Spring 2012

Data acquisition unit for low-noise, continuous glucose monitoring

Daniel Warren Cooley
University of Iowa

DATA ACQUISITION UNIT FOR LOW-NOISE, CONTINUOUS GLUCOSE MONITORING

by

Daniel Warren Cooley

<u>An Abstract</u>

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy degree
in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

May 2012

Thesis Supervisor: Professor David R. Andersen

ABSTRACT

As the number of people with diabetes continues to increase, research efforts improving glucose testing methods and devices are under way to improve outcomes and quality of life for diabetic patients. This dissertation describes the design and testing of a Data Acquisition Unit (DAU) providing low noise photocurrent spectra for use in a continuous glucose monitoring system. The goal of this research is to improve the signal to noise ratio (SNR) of photocurrent measurements to increase glucose concentration measurement accuracy. The glucose monitoring system consists of a portable monitoring device and base station. The monitoring device measures near infrared (IR) absorption spectra from interstitial fluid obtained by microdialysis or ultrafiltration probe and transmits the spectra to a base station via USB or a ZigBee radio link. The base station utilizes chemometric calibration methods to calculate glucose concentration from the photocurrent spectra. Future efforts envision credit card-sized monitoring devices.

The glucose monitor system measures the optical absorbance spectrum of an interstitial fluid (ISF) sample pumped through a fluid chamber inside a glucose sensor. Infrared LEDs in the glucose sensor illuminate the ISF sample with IR light covering the 2.2 to 2.4 micron wavelength region where glucose has unique features in its absorption spectrum. Light that passes through the sample propagates through a linearly variable bandpass filter and impinges on a photodiode array. The center frequency of the variable filter is graded along its length such that the filter and photodiode array form a spectrometer. The data acquisition unit (DAU) conditions and samples photocurrent from each photodiode channel and sends the resulting photocurrent spectra to the Main Controller Unit (MCU). The MCU filters photocurrent samples providing low noise photocurrent spectra to a base station via USB or Zigbee radio link.

The glucose monitoring system limit of detection (LOD) from a single glucose sensor wavelength is 5.8 mM with a system bandwidth of 0.00108 Hz. The partial least squares and net analyte signal methods show the system standard error of prediction for glucose are 1.12 mM and 1.88 mM, respectively - useful for detection of hyperglycemia but slightly high for indication of hypoglycemia.

Abstract Approved:__________________________________
Thesis Supervisor

____
Title and Department

____
Date

DATA ACQUISITION UNIT FOR LOW-NOISE, CONTINUOUS GLUCOSE MONITORING

by

Daniel Warren Cooley

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy degree
in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

May 2012

Thesis Supervisor: Professor David R. Andersen

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Daniel Warren Cooley

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Electrical and Computer Engineering at the
May 2012 graduation.

Thesis Committee: ___________________________________

David R. Andersen, Thesis Supervisor

____

Mark Arnold

____

Thomas Boggess

____

Anton Kruger

____

Hassan Raza

ACKNOWLEDGMENTS

I thank my family for consistent, positive support and advice as I worked on the PhD in order to update design skills and return to a research and development environment. Their consistent, positive moral support and encouragement helped me to continue while running photonic crystal simulations and bandgap calculations on a 24 hour per day, 7 day a week basis for over a year and during efforts to improve noise performance of the photocurrent measurement system.

I am indebted to Joo-Young Choi for her persistence and ability to record photocurrent spectra in the chemistry lab for extended periods of time. Joo-Young recorded spectra overnight without assistance even when it was offered and also applied the calibration method to the experimental data from the chemistry lab. Jue Qian also recorded spectra with the monitoring system in the chemistry lab.

I thank my advisor, David Andersen, for his broad range of research interests. Dave advises students on projects in physics, non-linear optics, theoretical electromagnetic calculations, embedded systems, and software engineering which helped during my research. Dave also shares some of my other interests including growing Hostas.

I thank the members of the examining committee for agreeing to serve on my committee and providing feedback on the comprehensive exam and dissertation.

I acknowledge several valuable discussions with Jon Olesberg regarding the glucose sensor and optical sensing techniques. Jon provided a glucose absorption spectrum recorded with an FTIR instrument. Jon also allowed me to utilize optical apparatus to align the IR LEDs with respect to the photodiode arrays and electronics equipment including low noise laboratory power supplies and thermo-electric controllers for long term periods which enabled me to record data in support of this thesis. Jon has a wide range of optical sensing and fabrication experience.

ABSTRACT

As the number of people with diabetes continues to increase, research efforts improving glucose testing methods and devices are under way to improve outcomes and quality of life for diabetic patients. This dissertation describes the design and testing of a Data Acquisition Unit (DAU) providing low noise photocurrent spectra for use in a continuous glucose monitoring system. The goal of this research is to improve the signal to noise ratio (SNR) of photocurrent measurements to increase glucose concentration measurement accuracy. The glucose monitoring system consists of a portable monitoring device and base station. The monitoring device measures near infrared (IR) absorption spectra from interstitial fluid obtained by microdialysis or ultrafiltration probe and transmits the spectra to a base station via USB or a ZigBee radio link. The base station utilizes chemometric calibration methods to calculate glucose concentration from the photocurrent spectra. Future efforts envision credit card-sized monitoring devices.

The glucose monitor system measures the optical absorbance spectrum of an interstitial fluid (ISF) sample pumped through a fluid chamber inside a glucose sensor. Infrared LEDs in the glucose sensor illuminate the ISF sample with IR light covering the 2.2 to 2.4 micron wavelength region where glucose has unique features in its absorption spectrum. Light that passes through the sample propagates through a linearly variable bandpass filter and impinges on a photodiode array. The center frequency of the variable filter is graded along its length such that the filter and photodiode array form a spectrometer. The data acquisition unit (DAU) conditions and samples photocurrent from each photodiode channel and sends the resulting photocurrent spectra to the Main Controller Unit (MCU). The MCU filters photocurrent samples providing low noise photocurrent spectra to a base station via USB or Zigbee radio link.

