ ((interrupt , no_auto_psv)) _CNInterrupt(void);
void __attribute__((__interrupt__)) __attribute__((no_auto_psv)) _T1Interr
```

B.5 The ulaw.c Firmware - uLaw Source File

Listing B.5: Firmware - uLaw Function.

```
#include "uLaw.h"

/*
** This routine converts from linear to ulaw
**
** Craig Reese: IDA/Supercomputing Research Center
** Joe Campbell: Department of Defense
** 29 September 1989
**
** References:
** 1) CCITT Recommendation G.711  (very difficult to follow)
** 2) "A New Digital Technique for Implementation of Any
**     Continuous PCM Companding Law," Villeret, Michel,
**     et al. 1973 IEEE Int. Conf. on Communications, Vol 1,
**     1973, pg. 11.12-11.17
** 3) MIL-STD-188-113,"Interoperability and Performance Standards
**     for Analog-to_Digital Conversion Techniques,"
**     17 February 1987
**
** Input: Signed 16 bit linear sample
** Output: 8 bit ulaw sample
*/

#define ZEROTRAP    /* turn on the trap as per the MIL-STD */
#define BIAS 0x84   /* define the add-in bias for 16 bit samples */
#define CLIP 32635

unsigned char
linear2ulaw(sample)
```

```
int sample; {
  static int exp_lut[256] = {0,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3,
                             4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
                             5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
                             5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
                             6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                             6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                             6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                             6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                             7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7};
  int sign, exponent, mantissa;
  unsigned char ulawbyte;

  /* Get the sample into sign-magnitude. */
  sign = (sample >> 8) & 0x80;          /* set aside the sign */
  if (sign != 0) sample = -sample;              /* get magnitude */
  if (sample > CLIP) sample = CLIP;             /* clip the magnitude */

  /* Convert from 16 bit linear to ulaw. */
  sample = sample + BIAS;
  exponent = exp_lut[(sample >> 7) & 0xFF];
  mantissa = (sample >> (exponent + 3)) & 0x0F;
  ulawbyte = ~(sign | (exponent << 4) | mantissa);
#ifdef ZEROTRAP
  if (ulawbyte == 0) ulawbyte = 0x02;   /* optional CCITT trap */
#endif

  return(ulawbyte);
}
```

B.6 The ulaw.h Firmware - uLaw Header File

Listing B.6: Firmware - uLaw Header File.

```
unsigned char linear2ulaw(int);
```

B.7 The twiddleFactors.c Firmware - Twiddle Factors

Listing B.7: Firmware - Twiddle Factors.

```
/*****************************************************************
*   2005 Microchip Technology Inc.
*
* FileName:          twiddleFactors.c
* Dependencies:      Header (.h) files if applicable, see below
* Processor:         dsPIC30Fxxxx
* Compiler:          MPLAB C30 v3.00 or higher
* IDE:               MPLAB IDE v7.52 or later
* Dev. Board Used: dsPICDEM 1.1 Development Board
* Hardware Dependencies: None
*
* SOFTWARE LICENSE AGREEMENT:
* Microchip Technology Incorporated ("Microchip") retains all ownership an
* intellectual property rights in the code accompanying this message and
```

```
* derivatives hereto. You may use this code, and any derivatives created
* any person or entity by or on your behalf, exclusively with Microchip's
* proprietary products. Your acceptance and/or use of this code constitu
* agreement to the terms and conditions of this notice.
*
* CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS".
NO
* WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LI
* TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS
* PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH MICROCHIP'S
* PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION
*
* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE LIABLE,
* IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF STATUTORY
* STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY INDIREC
* PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR COST
* ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF MICRO
* ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLE
* ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY R
* THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPE
* HAVE THIS CODE DEVELOPED.
*
* You agree that you are solely responsible for testing the code and
* determining its suitability. Microchip has no obligation to modify, tes
* certify, or support the code.
* ADDITIONAL NOTES:
*
*
*****************************************************************/

#include <dsp.h>
#include "fft.h"

#ifdef FFTTWIDCOEFFS_IN_PROGMEM

#if (FFT_BLOCK_LENGTH == 64)
        const fractcomplex twiddleFactors[] __attribute__ ((space(auto_psv
        {
        0x7FFF, 0x0000, 0x7F62, 0xF374, 0x7D8A, 0xE707, 0x7A7D, 0xDAD8,
        0x7642, 0xCF04, 0x70E3, 0xC3A9, 0x6A6E, 0xB8E3, 0x62F2, 0xAECC,
        0x5A82, 0xA57E, 0x5134, 0x9D0E, 0x471D, 0x9592, 0x3C57, 0x8F1D,
        0x30FC, 0x89BE, 0x2528, 0x8583, 0x18F9, 0x8276, 0x0C8C, 0x809E,
        0x0000, 0x8000, 0xF374, 0x809E, 0xE707, 0x8276, 0xDAD8, 0x8583,
        0xCF04, 0x89BE, 0xC3A9, 0x8F1D, 0xB8E3, 0x9592, 0xAECC, 0x9D0E,
        0xA57D, 0xA57D, 0x9D0E, 0xAECC, 0x9592, 0xB8E3, 0x8F1D, 0xC3A9,
        0x89BE, 0xCF04, 0x8583, 0xDAD8, 0x8276, 0xE707, 0x809E, 0xF374
        } ;
#endif
#if (FFT_BLOCK_LENGTH == 128)
        const fractcomplex twiddleFactors[] __attribute__ ((space(auto_psv
        {
        0x7FFF, 0x0000, 0x7FD9, 0xF9B8, 0x7F62, 0xF374, 0x7E9D, 0xED38,
        0x7D8A, 0xE707, 0x7C2A, 0xE0E6, 0x7A7D, 0xDAD8, 0x7885, 0xD4E1,
        0x7642, 0xCF04, 0x73B6, 0xC946, 0x70E3, 0xC3A9, 0x6DCA, 0xBE32,
        0x6A6E, 0xB8E3, 0x66D0, 0xB3C0, 0x62F2, 0xAECC, 0x5ED7, 0xAA0A,
        0x5A82, 0xA57E, 0x55F6, 0xA129, 0x5134, 0x9D0E, 0x4C40, 0x9930,
        0x471D, 0x9592, 0x41CE, 0x9236, 0x3C57, 0x8F1D, 0x36BA, 0x8C4A,
        0x30FC, 0x89BE, 0x2B1F, 0x877B, 0x2528, 0x8583, 0x1F1A, 0x83D6,
        0x18F9, 0x8276, 0x12C8, 0x8163, 0x0C8C, 0x809E, 0x0648, 0x8027,
        0x0000, 0x8000, 0xF9B8, 0x8027, 0xF374, 0x809E, 0xED38, 0x8163,
        0xE707, 0x8276, 0xE0E6, 0x83D6, 0xDAD8, 0x8583, 0xD4E1, 0x877C,
        0xCF04, 0x89BE, 0xC946, 0x8C4A, 0xC3A9, 0x8F1D, 0xBE32, 0x9236,
        0xB8E3, 0x9592, 0xB3C0, 0x9931, 0xAECC, 0x9D0E, 0xAA0A, 0xA129,
```

```
        0xA57E, 0xA57E, 0xA129, 0xAA0A, 0x9D0E, 0xAECC, 0x9931, 0xB3C0,
        0x9592, 0xB8E3, 0x9236, 0xBE32, 0x8F1D, 0xC3A9, 0x8C4A, 0xC946,
        0x89BE, 0xCF04, 0x877C, 0xD4E1, 0x8583, 0xDAD8, 0x83D6, 0xE0E6,
        0x8276, 0xE707, 0x8163, 0xED38, 0x809E, 0xF374, 0x8027, 0xF9B8
        };
#endif
#if (FFT_BLOCK_LENGTH == 256)
        const fractcomplex twiddleFactors[] __attribute__ ((space(auto_psv
        {
        0x7FFF, 0x0000, 0x7FF6, 0xFCDC, 0x7FD9, 0xF9B8, 0x7FA7, 0xF695,
        0x7F62, 0xF374, 0x7F0A, 0xF055, 0x7E9D, 0xED38, 0x7E1E, 0xEA1E,
        0x7D8A, 0xE707, 0x7CE4, 0xE3F4, 0x7C2A, 0xE0E6, 0x7B5D, 0xDDDC,
        0x7A7D, 0xDAD8, 0x798A, 0xD7D9, 0x7884, 0xD4E1, 0x776C, 0xD1EF,
        0x7642, 0xCF04, 0x7505, 0xCC21, 0x73B6, 0xC946, 0x7255, 0xC673,
        0x70E3, 0xC3A9, 0x6F5F, 0xC0E9, 0x6DCA, 0xBE32, 0x6C24, 0xBB85,
        0x6A6E, 0xB8E3, 0x68A7, 0xB64C, 0x66CF, 0xB3C0, 0x64E8, 0xB140,
        0x62F2, 0xAECC, 0x60EC, 0xAC65, 0x5ED7, 0xAA0A, 0x5CB4, 0xA7BD,
        0x5A82, 0xA57E, 0x5843, 0xA34C, 0x55F6, 0xA129, 0x539B, 0x9F14,
        0x5134, 0x9D0E, 0x4EC0, 0x9B18, 0x4C40, 0x9931, 0x49B4, 0x9759,
        0x471D, 0x9592, 0x447B, 0x93DC, 0x41CE, 0x9236, 0x3F17, 0x90A1,
        0x3C57, 0x8F1D, 0x398D, 0x8DAB, 0x36BA, 0x8C4A, 0x33DF, 0x8AFB,
        0x30FC, 0x89BE, 0x2E11, 0x8894, 0x2B1F, 0x877C, 0x2827, 0x8676,
        0x2528, 0x8583, 0x2224, 0x84A3, 0x1F1A, 0x83D6, 0x1C0B, 0x831C,
        0x18F9, 0x8276, 0x15E2, 0x81E3, 0x12C8, 0x8163, 0x0FAB, 0x80F7,
        0x0C8C, 0x809E, 0x096B, 0x8059, 0x0648, 0x8028, 0x0324, 0x800A,
        0x0000, 0x8000, 0xFCDC, 0x800A, 0xF9B8, 0x8028, 0xF695, 0x8059,
        0xF374, 0x809E, 0xF055, 0x80F7, 0xED38, 0x8163, 0xEA1E, 0x81E3,
        0xE707, 0x8276, 0xE3F5, 0x831C, 0xE0E6, 0x83D6, 0xDDDC, 0x84A3,
        0xDAD8, 0x8583, 0xD7D9, 0x8676, 0xD4E1, 0x877C, 0xD1EF, 0x8894,
        0xCF04, 0x89BE, 0xCC21, 0x8AFB, 0xC946, 0x8C4A, 0xC673, 0x8DAB,
        0xC3A9, 0x8F1D, 0xC0E9, 0x90A1, 0xBE32, 0x9236, 0xBB85, 0x93DC,
        0xB8E3, 0x9593, 0xB64C, 0x975A, 0xB3C0, 0x9931, 0xB140, 0x9B18,
        0xAECC, 0x9D0E, 0xAC65, 0x9F14, 0xAA0A, 0xA129, 0xA7BD, 0xA34C,
        0xA57E, 0xA57E, 0xA34C, 0xA7BD, 0xA129, 0xAA0A, 0x9F14, 0xAC65,
        0x9D0E, 0xAECC, 0x9B18, 0xB140, 0x9931, 0xB3C0, 0x975A, 0xB64C,
        0x9593, 0xB8E3, 0x93DC, 0xBB85, 0x9236, 0xBE32, 0x90A1, 0xC0E9,
        0x8F1D, 0xC3A9, 0x8DAB, 0xC673, 0x8C4A, 0xC946, 0x8AFB, 0xCC21,
        0x89BF, 0xCF04, 0x8894, 0xD1EF, 0x877C, 0xD4E1, 0x8676, 0xD7D9,
        0x8583, 0xDAD8, 0x84A3, 0xDDDC, 0x83D6, 0xE0E6, 0x831C, 0xE3F5,
        0x8276, 0xE707, 0x81E3, 0xEA1E, 0x8163, 0xED38, 0x80F7, 0xF055,
        0x809E, 0xF374, 0x8059, 0xF695, 0x8028, 0xF9B8, 0x800A, 0xFCDC
        };
#endif
#if (FFT_BLOCK_LENGTH == 512 )
        const fractcomplex twiddleFactors[] __attribute__ ((space(auto_psv
        {
        0x7FFF, 0x0000, 0x7FFE, 0xFE6E, 0x7FF6, 0xFCDC, 0x7FEA, 0xFB4A,
        0x7FD9, 0xF9B8, 0x7FC2, 0xF827, 0x7FA7, 0xF695, 0x7F87, 0xF505,
        0x7F62, 0xF374, 0x7F38, 0xF1E4, 0x7F0A, 0xF055, 0x7ED6, 0xEEC6,
        0x7E9D, 0xED38, 0x7E60, 0xEBAB, 0x7E1E, 0xEA1E, 0x7DD6, 0xE892,
        0x7D8A, 0xE707, 0x7D3A, 0xE57D, 0x7CE4, 0xE3F4, 0x7C89, 0xE26D,
        0x7C2A, 0xE0E6, 0x7BC6, 0xDF61, 0x7B5D, 0xDDDC, 0x7AEF, 0xDC59,
        0x7A7D, 0xDAD8, 0x7A06, 0xD958, 0x798A, 0xD7D9, 0x790A, 0xD65C,
        0x7885, 0xD4E1, 0x77FB, 0xD367, 0x776C, 0xD1EF, 0x76D9, 0xD079,
        0x7642, 0xCF04, 0x75A6, 0xCD92, 0x7505, 0xCC21, 0x7460, 0xCAB2,
        0x73B6, 0xC946, 0x7308, 0xC7DB, 0x7255, 0xC673, 0x719E, 0xC50D,
        0x70E3, 0xC3A9, 0x7023, 0xC248, 0x6F5F, 0xC0E9, 0x6E97, 0xBF8C,
        0x6DCA, 0xBE32, 0x6CF9, 0xBCDA, 0x6C24, 0xBB85, 0x6B4B, 0xBA33,
        0x6A6E, 0xB8E3, 0x698C, 0xB796, 0x68A7, 0xB64C, 0x67BD, 0xB505,
        0x66D0, 0xB3C0, 0x65DE, 0xB27F, 0x64E9, 0xB140, 0x63EF, 0xB005,
        0x62F2, 0xAECC, 0x61F1, 0xAD97, 0x60EC, 0xAC65, 0x5FE4, 0xAB36,
```