The glucose monitoring system limit of detection (LOD) from a single glucose sensor wavelength is 5.8 mM with a system bandwidth of 0.00108 Hz. The partial least squares and net analyte signal methods show the system standard error of prediction for glucose are 1.12 mM and 1.88 mM, respectively - useful for detection of hyperglycemia but slightly high for indication of hypoglycemia.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

According to the World Health Organization, 346 million people worldwide are afflicted with diabetes as of 2011[1]. Self monitoring of blood glucose (SMBG) performed a number of times per day provides a method to control glucose levels and limit the possibility of adverse outcomes due to diabetes. Despite the ever-increasing number of people with diabetes and consistent research efforts most commercially available SMBG systems require repeated, painful finger sticks, utilize electrochemical reactions with chemicals such as glucose oxidase, and suffer from limited accuracy[2]. Direct glucose measurement techniques exploit intrinsic features of the glucose molecule while indirect techniques such as electrochemical reactions and IR reflection spectroscopy are based on the effect glucose has on chemical, physical, or physiological features of the test sample[3]. IR reflection spectroscopy measuring scattering of skin tissue suffers from low selectivity to glucose as several constituents of interstitial fluid (ISF) modulate the refractive index of human skin[4]. IR transmission spectroscopy measures light absorption of the glucose molecule yielding a direct glucose monitoring method[3,4].

Early glucose testing methods, before the 1940's, had to be done in laboratories due to their complicated nature[5]. In the 1940's testing could be done in the home with tests that measured glucose in the urine. By the 1970's test strips impregnated with a chemical that changed color in proportion to the amount of glucose in the blood were available. This method was replaced by electrochemical tests performed by test strips, lancets, and portable electronic devices in the 1990's and is still used for SMBG today. Today research on developing an artificial pancreas to provide a type of cure for diabetes is underway but the main limitation in this effort is glucose sensing technology[5,6].

Recent research in electrochemical methods for glucose sensing describe fluorescent quantum dots[7], a fluorescent glucose detection sensor[8], development of a capillary-based sensor for glucose measurement in tear fluid[9], an electrochemical biosensor with a mediating chemical in a membrane[10], and use of carbon nanotubes in glucose sensors[11]. Research on the use of Raman spectroscopy for continuous glucose monitoring[12,13] exhibited continuous monitoring over 17 days with RMS errors of calibration and prediction of 3.6 mg/dL and 13.7 mg/dL in Ref. 12. Ref. 14 reported on a photonic crystal glucose sensor which changes size depending on glucose concentration, altering the wavelength of light diffracted by the crystal.

Recent research in infrared transmission spectroscopy evaluated glucose sensing methods using tunable laser diodes[15] and Fourier-transform infrared (FTIR) spectroscopy[16,17]. Heise et al[18] present a glucose monitoring system utilizing an FTIR instrument as part of a bedside monitoring system. We study low-noise photocurrent measurement techniques for the development of a portable, low cost, continuous glucose monitor. Our system consists of IR LEDs, photodiodes (PDs), and electronic circuitry custom designed and fabricated by our research group to enable development of a low cost, easily portable glucose monitoring device.

The measurement system[19,20,21] provides low-noise photocurrent spectra from a near-IR glucose sensor and calculates glucose concentration from IR absorption spectra with multivariate chemometric techniques[22,23]. The glucose sensor contains a narrow bandgap GaInAsSb common-cathode IR photodiode array, linearly variable bandpass filter, and 32-channel photodiode array and operates in the 2.2 to 2.4 micron wavelength range of the near-infrared spectrum where glucose has unique absorption features providing glucose specific information for multivariate determination of glucose concentration. The measurement system must minimize noise and drift in order to limit residual errors in the multivariate calibration process and ensure accurate glucose concentration results. Popular photocurrent measurement techniques include current input ADCs[24] as well as transimpedance amplifiers[25,26]. In our system TIAs translate photocurrent into voltages which are sampled by ADCs. Several factors inhibit attaining high SNR including recombination current in the GaInAsSb photodiodes, low photodiode shunt resistance, and limited amount of light impinging on the photodiode array. GaInAsSb photodiodes exhibit recombination current, a leakage current reducing the photodiode shunt resistance[27]. The low bandgap of IR photodiodes relative to typical Si photodiodes results in enhanced dark current[28] and hence reduced photodiode shunt resistance. Low shunt resistance in a TIA inhibits SNR for cases where thermal noise limits the SNR. Due to scattering losses in the sample and presence of the linearly variable filter, limited amounts of IR light impinge on the photodiode array producing low photocurrent (10 nA) levels and restricting SNR.

A series of lock-in amplifiers[29,30] implemented in software filter photocurrent samples and determine the amount of photocurrent detected by each channel. LIAs mix, or multiply, an input signal with a reference signal at a known frequency and low pass filter the mixer output. In our case the input signal and reference are in phase and of equal frequency and thus the mixer produces components at DC and twice the reference frequency. The low pass filter passes only the DC component which is proportional to the amplitude of the component of the input signal at the reference frequency. Dorrington and Kunnemeyer[31] presented a lock-in amplifier filtering method combining the mixing and low pass filtering operations into a pair of difference equations applied alternately to the stream of input samples. This lock-in method allows the use of a higher lock-in reference frequency, ameliorating the effect of operational amplifier input voltage noise on system SNR when op amp 1/f noise dominates op amp input voltage noise.

A block diagram of the glucose measurement system is shown in Fig. 1-1. The glucose sensor[6], located on the Data Acquisition Unit (DAU), has two LEDs[32] which provide light covering the combination region of the mid IR spectrum. A linearly variable filter and 32 element photodiode array form a spectrometer operating in the combination region of the IR spectrum. The DAU contains TIAs which convert photocurrents into voltages which are sampled by 24-bit Delta-Sigma ADCs. An SPI port transfers photocurrent samples to the MCU which processes the ADC samples and sends spectra to the base station at 1 Hz for display and further processing. The MCU provides ADC control signals and two LED control lines to the DAU.

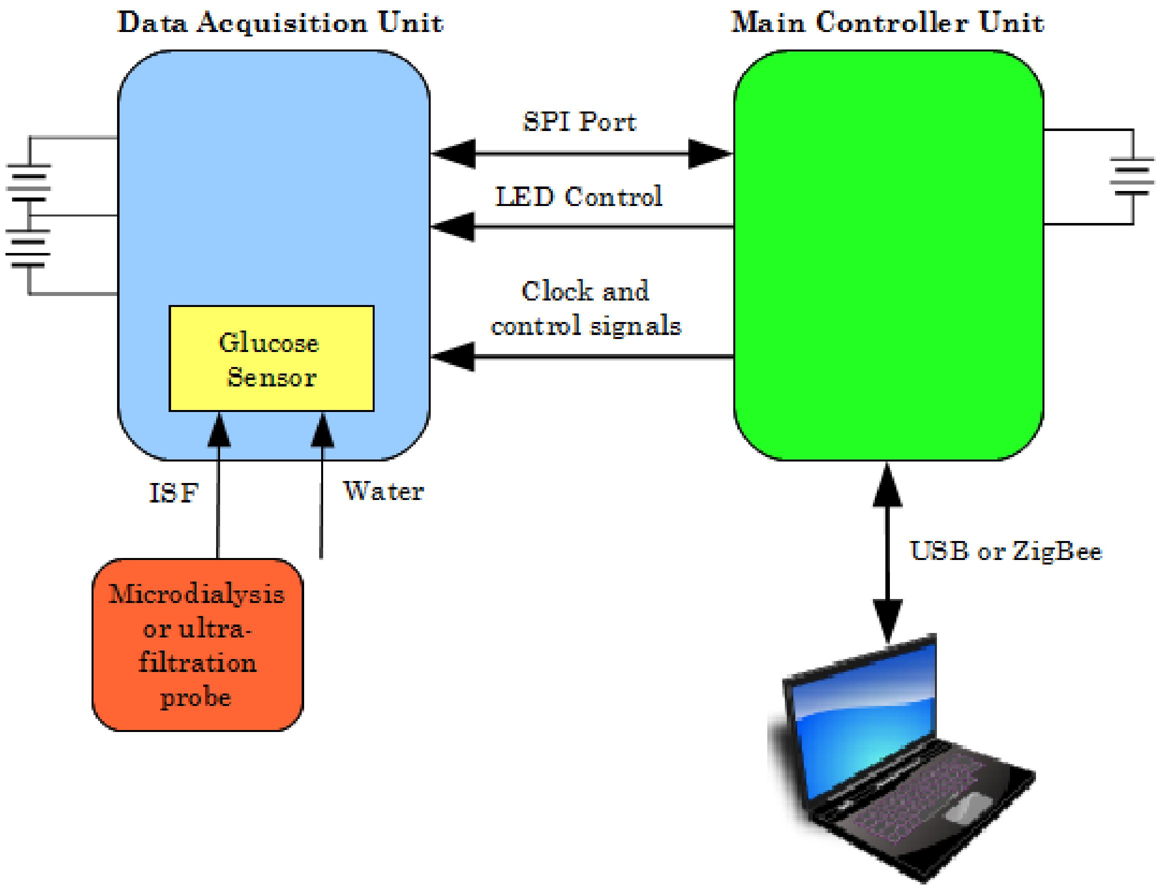


Figure 1-1. Block diagram of the glucose monitoring system.

Overall technical requirements for the DAU/MCU:

- The shunt resistance of IR photodiodes in the glucose sensor is approximately 10 kΩ to 30 kΩ.
- SNR > 40 dB for a 30 kΩ shunt resistance photodiode array.
- The MCU will be held horizontally with the DAU connected to the MCU and held vertically placing the long axis of sensor header in the vertical direction. This prevents air bubbles, if present, from collecting in the fluid chamber within the glucose sensor and interfering with measurements.
- The glucose sensor will be installed on the DAU with a 40 pin DIP ZIF socket.
- LED current must be adjustable in the range of 100 to 200 mA DC.
- The IR LEDs are common cathode and use 3 pin 0.1″ spacing connector.
- Four types of photocurrent measurements are required in succession: (1) LED1 on/LED2 off, (2) both LEDs off, (3) LED1 off/LED2 on, and (4) both LEDS off.
- The measurement device must originate its power from batteries.
- The MCU communicates with the base station via either USB or a ZigBee radio link.

Chapter 2 discusses theory of operation of the glucose sensor. Chapter 3 presents electrical noise theory and mathematical models predicting signal to noise ratio for several photocurrent measurement methods. Chapter 4 details the DAU hardware and software design. Chapter 5 documents assessment of monitoring system ability to provide low noise photocurrent spectra and Chapter 6 focuses on application of the DAU and MCU to continuous glucose sensing. Chapter 7 concludes the paper and discusses possible future extensions of this work.

CHAPTER 2

GLUCOSE SENSOR THEORY OF OPERATION

2.1 Glucose Sensor

The glucose monitoring device measures the absorbance of light in the combination region of the near-IR spectrum. The glucose sensor, Fig. 2-1, consists of two IR LEDs, two glass fluid chambers with square cross-section, a linearly variable bandpass filter, and a photodiode array. The IR LEDs emit light covering the 2.2 to 2.4 μm wavelength range and use backside geometry so that light is emitted directly into the fluid chamber. There are three regions in the near infrared spectrum where vibrations from glucose molecules can be detected: the short wavelength region (14286-7300 cm^{-1}), the first overtone region (6500 – 5500 cm^{-1}) and the combination region (5000 – 4000 cm^{-1})[3]. Features in the near-IR vibrational spectrum become weaker and broader as the wavelength decreases thus the combination region features are easier to detect. Since water absorbs light and tissues scatter light, there are optimum sample thicknesses for these three near-IR regions: combination region (1 mm), first overtone region (5 mm), and short wavelength region (10 mm). The glucose sensor used in this research operates in the combination region to take advantage of the relatively sharp absorption peaks of glucose for longer wavelengths. The optical path length through the fluid chamber is 0.8 mm, approximately the optimum path length for the combination region.