```
        0x5ED8, 0xAA0A, 0x5DC8, 0xA8E2, 0x5CB4, 0xA7BD, 0x5B9D, 0xA69C,
        0x5A83, 0xA57E, 0x5964, 0xA463, 0x5843, 0xA34C, 0x571E, 0xA238,
        0x55F6, 0xA128, 0x54CA, 0xA01C, 0x539B, 0x9F14, 0x5269, 0x9E0F,
        0x5134, 0x9D0E, 0x4FFB, 0x9C11, 0x4EC0, 0x9B17, 0x4D81, 0x9A22,
        0x4C40, 0x9930, 0x4AFB, 0x9843, 0x49B4, 0x9759, 0x486A, 0x9674,
        0x471D, 0x9592, 0x45CD, 0x94B5, 0x447B, 0x93DC, 0x4326, 0x9307,
        0x41CE, 0x9236, 0x4074, 0x9169, 0x3F17, 0x90A1, 0x3DB8, 0x8FDD,
        0x3C57, 0x8F1D, 0x3AF3, 0x8E62, 0x398D, 0x8DAB, 0x3825, 0x8CF8,
        0x36BA, 0x8C4A, 0x354E, 0x8BA0, 0x33DF, 0x8AFB, 0x326E, 0x8A5A,
        0x30FC, 0x89BE, 0x2F87, 0x8927, 0x2E11, 0x8894, 0x2C99, 0x8805,
        0x2B1F, 0x877B, 0x29A4, 0x86F6, 0x2827, 0x8676, 0x26A8, 0x85FA,
        0x2528, 0x8583, 0x23A7, 0x8511, 0x2224, 0x84A3, 0x209F, 0x843A,
        0x1F1A, 0x83D6, 0x1D93, 0x8377, 0x1C0C, 0x831C, 0x1A83, 0x82C6,
        0x18F9, 0x8276, 0x176E, 0x822A, 0x15E2, 0x81E2, 0x1455, 0x81A0,
        0x12C8, 0x8163, 0x113A, 0x812A, 0x0FAB, 0x80F6, 0x0E1C, 0x80C8,
        0x0C8C, 0x809E, 0x0AFB, 0x8079, 0x096B, 0x8059, 0x07D9, 0x803E,
        0x0648, 0x8027, 0x04B6, 0x8016, 0x0324, 0x800A, 0x0192, 0x8002,
        0x0000, 0x8000, 0xFE6E, 0x8002, 0xFCDC, 0x800A, 0xFB4A, 0x8016,
        0xF9B8, 0x8027, 0xF827, 0x803E, 0xF695, 0x8059, 0xF505, 0x8079,
        0xF374, 0x809E, 0xF1E4, 0x80C8, 0xF055, 0x80F6, 0xEEC6, 0x812A,
        0xED38, 0x8163, 0xEBAB, 0x81A0, 0xEA1E, 0x81E2, 0xE892, 0x822A,
        0xE707, 0x8276, 0xE57D, 0x82C6, 0xE3F4, 0x831C, 0xE26D, 0x8377,
        0xE0E6, 0x83D6, 0xDF61, 0x843A, 0xDDDC, 0x84A3, 0xDC59, 0x8511,
        0xDAD8, 0x8583, 0xD958, 0x85FA, 0xD7D9, 0x8676, 0xD65C, 0x86F6,
        0xD4E1, 0x877B, 0xD367, 0x8805, 0xD1EF, 0x8894, 0xD079, 0x8927,
        0xCF04, 0x89BE, 0xCD92, 0x8A5A, 0xCC21, 0x8AFB, 0xCAB2, 0x8BA0,
        0xC946, 0x8C4A, 0xC7DB, 0x8CF8, 0xC673, 0x8DAB, 0xC50D, 0x8E62,
        0xC3A9, 0x8F1D, 0xC248, 0x8FDD, 0xC0E9, 0x90A1, 0xBF8C, 0x9169,
        0xBE32, 0x9236, 0xBCDA, 0x9307, 0xBB85, 0x93DC, 0xBA33, 0x94B5,
        0xB8E3, 0x9592, 0xB796, 0x9674, 0xB64C, 0x9759, 0xB505, 0x9843,
        0xB3C0, 0x9930, 0xB27F, 0x9A22, 0xB140, 0x9B17, 0xB005, 0x9C11,
        0xAECC, 0x9D0E, 0xAD97, 0x9E0F, 0xAC65, 0x9F14, 0xAB36, 0xA01C,
        0xAA0A, 0xA128, 0xA8E2, 0xA238, 0xA7BD, 0xA34C, 0xA69C, 0xA463,
        0xA57D, 0xA57D, 0xA463, 0xA69C, 0xA34C, 0xA7BD, 0xA238, 0xA8E2,
        0xA128, 0xAA0A, 0xA01C, 0xAB36, 0x9F14, 0xAC65, 0x9E0F, 0xAD97,
        0x9D0E, 0xAECC, 0x9C11, 0xB005, 0x9B17, 0xB140, 0x9A22, 0xB27F,
        0x9930, 0xB3C0, 0x9843, 0xB504, 0x9759, 0xB64C, 0x9674, 0xB796,
        0x9592, 0xB8E3, 0x94B5, 0xBA33, 0x93DC, 0xBB85, 0x9307, 0xBCDA,
        0x9236, 0xBE32, 0x9169, 0xBF8C, 0x90A1, 0xC0E9, 0x8FDD, 0xC248,
        0x8F1D, 0xC3A9, 0x8E62, 0xC50D, 0x8DAB, 0xC673, 0x8CF8, 0xC7DB,
        0x8C4A, 0xC946, 0x8BA0, 0xCAB2, 0x8AFB, 0xCC21, 0x8A5A, 0xCD92,
        0x89BE, 0xCF04, 0x8927, 0xD079, 0x8894, 0xD1EF, 0x8805, 0xD367,
        0x877B, 0xD4E1, 0x86F6, 0xD65C, 0x8676, 0xD7D9, 0x85FA, 0xD958,
        0x8583, 0xDAD8, 0x8510, 0xDC59, 0x84A3, 0xDDDC, 0x843A, 0xDF61,
        0x83D6, 0xE0E6, 0x8377, 0xE26D, 0x831C, 0xE3F4, 0x82C6, 0xE57D,
        0x8275, 0xE707, 0x8229, 0xE892, 0x81E2, 0xEA1E, 0x81A0, 0xEBAB,
        0x8163, 0xED38, 0x812A, 0xEEC6, 0x80F6, 0xF055, 0x80C8, 0xF1E4,
        0x809E, 0xF374, 0x8079, 0xF505, 0x8059, 0xF695, 0x803E, 0xF827,
        0x8027, 0xF9B8, 0x8016, 0xFB4A, 0x800A, 0xFCDC, 0x8002, 0xFE6E
        };
#endif

#endif
```

B.8 The fft.h Firmware - FFT Declarations

Listing B.8: Firmware - FFT Declarations.