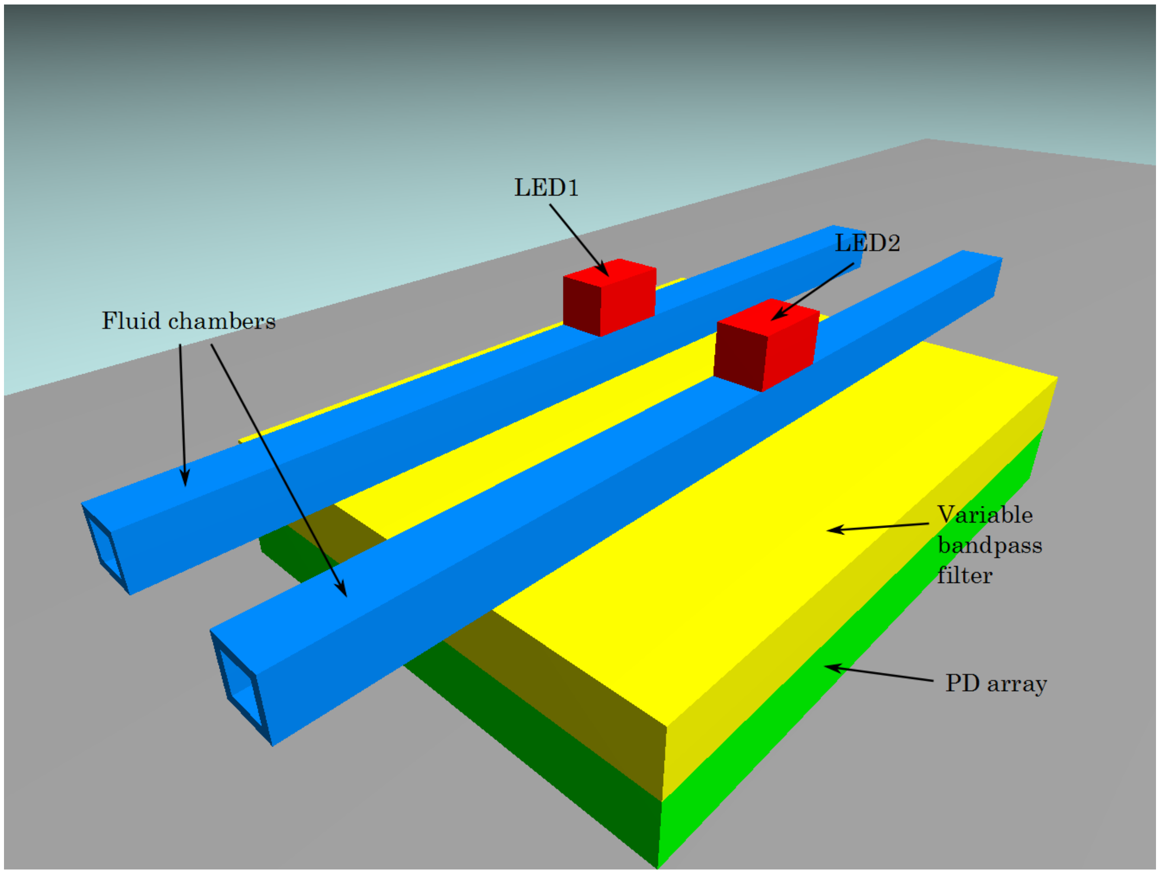


Figure 2-1. The glucose sensor. Two IR LEDs emit light into the fluid chambers which propagates through the chambers and variable filter before reaching the PD array.

A sample of ISF flows through one fluid chamber and a blank sample flows through the other fluid chamber. One LED shines light through each fluid chamber – the sensor measures absorption spectra of the blank and the fluid sample allowing multivariate, chemometric mathematical techniques to extract glucose concentration from the two spectra. These math techniques require low noise spectra in order to calculate analyte concentration with low uncertainty. The sensor mates to the PC board with a 40 pin DIP integrated circuit header, with pin configurations shown in Fig. 2-2.

The intensity, I, of monochromatic light as it passes through a material is

$$I = I_0 10^{-\alpha}, \tag{2.1}$$

where I_0 is the initial light intensity and α is the absorbance. The Beer-Lambert law states that

$$\alpha = \varepsilon l c, \tag{2.2}$$

where ε is the molar absorptivity, l is the optical path length, and c is the concentration of the absorbing medium. Equation (2.1) enables measurement of absorbance and hence concentration for a constant optical path length from measurements of light intensity before and after passing through the absorbing medium.

2.2 PN Diode and PIN Photodiode

A typical pn diode is shown in Fig. 2-3(a) and its energy band diagram with zero bias is in Fig. 2-3(b)[33]. The left, p type, portion of the diode is doped with N_A acceptor atoms/m^3 and the right, n type, region is doped with N_D donor atoms/m^3. At the PN junction mobile holes from the p region and mobile electrons from the n region diffuse across the junction leaving a depletion region at the center of the diode from $-x_p$ to x_n. This displacement of charge creates a potential difference between the p and n regions called the built in potential, V_{bi}, which sweeps electrons in the depletion region towards the n region and sweeps holes in the depletion region towards the p region.

The IR photodiodes in the glucose sensor are PIN type diodes, Fig. 2-4(a),

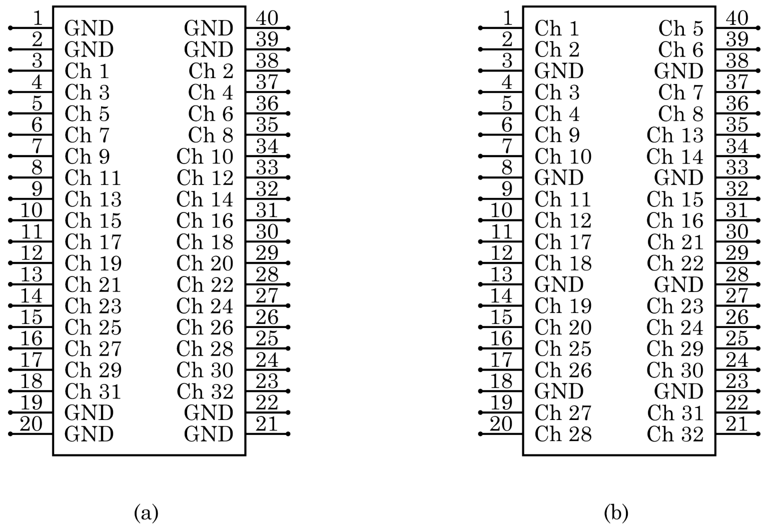


Figure 2-2. Glucose sensor pin configurations. (a) Original pin configuration. (b) New pin configuration with interspersed ground pins.

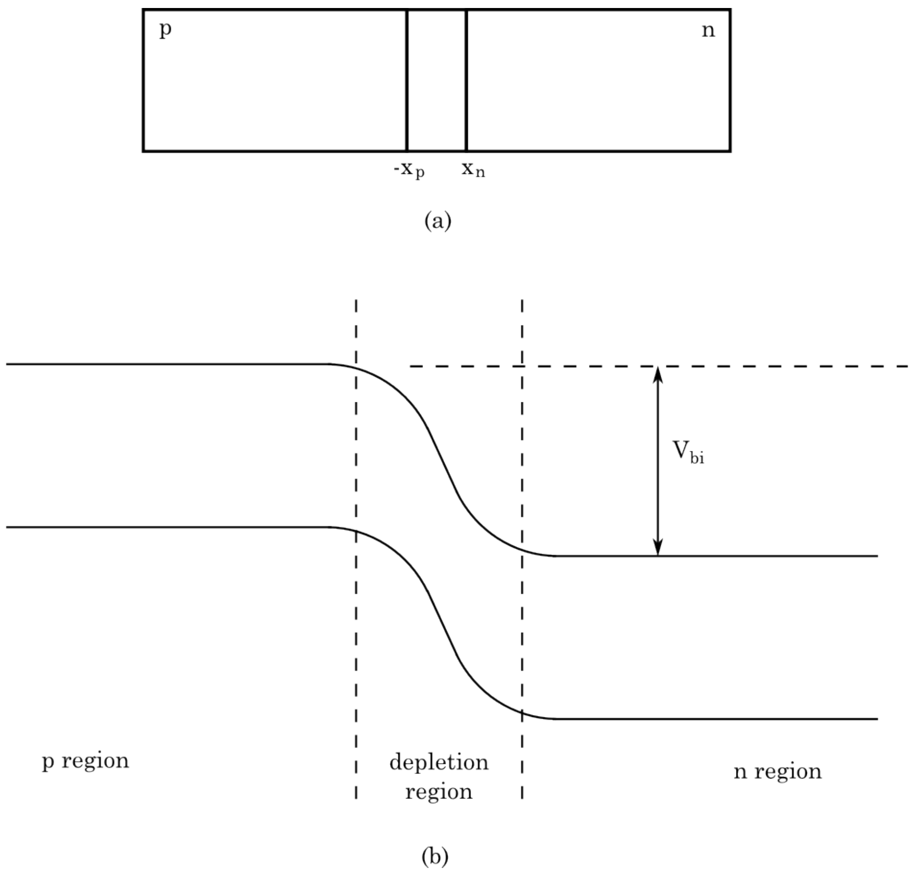


Figure 2-3. PN diode. (a) Drawing of diode. (b) Energy band diagram.

with a thin p region and an intrinsic or lightly doped region (i region) of thickness W_i between the p and n layers. Photons which pass through the top p layer and enter the i region can create an electron-hole pair, see Fig. 2-4(b). The electron moves to the n region and the hole moves to the p region creating photocurrent. The p region is kept very thin to allow light to pass into the i region because carriers generated by a photon absorbed in the p region do not contribute to the photocurrent as there is no potential difference there to move the carriers to the diode terminals. If a photon absorbed in the i region has energy greater than the bandgap energy,

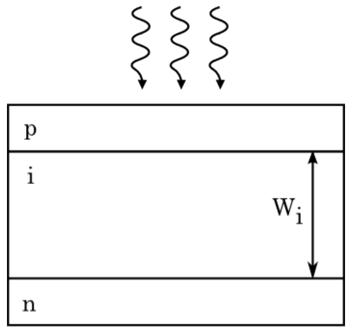
$$E_{gap} = hf, \tag{2.3}$$

where h is Planck's constant and f is the photon frequency, it can create an electron-hole pair. This sets a low frequency bound and upper wavelength bound on the PD absorption spectrum. The inverse of the absorption coefficient, $1/\alpha$, is approximately the average penetration depth of light in the material. Choosing the intrinsic material thickness, W_i, equal to $1/\alpha$ sets a lower wavelength limitation on the photodiode absorbance spectrum.

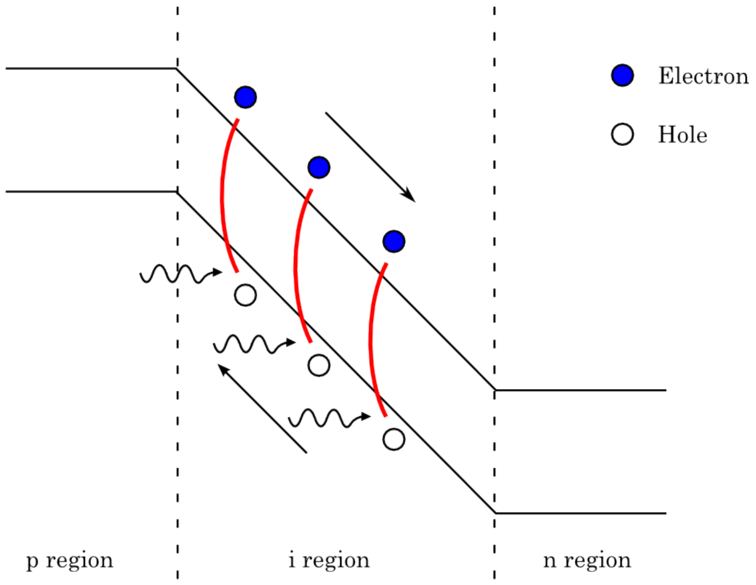
2.3 Glucose Sensor IR Photodiode Characteristics