```
/* Constant Definitions */
#define FFT_BLOCK_LENGTH        64      /* = Number of frequency points in
#define LOG2_BLOCK_LENGTH       8       /* = Number of "Butterfly" Stages
#define SAMPLING_RATE           8000    /* = Rate at which input signal w
```

```
/* SAMPLING_RATE is used to calcu
/* of the largest element in the I

#define FFTTWIDCOEFFS_IN_PROGMEM
```

B.9 The FormMain.cs Phonocardiogram - Main Form

FormMain.cs is the main menu of the Windows Phonocardiom Host. The main menu also graphs and plays back the ascultation signal in real time.

Listing B.9: Phonocardiogram - Main Form.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;
using SlimDX.DirectSound;
using SlimDX.Multimedia;
using MathNet.Numerics;
using MathNet.Numerics.IntegralTransforms;

namespace Phonocardiogram
{
    public partial class FormMain : Form
    {
        static short[] seg_end = {0xFF, 0x1FF, 0x3FF, 0x7FF,
                                  0xFFF, 0x1FFF, 0x3FFF, 0x7FFF};

        double lastUpdate = 0.0;
        const int SamplesPerSecond = 8000;

        static Queue<InData> _buffer = new Queue<InData>();
        static CircularBuffer _buffer2 = new CircularBuffer();
        private double[] dwtArray = new double[128];

        delegate void AddTimeDomainDataCallback(byte[] buffer, int len);

        public FormMain()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                timerRefresh.Enabled = true;
                serialPortBT.Open();

                InitGraph();

                InitSound();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void InitGraph()
        {
```

```
            scopeTime.Channels[0].Data.SampleRate = SamplesPerSecond;
            waterfallDWT.XAxis.Samples = SamplesPerSecond * 3 / 258;
            waterfallDWT.XAxis.MaxTick.AutoScale = false;
        }

        private void UpdateScalogram(double[] frame)
        {
            daub dwt = new daub();
            dwt.daubTrans(frame);

            for (int i = 0; i < frame.Length; i++)
                frame[i] = Math.Abs(frame[i] / frame.Length);

            waterfallDWT.Data.AddData(frame);
        }

        private void UpdateSpectogram(double[] frame)
        {
            Complex[] data = new Complex[frame.Length];
            for (int j = 0; j < frame.Length; j++)
                data[j] = frame[j];

            var fft = new MathNet.Numerics.IntegralTransforms.Algorithms.I
            fft.Radix2Forward(data, FourierOptions.Matlab);

            int sampleRate = 4000;
            double res = (double)(data.Length / 2) / sampleRate;

            res = 1 / res;

            double[] fftFrame = new double[data.Length / 2];
            for (int j = 0; j < data.Length / 2; j++)
                fftFrame[j] = Math.Abs(data[j].Real) / (frame.Length /2 );

            waterfallDWT.Data.AddData(fftFrame);
        }

        private void serialPortBT_DataReceived(object sender, System.IO.P
        {
            SerialPort sp = (SerialPort)sender;

            byte[] buffer = new byte[256];
            int dataReceived = sp.Read(buffer, 0, buffer.Length);

            try
            {
                AddTimeDomainData(buffer, dataReceived);
            }
            catch
            {
                // TODO: Handle this error
            }
        }


        int ulaw2linear(byte ulawbyte)
        {
            int[] exp_lut = { 0, 132, 396, 924, 1980, 4092, 8316, 16764 }
            int sign, exponent, mantissa, sample;

            ulawbyte = (byte)~ulawbyte;
```

```
            sign = (ulawbyte & 0x80);
            exponent = (ulawbyte >> 4) & 0x07;
            mantissa = ulawbyte & 0x0F;
            sample = exp_lut[exponent] + (mantissa << (exponent + 3));
            if(sign != 0) sample = -sample;

            return(sample);
        }

        private void AddTimeDomainData(byte[] buffer, int len)
        {
            lock (_buffer)
            {

                for (int i = 0; i < len; i++)
                {
                    InData data = new InData();
                    data.Value = buffer[i];
                    data.TimeStamp = DateTime.Now;
                    lastUpdate = lastUpdate + (1.0 / 5000);
                    data.LastUpdate = lastUpdate;
                    _buffer.Enqueue(data);
                    _buffer2.Add(buffer[i]);
                }
            }
        }

        private void CloseSerialPort(Object stateInfo)
        {
            serialPortBT.Close();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            timerRefresh.Enabled = false;
            ThreadPool.QueueUserWorkItem(new WaitCallback(CloseSerialPort
        }

        private void timerRefresh_Tick(object sender, EventArgs e)
        {
            UpdateGraphs();
        }

        private void UpdateGraphs()
        {
            double[] frame = new double[256];
            int element = 0;

            int i = 0;
            while (_buffer.Count != 0 && i < 10)
            {
                InData data = _buffer.Dequeue();
                if (data != null)
                {
                    double pcmSample = (double)ulaw2linear(data.Value);
                    scopeTime.Channels[0].Data.AddYPoint(pcmSample);
                    i++;

                    frame[element] = pcmSample;
```

```
                    element++;
                    if (element > frame.Length - 1)
                    {
                        UpdateSpectogram(frame);
                        element = 0;
                    }
                }
            }
        }

        private void InitSound()
        {
            DirectSound ds = new DirectSound(DirectSoundGuid.DefaultPlayba

            ds.SetCooperativeLevel(this.Handle, CooperativeLevel.Priority)

            WaveFormat format = new WaveFormat();
            format.BitsPerSample = 8;
            format.BlockAlignment = 1;
            format.Channels = 1;
            format.FormatTag = WaveFormatTag.Pcm;
            format.SamplesPerSecond = SamplesPerSecond;
            format.AverageBytesPerSecond = format.SamplesPerSecond * forma

            SoundBufferDescription desc = new SoundBufferDescription();
            desc.Format = format;
            desc.Flags = BufferFlags.GlobalFocus | BufferFlags.ControlPos
            desc.SizeInBytes = 8 * format.AverageBytesPerSecond;
            SecondarySoundBuffer sBuffer1 = new SecondarySoundBuffer(ds, d

            NotificationPosition[] notifications = new NotificationPositio
            notifications[0].Offset = 0;        // At the beginning of the
            notifications[1].Offset = 4 * format.AverageBytesPerSecond; //
            notifications[0].Event = new AutoResetEvent(false);
            notifications[1].Event = new AutoResetEvent(false);
            sBuffer1.SetNotificationPositions(notifications);

            Thread CaptureThread = new Thread((ThreadStart)delegate
            {
                byte[] bytes = new byte[4 * format.AverageBytesPerSecond];
                _buffer2.Read(bytes, 4 * format.AverageBytesPerSecond);
// load the first half of the buffer, then begin playback
                sBuffer1.Write<byte>(bytes, 0, LockFlags.None);
                sBuffer1.Play(0, PlayFlags.Looping);

                while (true)
                {
                    notifications[0].Event.WaitOne();     //wait until a
                    _buffer2.Read(bytes, 4 * format.AverageBytesPerSecond)
//read the next batch of audio data
                    sBuffer1.Write<byte>(bytes, 4 * format.AverageBytesPer
//write to the second half of the buffer
                    notifications[1].Event.WaitOne();    //block till pla
                    _buffer2.Read(bytes, 4 * format.AverageBytesPerSecond)
//read audio data from the stream
                    sBuffer1.Write<byte>(bytes, 0, LockFlags.None);
// write the data to the first half of the buffer
                }
```

```
            });
            CaptureThread.Start();

        }
    }

    public class InData
    {
        public byte Value;
        public DateTime TimeStamp;
        public double LastUpdate;
    }

    public class CircularBuffer
    {
        Queue<byte> _queue;

        public CircularBuffer()
        {
            _queue = new Queue<byte>();
        }

        public void Add(byte value)
        {

                _queue.Enqueue(value);
        }

        public void Read(byte[] bytes, int len)
        {
            for (int i = 0; i < len; i++)
            {
                if (_queue.Count > 0)
                {
                    bytes[i] = _queue.Dequeue();
                }

            }
        }

    }
}
```

B.10 Program.cs Phonocardiogram - DWT

DWT.cs performs an in-place discrete wavelett transform on an array of data using the Daubechies D4 coefficients. This code was ported to C# from Java code developed by Ian Kaplan. See http://www.bearcave.com/misl/misl_tech/wavelets/daubechies/index.html for more information.

Listing B.10: Phonocardiogram - DWT.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Phonocardiogram
{

/**
  <p>
  Daubechies D4 wavelet transform (D4 denotes four coefficients)
  </p>
  <p>
  I have to confess up front that the comment here does not even come
  close to describing wavelet algorithms and the Daubechies D4
  algorithm in particular.  I don't think that it can be described in
  anything less than a journal article or perhaps a book.  I even have
  to apologize for the notation I use to describe the algorithm, which
  is barely adequate.  But explaining the correct notation would take
  a fair amount of space as well.  This comment really represents some
  notes that I wrote up as I implemented the code.  If you are
  unfamiliar with wavelets I suggest that you look at the bearcave.com
  web pages and at the wavelet literature.  I have yet to see a really
  good reference on wavelets for the software developer.  The best
  book I can recommend is <i>Ripples in Mathematics</i> by Jensen and
  Cour-Harbo.
  </p>

  <p>
  All wavelet algorithms have two components, a wavelet function and a
  scaling function.  These are sometime also referred to as high pass
  and low pass filters respectively.
  </p>

  <p>
  The wavelet function is passed two or more samples
  and calculates a wavelet coefficient.  In the case of
  the Haar wavelet this is
  </p>

  <pre>
  coef<sub>i</sub> = odd<sub>i</sub> - even<sub>i</sub>
  or
  coef<sub>i</sub> = 0.5 * (odd<sub>i</sub> - even<sub>i</sub>)
  </pre>
  <p>
  depending on the version of the Haar algorithm used.
  </p>
  <p>
  The scaling function produces a smoother version of the
  original data.  In the case of the Haar wavelet algorithm
  this is an average of two adjacent elements.
  </p>
  <p>
  The Daubechies D4 wavelet algorithm also has a wavelet
  and a scaling function.  The coefficients for the
  scaling function are denoted as h<sub>i</sub> and the
  wavelet coefficients are g<sub>i</sub>.
  </p>
  <p>
  Mathematicians like to talk about wavelets in terms of
  a wavelet algorithm applied to an infinite data set.
```

```
   In this case one step of the forward transform can be expressed
   as the infinite matrix of wavelet coefficients
   represented below multiplied by the infinite signal
   vector.
   </p>
   <pre>
      a<sub>i</sub> = ...h0,h1,h2,h3, 0, 0, 0, 0, 0, 0, 0, ...
s<sub>i</sub>
      c<sub>i</sub> = ...g0,g1,g2,g3, 0, 0, 0, 0, 0, 0, 0, ...
s<sub>i+1</sub>
    a<sub>i+1</sub> = ...0, 0, h0,h1,h2,h3, 0, 0, 0, 0, 0, ...
s<sub>i+2</sub>
    c<sub>i+1</sub> = ...0, 0, g0,g1,g2,g3, 0, 0, 0, 0, 0, ...
s<sub>i+3</sub>
    a<sub>i+2</sub> = ...0, 0, 0, 0, h0,h1,h2,h3, 0, 0, 0, ...
s<sub>i+4</sub>
    c<sub>i+2</sub> = ...0, 0, 0, 0, g0,g1,g2,g3, 0, 0, 0, ...
s<sub>i+5</sub>
    a<sub>i+3</sub> = ...0, 0, 0, 0, 0, 0, h0,h1,h2,h3, 0, ...
s<sub>i+6</sub>
    c<sub>i+3</sub> = ...0, 0, 0, 0, 0, 0, g0,g1,g2,g3, 0, ...
s<sub>i+7</sub>
   </pre>
   <p>
   The dot product (inner product) of the infinite vector and
   a row of the matrix produces either a smoother version of the
   signal (a<sub>i</sub>) or a wavelet coefficient (c<sub>i</sub>).
   </p>
   <p>
   In an ordered wavelet transform, the smoothed (a<sub>i</sub>) are
   stored in the first half of an <i>n</i> element array region.
The
   wavelet coefficients (c<sub>i</sub>) are stored in the second half
   the <i>n</i> element region.  The algorithm is recursive.  The
   smoothed values become the input to the next step.
   </p>
   <p>
   The transpose of the forward transform matrix above is used
   to calculate an inverse transform step.  Here the dot product is
   formed from the result of the forward transform and an inverse
   transform matrix row.
   </p>
   <pre>
        s<sub>i</sub> = ...h2,g2,h0,g0, 0, 0, 0, 0, 0, 0, 0, ...
a<sub>i</sub>
      s<sub>i+1</sub> = ...h3,g3,h1,g1, 0, 0, 0, 0, 0, 0, 0, ...
c<sub>i</sub>
      s<sub>i+2</sub> = ...0, 0, h2,g2,h0,g0, 0, 0, 0, 0, 0, ...
a<sub>i+1</sub>
      s<sub>i+3</sub> = ...0, 0, h3,g3,h1,g1, 0, 0, 0, 0, 0, ...
c<sub>i+1</sub>
      s<sub>i+4</sub> = ...0, 0, 0, 0, h2,g2,h0,g0, 0, 0, 0, ...
a<sub>i+2</sub>
      s<sub>i+5</sub> = ...0, 0, 0, 0, h3,g3,h1,g1, 0, 0, 0, ...
c<sub>i+2</sub>
      s<sub>i+6</sub> = ...0, 0, 0, 0, 0, 0, h2,g2,h0,g0, 0, ...
a<sub>i+3</sub>
```