The photocurrent design goal of the glucose sensor is 10 nA of photocurrent. The maximum photocurrent from one typical photodiode, the S1133, is 100 mA, 10,000 times greater than the IR photodiodes when installed in the sensor[34]. Since the SNR is

$$SNR = 10\log_{10}\left(\frac{I_{PC}R_F}{\sigma}\right), \tag{2.4}$$



(a)



(b)

Figure 2-4. PIN photodiode. (a) Drawing of diode. (b) Energy band diagram.

where σ is the sample standard deviation, low photocurrent directly suppresses system SNR. The sensor geometry allows limited amounts of light to pass through the sample to the variable filter and only a portion of that light will be transmitted through the filter to the photodiodes. In this research we calculate the signal to noise ratio in deciBels, Eq. 2.4, using 10 log because 1 dB is defined as 10 times the base 10 logarithm of a power ratio and photodiode photocurrent,

I_{PC}, is proportional to the luminosity or power per unit area reaching the photodiode.

The I-V characteristic curve of the IR photodiodes in the glucose sensor differs from that of typical Si photodiodes. The IR photodiodes have a lower reverse breakdown voltage, larger reverse saturation current, and lower shunt resistance than typical Si photodiodes - see Table I for a comparison of these quantities for the glucose sensor IR diodes as installed in the sensor and the S1133, a typical Si photodiode. The I-V characteristic curve for the IR photodiodes is shown in Fig. 2-5 and the I-V curve near the origin is in Fig. 2-6. The large reverse saturation current and low reverse saturation voltage alter the I-V characteristic such that with a very small bias dark current easily overcomes the photocurrent and enhances shot noise. Thus the TIA cannot apply a bias

Table I. A comparison of the IR photodiodes as installed in the glucose sensor and the S1133.

Device	IR Photodiode	S1133
Reverse Breakdown Volt.	1 – 2 V	10 V
Reverse Saturation Current	100 μA	15 pA @-10V
Shunt Resistance	10 kΩ to 30 kΩ	100 GΩ

voltage across the photodiodes.

The shunt resistance of the IR photodiodes is approximately 10 kΩ to 30 kΩ, much lower than the 100 GΩ shunt resistance of the S1133. IR PDs have lower shunt resistance due to recombination current, a leakage current in parallel with R_{PD},[27] and due to their low bandgap. The reduced bandgap results in enhanced dark current[28] and hence reduced shunt resistance. When a photon has energy equal to the bandgap energy,

$$\lambda_{GAP} = \frac{c}{f} = \frac{hc}{E_{GAP}}. \tag{2.5}$$

Since IR photodiodes operate at longer wavelengths than Si PDs their bandgap energy is lower than Si PDs. The reverse saturation current of a diode is given by

$$I_o = qA\left(\frac{D_N n_i^2}{L_N N_A} + \frac{D_P n_i^2}{L_P N_D}\right).^{33} \tag{2.6}$$

Under equilibrium,

$$E_{GAP} = qV_{bi} = kT \ln\left(\frac{N_A N_D}{n_i^2}\right). \tag{2.7}$$

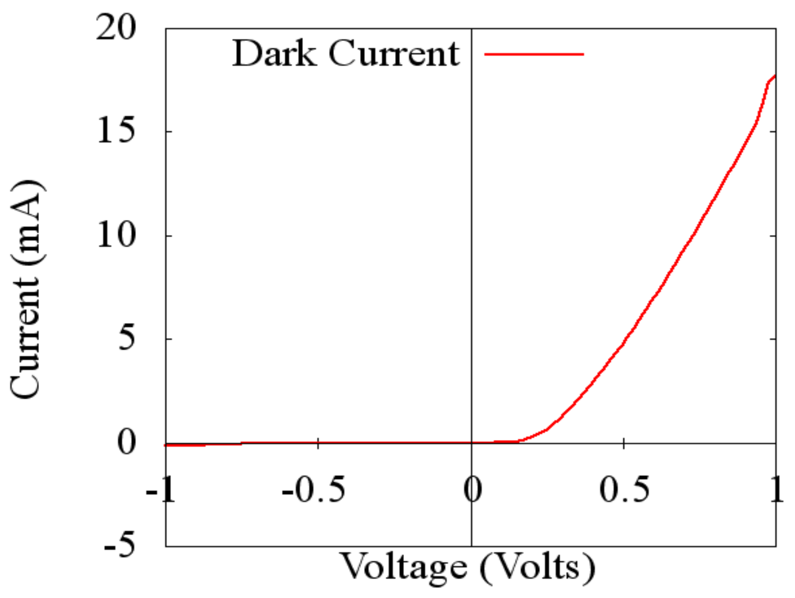


Figure 2-5. IR photodiode I-V characteristic curve[35].