```
    s<sub>i+7</sub> = ...0, 0, 0, 0, 0, 0, h3,g3,h1,g1, 0, ...
c<sub>i+3</sub>
  </pre>

  <p>
  Using a standard dot product is grossly inefficient since most
  of the operands are zero. In practice the wavelet coefficient
  values are moved along the signal vector and a four element
  dot product is calculated. Expressed in terms of arrays, for
  the forward transform this would be:
  </p>
  <pre>
  a<sub>i</sub> = s[i]*h0 + s[i+1]*h1 + s[i+2]*h2 + s[i+3]*h3
  c<sub>i</sub> = s[i]*g0 + s[i+1]*g1 + s[i+2]*g2 + s[i+3]*g3
  </pre>
  <p>
  This works fine if we have an infinite data set, since we don't
  have to worry about shifting the coefficients "off the end" of
  the signal.
  </p>
  <p>
  I sometimes joke that I left my infinite data set in my other bear
  suit. The only problem with the algorithm described so far is that
  we don't have an infinite signal. The signal is finite. In fact
  not only must the signal be finite, but it must have a power of two
  number of elements.
  </p>
  <p>
  If i=N-1, the i+2 and i+3 elements will be beyond the end of
  the array. There are a number of methods for handling the
  wavelet edge problem. This version of the algorithm acts
  like the data is periodic, where the data at the start of
  the signal wraps around to the end.
  </p>
  <p>
  This algorithm uses a temporary array. A Lifting Scheme version of
  the Daubechies D4 algorithm does not require a temporary. The
  matrix discussion above is based on material from <i>Ripples in
  Mathematics</i>, by Jensen and Cour-Harbo. Any error are mine.
  </p>

  <p>
  <b>Author</b>: Ian Kaplan<br>
  <b>Use</b>: You may use this software for any purpose as long
  as I cannot be held liable for the result. Please credit me
  with authorship if use use this source code.
  </p>
 */
class daub {
   protected static double sqrt_3 = Math.Sqrt( 3 );
   protected static double denom = 4 * Math.Sqrt(2);
   //
   // forward transform scaling (smoothing) coefficients
   //
   protected static double h0 = (1 + sqrt_3) / denom;
   protected static double h1 = (3 + sqrt_3) / denom;
   protected static double h2 = (3 - sqrt_3) / denom;
   protected static double h3 = (1 - sqrt_3) / denom;
   //
   // forward transform wavelet coefficients
```

```
//
protected static double g0 = h3;
protected static double g1 = -h2;
protected static double g2 = h1;
protected static double g3 = -h0;

//
// Inverse transform coefficients for smoothed values
//
protected static double Ih0 = h2;
protected static double Ih1 = g2;  // h1
protected static double Ih2 = h0;
protected static double Ih3 = g0;  // h3
//
// Inverse transform for wavelet values
//
protected static double Ig0 = h3;
protected static double Ig1 = g3;  // -h0
protected static double Ig2 = h1;
protected static double Ig3 = g1;  // -h2

/**
  <p>
  Forward wavelet transform.
  </p>
  <p>
  Note that at the end of the computation the
  calculation wraps around to the beginning of
  the signal.
  </p>
 */
protected void transform( double[] a, int n )
{
   if (n >= 4) {
      int i, j;
      int half = n >> 1;

      double[] tmp = new double[n];

      i = 0;
      for (j = 0; j < n-3; j = j + 2) {
         tmp[i]      = a[j]*h0 + a[j+1]*h1 + a[j+2]*h2 + a[j+3]*h3;
         tmp[i+half] = a[j]*g0 + a[j+1]*g1 + a[j+2]*g2 + a[j+3]*g3;
         i++;
      }

      tmp[i]      = a[n-2]*h0 + a[n-1]*h1 + a[0]*h2 + a[1]*h3;
      tmp[i+half] = a[n-2]*g0 + a[n-1]*g1 + a[0]*g2 + a[1]*g3;

      for (i = 0; i < n; i++) {
         a[i] = tmp[i];
      }
   }
} // transform


protected void invTransform( double[] a, int n )
{
   if (n >= 4) {
     int i, j;
     int half = n >> 1;
```

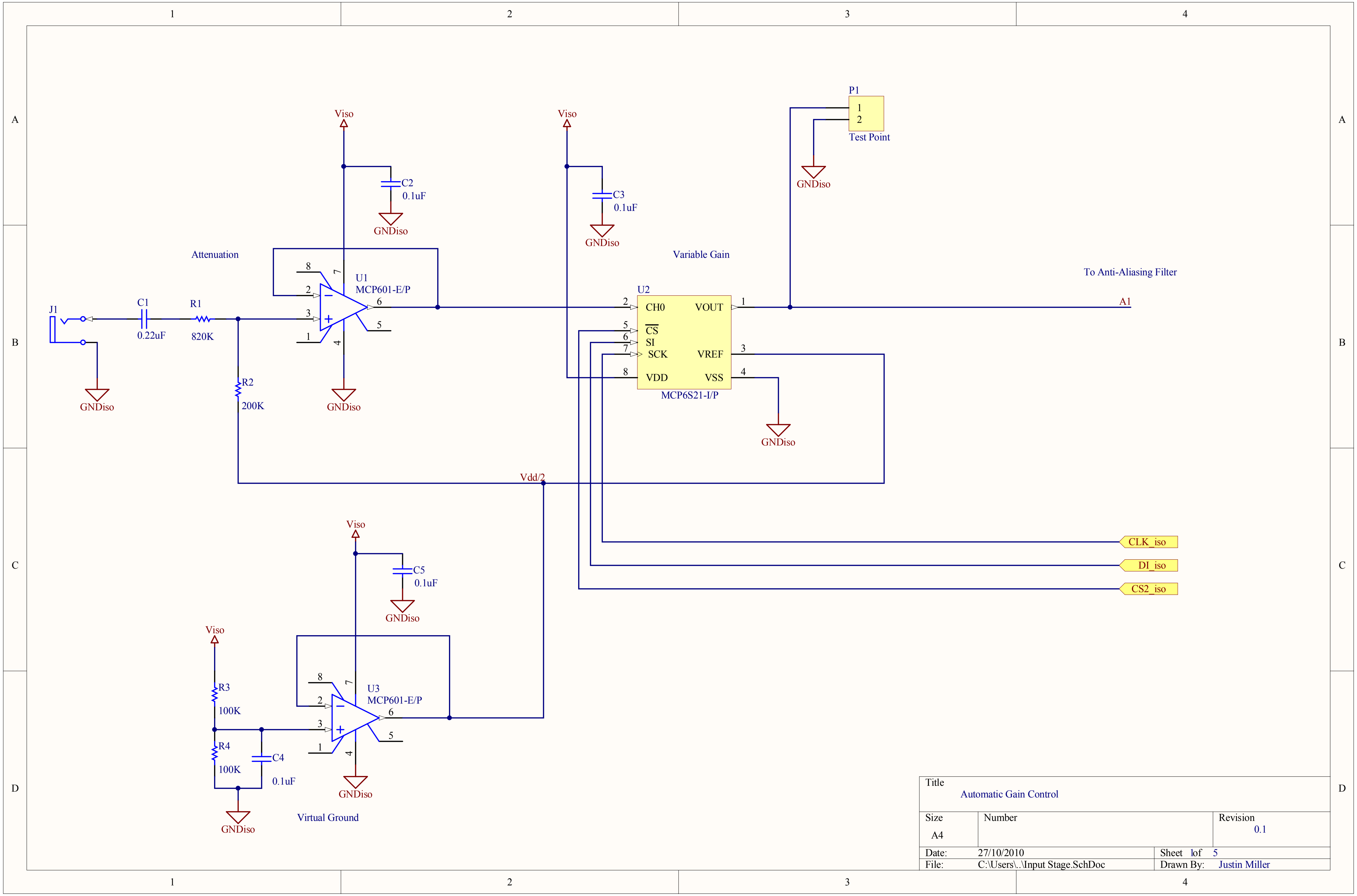
```
        int halfPls1 = half + 1;

        double[] tmp = new double[n];

        //      last smooth val   last coef.   first smooth   first coef
        tmp[0] = a[half-1]*Ih0 + a[n-1]*Ih1 + a[0]*Ih2 + a[half]*Ih3;
        tmp[1] = a[half-1]*Ig0 + a[n-1]*Ig1 + a[0]*Ig2 + a[half]*Ig3;
        j = 2;
        for (i = 0; i < half-1; i++) {
          //     smooth val     coef. val        smooth val    coef. val
          tmp[j++] = a[i]*Ih0 + a[i+half]*Ih1 + a[i+1]*Ih2 + a[i+halfPls1]
          tmp[j++] = a[i]*Ig0 + a[i+half]*Ig1 + a[i+1]*Ig2 + a[i+halfPls1]
        }
        for (i = 0; i < n; i++) {
          a[i] = tmp[i];
        }
      }
    }

    /**
      Forward Daubechies D4 transform
     */
    public void daubTrans( double[] s)
    {
       int N = s.Length;
       int n;
       for (n = N; n >= 4; n >>= 1) {
          transform( s, n );
       }
    }

    /**
      Inverse Daubechies D4 transform
     */
    public void invDaubTrans( double[] coef)
    {
       int N = coef.Length;
       int n;
       for (n = 4; n <= N; n <<= 1) {
          invTransform( coef, n );
       }
    }

} // daub

}
```

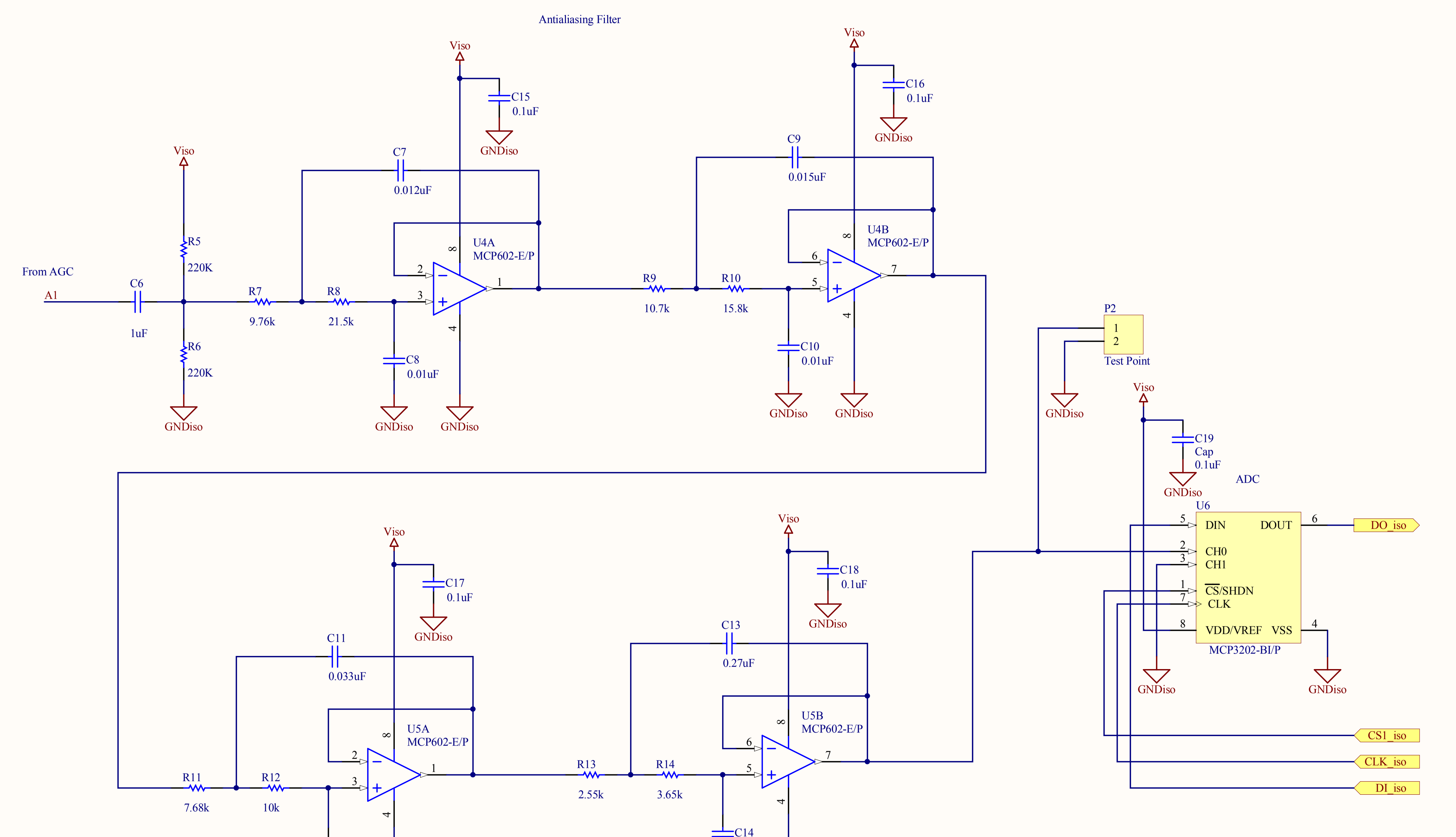
Appendix C

Schematics

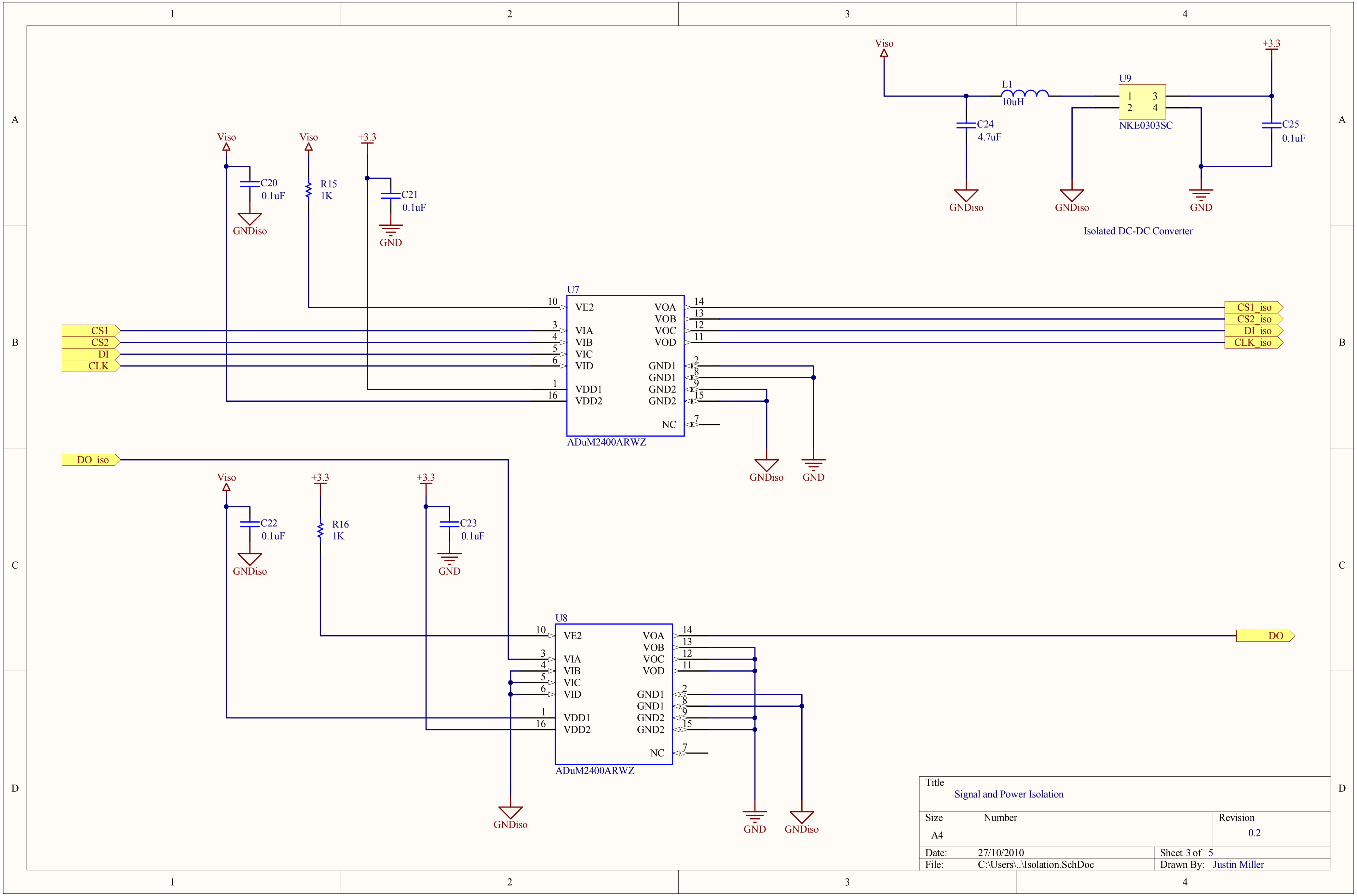
Listed in this appendix are the schematics for the wireless acquisition module in the following order:

1. Automatic Gain Control
2. Anti-Aliasing Filter and Analog-To-Digital Converter
3. Signal and Power Isolation
4. Microcontroller and Bluetooth Module
5. Power Supply and Battery Management

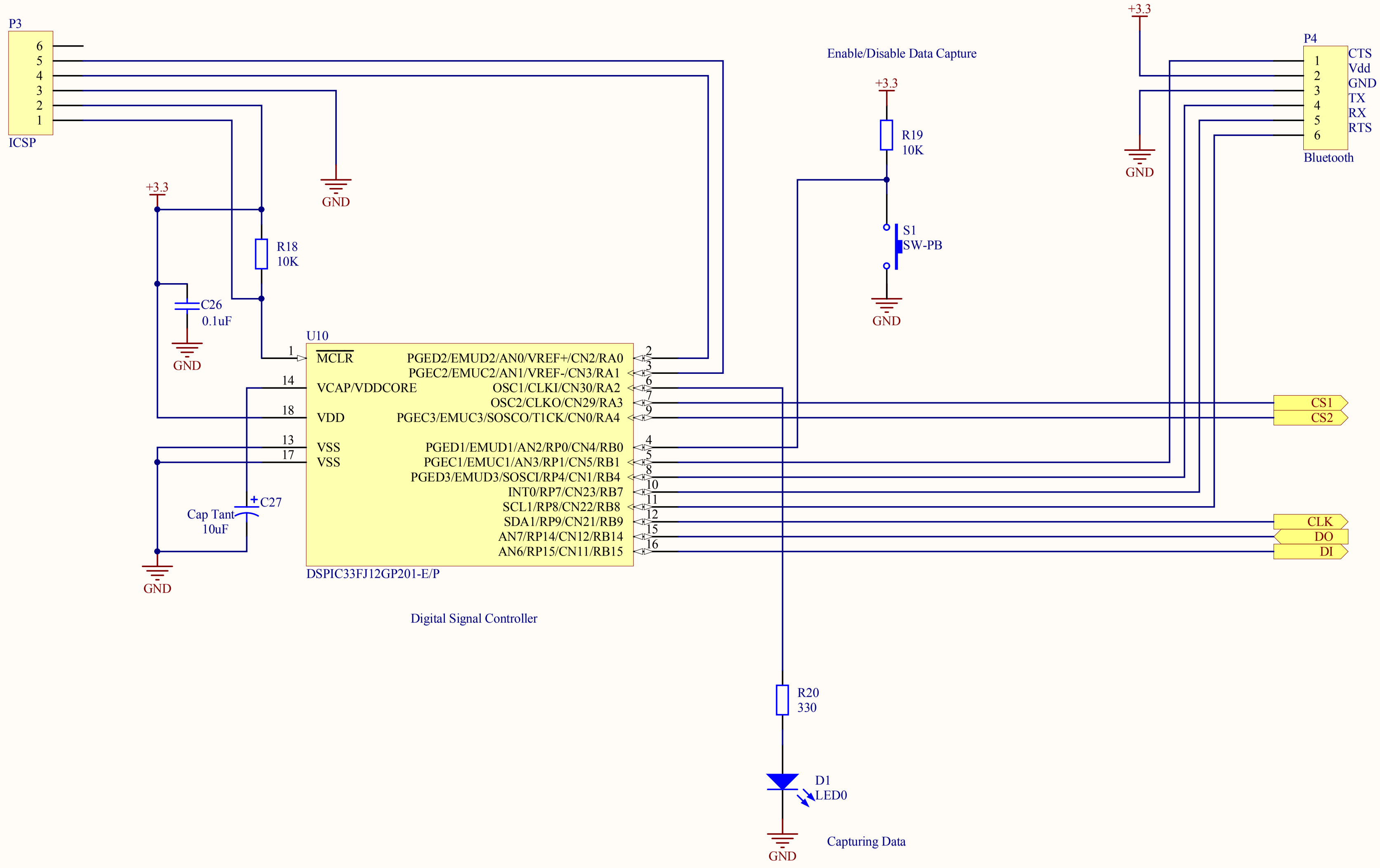

1
2
3
4
A
B
C
D
Attenuation
J1
C1
0.22uF
R1
820K
R2
200K
GNDiso
Viso
C2
0.1uF
U1
MCP601-E/P
Viso
C3
0.1uF
Variable Gain
U2
CH0
CS
SI
SCK
VDD
VOUT
VREF
VSS
MCP6S21-I/P
P1
Test Point
To Anti-Aliasing Filter
A1
Vdd/2
CLK_iso
DI_iso
CS2_iso
Viso
C5
0.1uF
U3
MCP601-E/P
R3
100K
R4
100K
C4
0.1uF
Virtual Ground
Title
Automatic Gain Control
Size
A4
Number
Revision
0.1
Date: 27/10/2010
File: C:\Users\..\Input Stage.SchDoc
Sheet 1of 5
Drawn By: Justin Miller



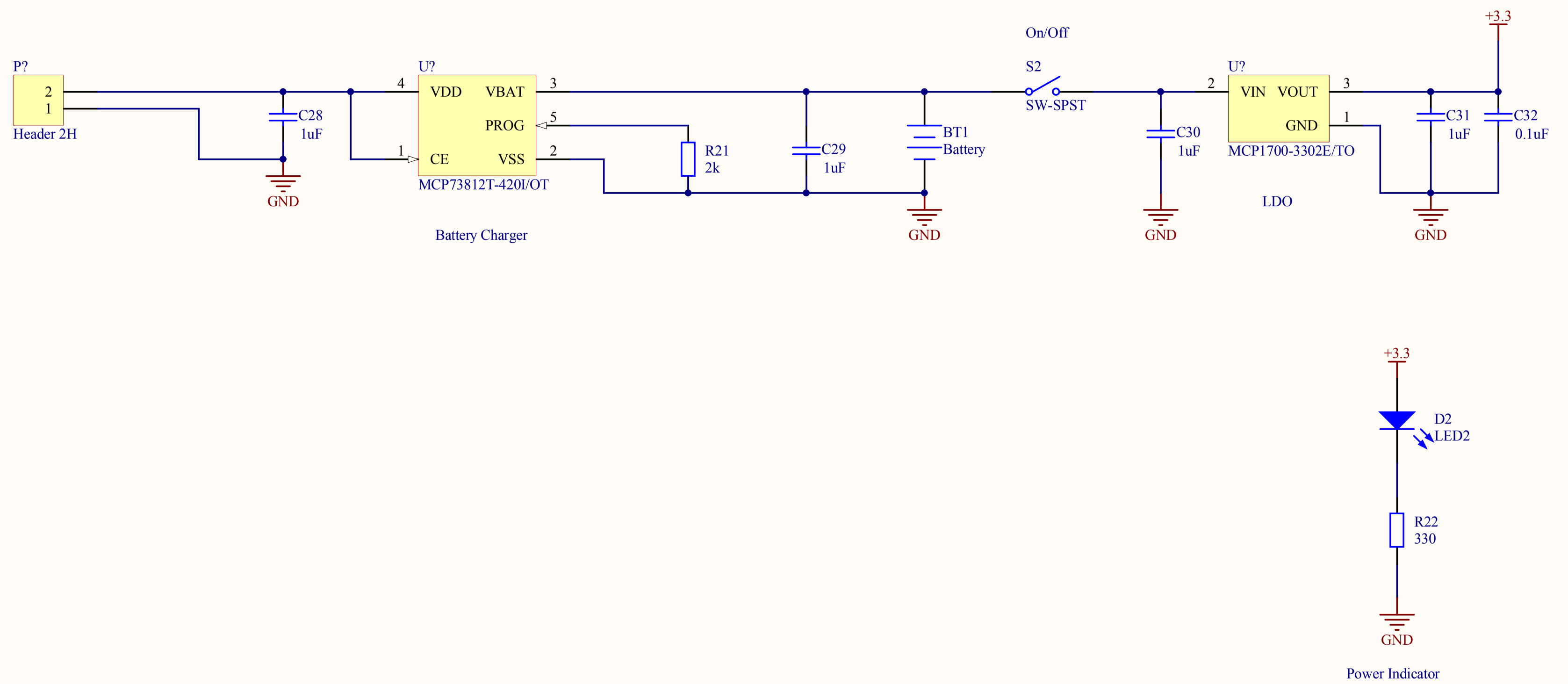
Title	Analog to Digital Converter		
Size	A4	Number	Revision 0.2
Date:	27/10/2010	Sheet 2 of 5	
File:	C:\Users\..\ADCStage.SchDoc	Drawn By:	Justin Miller



Viso
+3.3
L1
10uH
U9
NKE0303SC
C24
4.7uF
C25
0.1uF
GNDiso
GND
Isolated DC-DC Converter
C20
0.1uF
R15
1K
C21
0.1uF
U7
ADuM2400ARWZ
VE2
VIA
VIB
VIC
VID
VDD1
VDD2
VOA
VOB
VOC
VOD
GND1
GND2
NC
CS1
CS2
DI
CLK
CS1_iso
CS2_iso
DI_iso
CLK_iso
DO_iso
C22
0.1uF
R16
1K
C23
0.1uF
U8
ADuM2400ARWZ
DO
Title
Signal and Power Isolation
Size
A4
Number
Revision
0.2
Date: 27/10/2010
Sheet 3 of 5
File: C:\Users\..\Isolation.SchDoc
Drawn By: Justin Miller



Title			
Digital Signal Controller and Bluetooth Module			
Size: A4	Number		Revision: 0.2
Date: 27/10/2010		Sheet 4of 5	
File: C:\Users\..\Aquisition Module.SchDoc		Drawn By: Justin Miller	



Title	Power Supply and Battery Management		
Size	Number		Revision
A4			0.1
Date:	27/10/2010	Sheet 5 of 5	
File:	C:\Users\..\Power Supply.SchDoc	Drawn By:	Justin Miller

Appendix D

Datasheets

MCP601/2/3/4

2.7V to 5.5V Single-Supply CMOS Op Amps

Features

- Single-Supply: 2.7V to 5.5V
- Rail-to-Rail Output
- Input Range Includes Ground
- Gain Bandwidth Product: 2.8 MHz (typ.)
- Unity-Gain Stable
- Low Quiescent Current: 230 µA/amplifier (typ.)
- Chip Select ($\overline{CS}$): **MCP603 only**
- Temperature Ranges:
  - Industrial: -40°C to +85°C
  - Extended: -40°C to +125°C
- Available in Single, Dual and Quad

Typical Applications

- Portable Equipment
- A/D Converter Driver
- Photo Diode Pre-amp
- Analog Filters
- Data Acquisition
- Notebooks and PDAs
- Sensor Interface

Available Tools

- SPICE Macro Models at www.microchip.com
- FilterLab® Software at www.microchip.com

Description

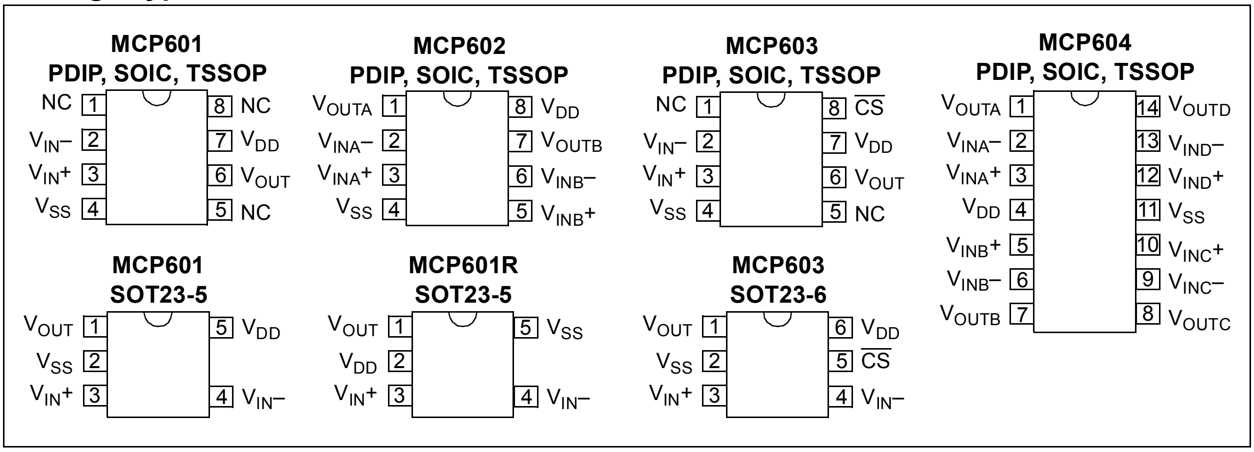
The Microchip Technology Inc. MCP601/2/3/4 family of low-power operational amplifiers (op amps) are offered in single (MCP601), single with Chip Select ($\overline{CS}$) (MCP603), dual (MCP602) and quad (MCP604) configurations. These op amps utilize an advanced CMOS technology that provides low bias current, high-speed operation, high open-loop gain and rail-to-rail output swing. This product offering operates with a single supply voltage that can be as low as 2.7V, while drawing 230 µA (typ.) of quiescent current per amplifier. In addition, the common mode input voltage range goes 0.3V below ground, making these amplifiers ideal for single-supply operation.

These devices are appropriate for low-power, battery-operated circuits due to the low quiescent current, for A/D convert driver amplifiers because of their wide bandwidth or for anti-aliasing filters by virtue of their low input bias current.

The MCP601, MCP602 and MCP603 are available in standard 8-lead PDIP, SOIC and TSSOP packages. The MCP601 and MCP601R are also available in a standard 5-lead SOT-23 package, while the MCP603 is available in a standard 6-lead SOT-23 package. The MCP604 is offered in standard 14-lead PDIP, SOIC and TSSOP packages.

The MCP601/2/3/4 family is available in the Industrial and Extended temperature ranges and has a power supply range of 2.7V to 5.5V.

Package Types



MCP6S21/2/6/8

Single-Ended, Rail-to-Rail I/O, Low Gain PGA

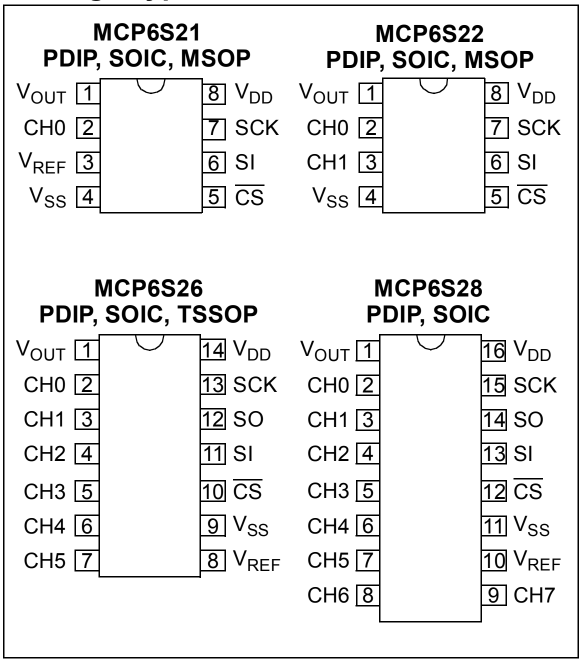
Features

- Multiplexed Inputs: 1, 2, 6 or 8 channels
- 8 Gain Selections:
  - +1, +2, +4, +5, +8, +10, +16 or +32 V/V
- Serial Peripheral Interface (SPI™)
- Rail-to-Rail Input and Output
- Low Gain Error: ±1% (max)
- Low Offset: ±275 µV (max)
- High Bandwidth: 2 to 12 MHz (typ)
- Low Noise: 10 nV/√Hz @ 10 kHz (typ)
- Low Supply Current: 1.0 mA (typ)
- Single Supply: 2.5V to 5.5V

Typical Applications

- A/D Converter Driver
- Multiplexed Analog Applications
- Data Acquisition
- Industrial Instrumentation
- Test Equipment
- Medical Instrumentation

Package Types

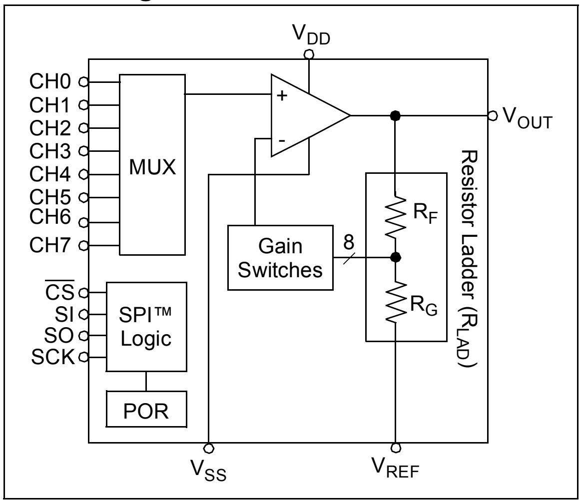


Description

The Microchip Technology Inc. MCP6S21/2/6/8 are analog Programmable Gain Amplifiers (PGA). They can be configured for gains from +1 V/V to +32 V/V and the input multiplexer can select one of up to eight channels through an SPI port. The serial interface can also put the PGA into shutdown to conserve power. These PGAs are optimized for high speed, low offset voltage and single-supply operation with rail-to-rail input and output capability. These specifications support single supply applications needing flexible performance or multiple inputs.