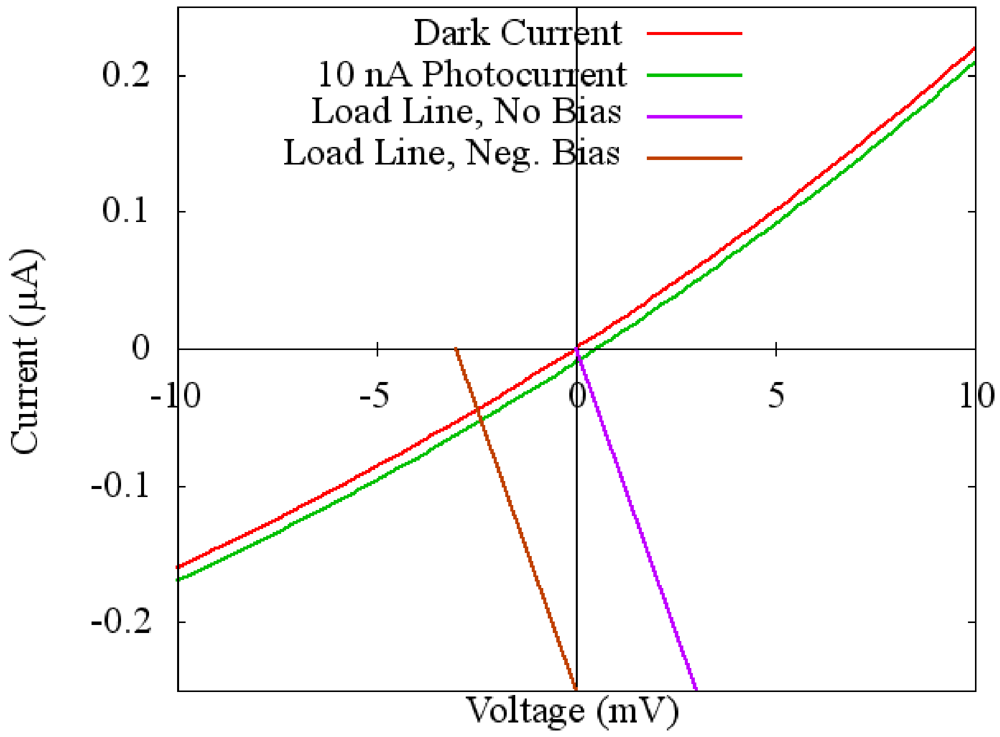


Figure 2-6. IR photodiode characteristic curves near the origin[35].

Solving Eq. 2.7 for n_i^2 we find

$$n_i^2 = N_A N_D e^{-E_{GAP}/kT} . \tag{2.8}$$

Substituting Eq. 2.8 into Eq. 2.6 shows

$$I_o = qA\left(\frac{D_N N_D}{L_N} + \frac{D_P N_A}{L_P}\right) e^{-E_{GAP}/kT} , \tag{2.9}$$

indicating IR PDs have exponentially higher reverse saturation current than Si PDs. The slope of the ideal diode equation,

$$I = I_o\left(e^{qV/kT} - 1\right), \tag{2.10}$$

at the origin is 1/R$_{SHUNT}$. Differentiating Eq. 2.10 with respect to voltage we find that

$$\frac{dI}{dV}=I_0 e^{qV/kT}\frac{q}{kT}. \tag{2.11}$$

Evaluating Eq. 2.11 at V=0 we find the result

$$R_{Shunt}=\frac{kT}{I_0 q} \tag{2.12}$$

showing lower shunt resistance for IR PDs due to elevated reverse leakage current.

2.4 IR LED Design

LEDs in the glucose sensor provide illumination covering the 2.2 to 2.4 micron wavelength region. The selection of materials for p and n type regions determines the LED emission spectrum. The materials forming the LED must have equal lattice constants to minimize mechanical strain in the structure as the LEDs are built in layers upon a substrate material. The designer chooses the material composition for the desired bandgap matching the lattice constant of the substrate. Figure 2-7 shows a plot of bandgap wavelength versus lattice constant for several semiconductor materials including GaSb, the LED substrate material for the glucose sensor. Material compositions on a horizontal line through the GaSb point in Fig. 2-7 match the lattice constant of GaSb and result in high quality devices. The glucose sensor LEDs utilize material composition with 21% indium, 79% gallium, and arsenic concentration to lattice match with GaSb for an upper wavelength limit of 2.5 microns.

The frequency of the peak LED emission spectrum, ν_p, is given by

$$h\nu_p=E_G+\frac{k_B T}{2}, \tag{2.13}$$

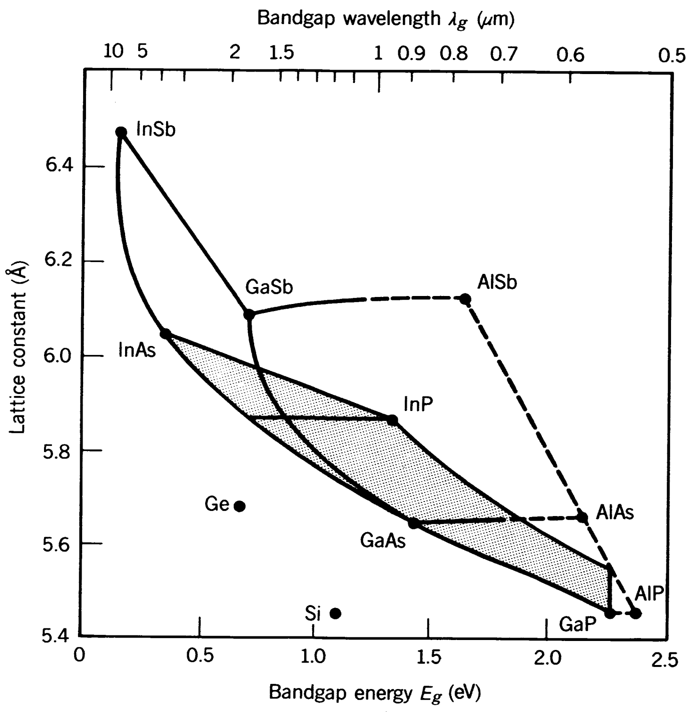


Figure 2-7. Plot of lattice constant versus wavelength for several semiconductor materials.[36]

where E_G is the semiconductor bandgap energy. The full width at half maximum of LED emission in Hz is given by

$$\nu = \frac{1.8 k_B T}{h} \tag{2.14}$$

where h is Planck's constant.

2.5 Summary

The glucose sensor consists of a spectrometer formed by the linearly variable filter and photodiode array with illumination from two IR LEDs. The glucose monitor system measures the absorbance spectrum of ISF in the combination region of the near IR spectrum where glucose has unique light absorption peaks. The SNR goal for photocurrent measurements is 40 dB for a 30 kΩ shunt resistance PD array. We cannot reverse bias the photodiodes because the shot noise from the dark current would dominate the photocurrent shot noise and degrade SNR. The photodiodes provide relatively low photocurrent when installed in the sensor due to sensor geometry. IR photodiodes have reduced shunt resistance due to recombination current of GaInAsSb devices and the reduced bandgap of IR PDs. Careful choice of LED material composition ensures the IR LED emission covers the 2.2 to 2.4 micron wavelength range.