The one channel MCP6S21 and the two channel MCP6S22 are available in 8-pin PDIP, SOIC and MSOP packages. The six channel MCP6S26 is available in 14-pin PDIP, SOIC and TSSOP packages. The eight channel MCP6S28 is available in 16-pin PDIP and SOIC packages. All parts are fully specified from -40°C to +85°C.

Block Diagram



MCP3201

2.7V 12-Bit A/D Converter with SPI™ Serial Interface

Features

- 12-bit resolution
- ±1 LSB max DNL
- ±1 LSB max INL (MCP3201-B)
- ±2 LSB max INL (MCP3201-C)
- On-chip sample and hold
- SPI™ serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V - 5.5V
- 100ksps max. sampling rate at V_{DD} = 5V
- 50ksps max. sampling rate at V_{DD} = 2.7V
- Low power CMOS technology
- 500 nA typical standby current, 2 µA max.
- 400 µA max. active current at 5V
- Industrial temp range: -40°C to +85°C
- 8-pin MSOP, PDIP, SOIC and TSSOP packages

Applications

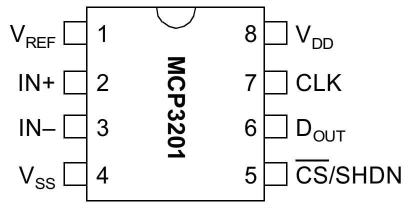
- Sensor Interface
- Process Control
- Data Acquisition
- Battery Operated Systems

Description

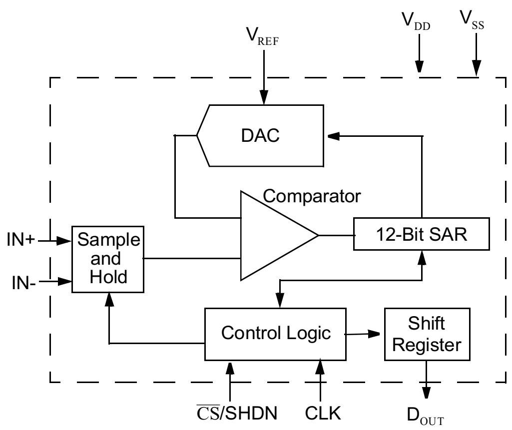
The Microchip Technology Inc. MCP3201 is a successive approximation 12-bit Analog-to-Digital (A/D) Converter with on-board sample and hold circuitry. The device provides a single pseudo-differential input. Differential Nonlinearity (DNL) is specified at ±1 LSB, and Integral Nonlinearity (INL) is offered in ±1 LSB (MCP3201-B) and ±2 LSB (MCP3201-C) versions. Communication with the device is done using a simple serial interface compatible with the SPI protocol. The device is capable of sample rates of up to 100 ksps at a clock rate of 1.6 MHz. The MCP3201 operates over a broad voltage range (2.7V - 5.5V). Low current design permits operation with typical standby and active currents of only 500 nA and 300 µA, respectively. The device is offered in 8-pin MSOP, PDIP, TSSOP and 150 mil SOIC packages.

Package Types

MSOP, PDIP, SOIC, TSSOP



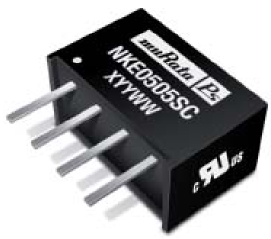
Functional Block Diagram



Murata Power Solutions

NKE Series

Isolated Sub-Miniature 1W Single Output DC/DC Converters



FEATURES

- RoHS Compliant
- Sub-Miniature SIP & DIP Styles
- 3kVDC Isolation
- UL Recognised
- Wide Temperature performance at full 1 Watt load, –40°C to 85°C
- Increased Power Density to 2.09W/cm^3
- UL 94V-0 Package Material
- Footprint at 0.69cm^2
- Industry Standard Pinout
- 3.3V, 5V & 12V Input
- 3.3V, 5V, 9V, 12V and 15V Output
- Internal SMD Construction
- Fully Encapsulated with Toroidal Magnetics
- MTTF up to 2.4 Million hours
- Custom Solutions Available
- No Electrolytic or Tantalum Capacitors

DESCRIPTION

The NKE sub-miniature series of DC/DC Converters is particularly suited to isolating and/or converting DC power rails. A smaller package size, improved efficiency, lower output ripple and 3kVDC isolation capability through state of the art packaging and improved technology. The galvanic isolation allows the device to be configured to provide an isolated negative rail in systems where only positive rails exist. The wide temperature range guarantees startup from –40°C and full 1 watt output at 85°C.

C US

RoHS COMPLIANT 2002/95/EC

For full details go to www.murata-ps.com/rohs

SELECTION GUIDE

Order Code	Nominal Input Voltage	Output Voltage	Output Current	Input Current at Rated Load	Efficiency		Isolation Capacitance	MTTF[1]	Package Style
	V	V	mA	mA	%		pF	kHrs	
					Min.	Typ.			
NKE0303DC	3.3	3.3	303	400	68	72	30	1234	
NKE0305DC	3.3	5	200	400	72	75	35	632	
NKE0309DC	3.3	9	111	403	71	74	30	1204	DIP
NKE0312DC[3]	3.3	12	83	398	73	76	33		
NKE0315DC[3]	3.3	15	66	394	74	77	35		
NKE0303SC	3.3	3.3	303	400	68	72	30	1234	
NKE0305SC	3.3	5	200	400	72	75	35	632	SIP
NKE0309SC	3.3	9	111	403	71	74	30	1204	
NKE0503DC	5	3.3	303	270	70	74	40	619	
NKE0505DC	5	5	200	289	66	69	28	2414	
NKE0505DEC	5	5	200	250	75	77	34	419	
NKE0509DC	5	9	111	266	72	75	29	1173	DIP
NKE0512DC	5	12	83	260	73	78	30	633	
NKE0515DC	5	15	66	256	74	78	32	360	
NKE0503SC	5	3.3	303	270	70	74	40	619	
NKE0505SC	5	5	200	289	66	69	28	2414	
NKE0505SEC	5	5	200	250	75	77	34	419	SIP
NKE0509SC	5	9	111	266	72	75	29	1173	
NKE0512SC	5	12	83	260	73	78	30	633	
NKE0515SC	5	15	66	256	74	78	32	360	
NKE1205DC	12	5	200	117	68	72	35	620	
NKE1209DC	12	9	111	107	72	78	50	488	
NKE1212DC	12	12	83	105	73	79	57	360	DIP
NKE1215DC	12	15	66	103	76	81	60	252	
NKE1205SC	12	5	200	117	68	72	35	620	
NKE1209SC	12	9	111	107	72	78	50	488	
NKE1212SC	12	12	83	105	73	79	57	360	SIP
NKE1215SC	12	15	66	103	76	81	60	252	

NKE0505SEC/NKE0505DEC offers higher efficiency than NKE0505SC/NKE0505DC but over a narrower operating temperature range. See temperature characteristics graph.

INPUT CHARACTERISTICS

Parameter	Conditions	Min.	Typ.	Max.	Units
Voltage range	Continuous operation, 3.3V input types	2.97	3.3	3.63	V
	Continuous operation, 5V input types	4.5	5.0	5.5	
	Continuous operation, 12V input types	10.8	12.0	13.2	
Reflected ripple current	3.3V input types		40	60	mA p-p

ABSOLUTE MAXIMUM RATINGS

Lead temperature 1.5mm from case for 10 seconds	300°C
Internal power dissipation	530mW
Input voltage V_{IN}, NKE03 types	5.5V
Input voltage V_{IN}, NKE05 types	7V
Input voltage V_{IN}, NKE12 types	15V

1. Calculated using MIL-HDBK-217F with nominal input voltage at full load.

All specifications typical at T_A=25°C, nominal input voltage and rated output current unless otherwise specified.

Quad-Channel Digital Isolators

ADuM2400/ADuM2401/ADuM2402

FEATURES

Low power operation
- **5 V operation**
  - **1.0 mA per channel maximum @ 0 Mbps to 2 Mbps**
  - **3.5 mA per channel maximum @ 10 Mbps**
  - **31 mA per channel maximum @ 90 Mbps**
- **3 V operation**
  - **0.7 mA per channel maximum @ 0 Mbps to 2 Mbps**
  - **2.1 mA per channel maximum @ 10 Mbps**
  - **20 mA per channel maximum @ 90 Mbps**

Bidirectional communication
3 V/5 V level translation
High temperature operation: 105°C
High data rate: dc to 90 Mbps (NRZ)
Precise timing characteristics
- **2 ns maximum pulse width distortion**
- **2 ns maximum channel-to-channel matching**

High common-mode transient immunity: >25 kV/μs
Output enable function
16-lead SOIC wide body package (RoHS compliant)
Safety and regulatory approvals
- **UL recognition: 5000 V rms for 1 minute per UL 1577**
- **CSA Component Acceptance Notice #5A**
  - **IEC 60950-1: 600 V rms (reinforced)**
  - **IEC 60601-1: 250 V rms (reinforced)**
- **VDE Certificate of Conformity**
  - **DIN V VDE V 0884-10 (VDE V 0884-10):2006-12**
  - **V_{IORM} = 846 V peak**

APPLICATIONS

General-purpose, high voltage, multichannel isolation
Medical equipment
Motor drives
Power supplies

GENERAL DESCRIPTION

The ADuM240x[1] are 4-channel digital isolators based on Analog Devices, Inc., *i*Coupler® technology. Combining high speed CMOS and monolithic air core transformer technology, these isolation components provide outstanding performance characteristics that are superior to alternatives, such as optocoupler devices.

By avoiding the use of LEDs and photodiodes, *i*Coupler devices remove the design difficulties commonly associated with optocouplers. The typical optocoupler concerns regarding uncertain current transfer ratios, nonlinear transfer functions, and temperature and lifetime effects are eliminated with the simple *i*Coupler digital interfaces and stable performance characteristics. Furthermore, *i*Coupler devices run at one-tenth to one-sixth the power of optocouplers at comparable signal data rates.

The ADuM240x isolators provide four independent isolation channels in a variety of channel configurations and data rates (see the Ordering Guide). The ADuM240x models operate with the supply voltage of either side ranging from 2.7 V to 5.5 V, providing compatibility with lower voltage systems as well as enabling a voltage translation functionality across the isolation barrier. In addition, the ADuM240x provide low pulse width distortion (<2 ns for CRWZ grade) and tight channel-to-channel matching (<2 ns for CRWZ grade). Unlike other optocoupler alternatives, the ADuM240x isolators have a patented refresh feature that ensures dc correctness in the absence of input logic transitions and during power-up/power-down conditions.

[1] Protected by U.S. Patents 5,952,849; 6,873,065; and 7,075,329. Other patents pending.

FUNCTIONAL BLOCK DIAGRAMS

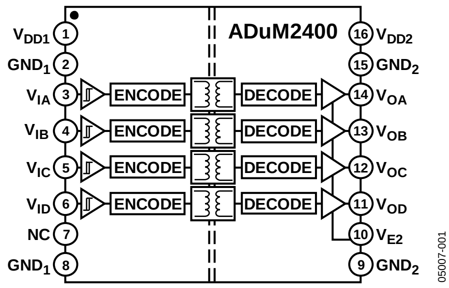


Figure 1. ADuM2400

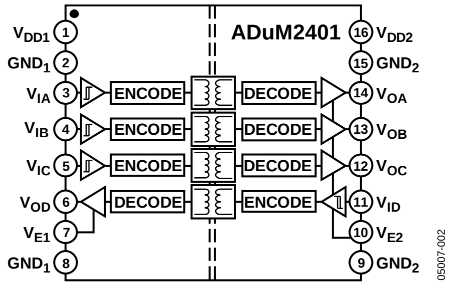


Figure 2. ADuM2401

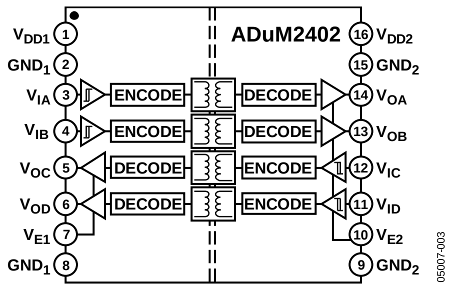


Figure 3. ADuM2402

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.461.3113

dsPIC33FJ12GP201/202

High-Performance, 16-Bit Digital Signal Controllers

Operating Range:

- Up to 40 MIPS operation (at 3.0-3.6V):
  - Industrial temperature range (-40°C to +85°C)
  - Extended temperature range (-40°C to +125°C)

High-Performance DSC CPU:

- Modified Harvard architecture
- C compiler optimized instruction set
- 16-bit-wide data path
- 24-bit-wide instructions
- Linear program memory addressing up to 4M instruction words
- Linear data memory addressing up to 64 Kbytes
- 83 base instructions, mostly one word/one cycle
- Sixteen 16-bit general purpose registers
- Two 40-bit accumulators with rounding and saturation options
- Flexible and powerful addressing modes:
  - Indirect
  - Modulo
  - Bit-Reversed
- Software stack
- 16 x 16 fractional/integer multiply operations
- 32/16 and 16/16 divide operations
- Single-cycle multiply and accumulate:
  - Accumulator write back for DSP operations
  - Dual data fetch
- Up to ±16-bit shifts for up to 40-bit data

Interrupt Controller:

- 5-cycle latency
- Up to 21 available interrupt sources
- Up to three external interrupts
- Seven programmable priority levels
- Four processor exceptions

On-Chip Flash and SRAM:

- Flash program memory (12 Kbytes)
- Data SRAM (1024 bytes)
- Boot and General Security for Program Flash

Digital I/O:

- Peripheral Pin Select Functionality
- Up to 21 programmable digital I/O pins
- Wake-up/interrupt-on-change for up to 21 pins
- Output pins can drive from 3.0V to 3.6V
- Up to 5V output with open drain configuration
- All digital input pins are 5V tolerant
- 4 mA sink on all I/O pins

System Management:

- Flexible clock options:
  - External, crystal, resonator, internal RC
  - Fully integrated Phase-Locked Loop (PLL)
  - Extremely low-jitter PLL
- Power-up Timer
- Oscillator Start-up Timer/Stabilizer
- Watchdog Timer with its own RC oscillator
- Fail-Safe Clock Monitor
- Reset by multiple sources

Power Management:

- On-chip 2.5V voltage regulator
- Switch between clock sources in real time
- Idle, Sleep and Doze modes with fast wake-up

Timers/Capture/Compare:

- Timer/Counters, up to three 16-bit timers:
  - Can pair up to make one 32-bit timer
  - One timer runs as Real-Time Clock with external 32.768 kHz oscillator
  - Programmable prescaler
- Input Capture (up to four channels):
  - Capture on up, down, or both edges
  - 16-bit capture input functions
  - 4-deep FIFO on each capture
- Output Compare (up to two channels):
  - Single or Dual 16-bit Compare mode
  - 16-bit Glitchless PWM Mode

Communication Modules:

- 4-wire SPI:
  - Framing supports I/O interface to simple codecs
  - Supports 8-bit and 16-bit data
  - Supports all serial clock formats and sampling modes
- I^2C™:
  - Full Multi-Master Slave mode support
  - 7-bit and 10-bit addressing
  - Bus collision detection and arbitration
  - Integrated signal conditioning
  - Slave address masking
- UART:
  - Interrupt on address bit detect
  - Interrupt on UART error
  - Wake-up on Start bit from Sleep mode
  - 4 character TX and RX FIFO buffers
  - LIN bus support
  - IrDA® encoding and decoding in hardware
  - High-Speed Baud mode
  - Hardware Flow Control with CTS and RTS

Analog-to-Digital Converters (ADCs):

- 10-bit, 1.1 Msps or 12-bit, 500 Ksps conversion:
  - Two and four simultaneous samples (10-bit ADC)
  - Up to 10 input channels with auto-scanning
  - Conversion start can be manual or synchronized with one of four trigger sources
  - Conversion possible in Sleep mode
  - ±2 LSb max integral nonlinearity
  - ±1 LSb max differential nonlinearity

CMOS Flash Technology:

- Low-power, high-speed Flash technology
- Fully static design
- 3.3V (±10%) operating voltage
- Industrial and extended temperature
- Low power consumption

Packaging:

- 18-pin SDIP/SOIC
- 28-pin SDIP/SOIC/SSOP/QFN

Note: See Table 1 for the exact peripheral features per device.

dsPIC33FJ12GP201/202 Product Families

The device names, pin counts, memory sizes, and peripheral availability of each family are listed below, followed by their pinout diagrams.

TABLE 1: dsPIC33FJ12GP201/202 CONTROLLER FAMILIES

Device	Pins	Program Flash Memory (Kbyte)	RAM (Kbyte)	Remappable Peripherals							10-Bit/12-Bit ADC	I²C™	I/O Pins (Max)	Packages
				Remappable Pins	16-bit Timer	Input Capture	Output Compare Std. PWM	UART	External Interrupts(2)	SPI				
dsPIC33FJ12GP201	18	12	1	8	3(1)	4	2	1	3	1	1 ADC, 6 ch	1	13	SDIP SOIC
dsPIC33FJ12GP202	28	12	1	16	3(1)	4	2	1	3	1	1 ADC, 10 ch	1	21	SDIP SOIC SSOP QFN

Note 1: Only two out of three timers are remappable.

2: Only two out of three interrupts are remappable.

MICROCHIP

MCP1700

Low Quiescent Current LDO

Features

- 1.6 µA Typical Quiescent Current
- Input Operating Voltage Range: 2.3V to 6.0V
- Output Voltage Range: 1.2V to 5.0V
- 250 mA Output Current for output voltages ≥ 2.5V
- 200 mA Output Current for output voltages < 2.5V
- Low Dropout (LDO) voltage
  - 178 mV typical @ 250 mA for V_{OUT} = 2.8V
- 0.4% Typical Output Voltage Tolerance
- Standard Output Voltage Options:
  - 1.2V, 1.8V, 2.5V, 3.0V, 3.3V, 5.0V
- Stable with 1.0 µF Ceramic Output capacitor
- Short Circuit Protection
- Overtemperature Protection

Applications

- Battery-powered Devices
- Battery-powered Alarm Circuits
- Smoke Detectors
- CO^2 Detectors
- Pagers and Cellular Phones
- Smart Battery Packs
- Low Quiescent Current Voltage Reference
- PDAs
- Digital Cameras
- Microcontroller Power

Related Literature

- AN765, “Using Microchip’s Micropower LDOs”, DS00765, Microchip Technology Inc., 2002
- AN766, “Pin-Compatible CMOS Upgrades to BiPolar LDOs”, DS00766, Microchip Technology Inc., 2002
- AN792, “A Method to Determine How Much Power a SOT23 Can Dissipate in an Application”, DS00792, Microchip Technology Inc., 2001

General Description

The MCP1700 is a family of CMOS low dropout (LDO) voltage regulators that can deliver up to 250 mA of current while consuming only 1.6 µA of quiescent current (typical). The input operating range is specified from 2.3V to 6.0V, making it an ideal choice for two and three primary cell battery-powered applications, as well as single cell Li-Ion-powered applications.

The MCP1700 is capable of delivering 250 mA with only 178 mV of input to output voltage differential (V_{OUT} = 2.8V). The output voltage tolerance of the MCP1700 is typically ±0.4% at +25°C and ±3% maximum over the operating junction temperature range of -40°C to +125°C.

Output voltages available for the MCP1700 range from 1.2V to 5.0V. The LDO output is stable when using only 1 µF output capacitance. Ceramic, tantalum or aluminum electrolytic capacitors can all be used for input and output. Overcurrent limit and overtemperature shutdown provide a robust solution for any application.

Package options include the SOT-23, SOT-89 and TO-92.

Package Types