CHAPTER 3

PHOTOCURRENT MEASUREMENT METHODS

3.1 Background

Popular photocurrent measurement methods include sampling current with current input ADCs and translating current to a voltage with transimpedance amplifiers for subsequent sampling with standard voltage input ADCs. A current input ADC samples the current flowing into its input pins. Current input ADCs provided photocurrent measurements in the original configuration of the glucose monitoring system[24]. In Ref. 31 researchers designed a system using a lock in amplifier and current input ADC to measure currents in the pA range. The lock in amplifier used spectral reversal to remove undesired low frequency noise.

The most popular photocurrent monitoring method utilizes the TIA, see Fig. 3-1, to converts an input current to a voltage. The output voltage of the TIA is

$$E_O = -I_{PC}R_F, \tag{3.1}$$

where I_{PC} is the photocurrent and R_F is the feedback resistor. Designers typically use the TIA for optical receivers where maximum bandwidth is desired to transmit data at high speed - several methods for attaining high bandwidth with the TIA are available[26].

The remainder of the chapter documents development of photocurrent

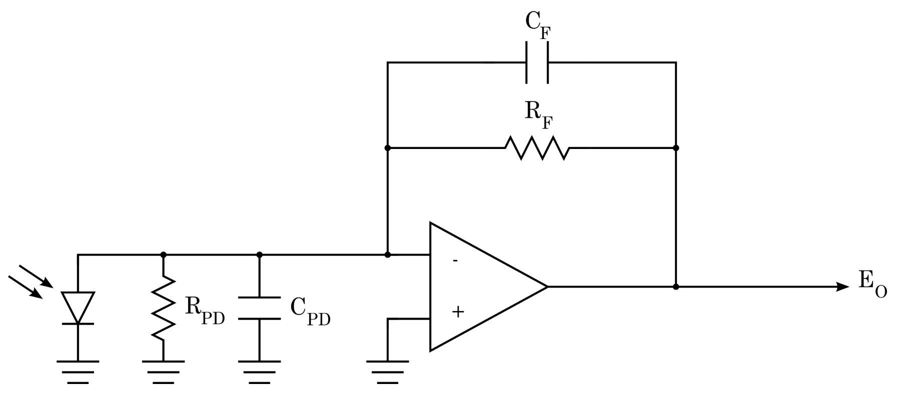


Figure 3-1. Transimpedance amplifier schematic with photodiode capacitance and feedback capacitor.

measurement methods utilizing the TIA to translate photocurrent into a voltage for measurement with an ADC. A lock in amplifier implemented in firmware provides low noise measurement of photocurrent spectra. Analysis of noise sources in the TIA and photodiode and spectral analysis of filtering methods employed by the measurement system yields a noise model predicting noise characteristics of the system.

3.2 Noise Theory

Two types of noise affecting electrical circuits are thermal noise and shot noise[37]. The thermal noise voltage of a resistor R appears in series with the resistor and is given by

$$E_T = \sqrt{4\mathrm{kTR}\,\Delta f}\,, \tag{3.2}$$

where k is Boltzmann's constant, T is temperature in Kelvin, and Δf is the double sided measurement bandwidth. The thermal noise may also be transformed into a current source of value E_T / R in parallel with the resistor.

Many discrete electrons comprise current flowing through a conductor. Each electron passing a potential barrier in a circuit causes a small burst of current and this causes noise called shot noise. The shot noise for I Amps of DC current is

$$I_{Shot}=\sqrt{2\mathrm{eI}\,\Delta f}\,, \tag{3.3}$$

where e is the electronic charge and Δf is the single sided measurement bandwidth.

A noise source has many different frequency components and its amplitude and phase vary randomly. Thus when two uncorrelated noise sources with different instantaneous frequency and magnitude are connected together in series the power from the two noise sources cannot combine constructively or destructively and the resulting total power is the sum of the two powers. Since we add the power from two sources to find total power, to add two voltage noise sources V_1 and V_2 we add them using the sum of squares fashion:

$$V_{Total}^2=V_1^1+V_2^2. \tag{3.4}$$

The bandwidth of a low pass filter or amplifier, f_{3dB}, is the frequency where the output power drops to half the maximum output power. Noise above the 3 dB bandwidth gets attenuated but still passes through the system. The equivalent noise bandwidth of a system accounts for the additional noise above the 3 dB bandwidth. The equivalent noise bandwidth of a system is the bandwidth of a rectangular power gain spectrum with area equal to the area under the systems power gain spectrum and magnitude equal to the systems maximum gain, see Fig. 3-2. For a one pole low pass filter with response

$$A_v(f) = \frac{1}{1 + if / f_{3\mathrm{dB}}}, \tag{3-5}$$

where $f_{3\mathrm{dB}}$ is the 3 dB bandwidth, the single sided equivalent noise bandwidth is[37]

$$\Delta f = \frac{1}{A_{vo}^2} \int_0^\infty |A_v(f)|^2 df = \frac{\pi}{2} f_{3\mathrm{dB}}, \tag{3-6}$$

where A_{vo} is the DC voltage gain of the filter.

3.3 Digital Filtering

3.3.1 Single Pole Low Pass Filter

An RC low pass filter attenuates frequencies above its characteristic frequency $f = 1/(2\pi RC)$. We use the following process to implement the transfer function of a single pole RC filter,

$$H(s) = \frac{\frac{1}{sC}}{R + \frac{1}{sC}} = \frac{1}{1 + sRC}, \tag{3.7}$$

in the digital domain. After application of the bilinear transform to Eq. 3.7 we find

$$H(z) = H\left(\frac{2}{T}\frac{z-1}{z+1}\right) = \frac{1}{1 + \frac{2}{T}\left(\frac{z-1}{z+1}\right)RC}, \tag{3.8}$$

where T is the sample period[38]. Eq. 3.8 can be simplified to

$$H(z) = A\left(\frac{1 + z^{-1}}{1 + Bz^{-1}}\right), \tag{3.9}$$

where

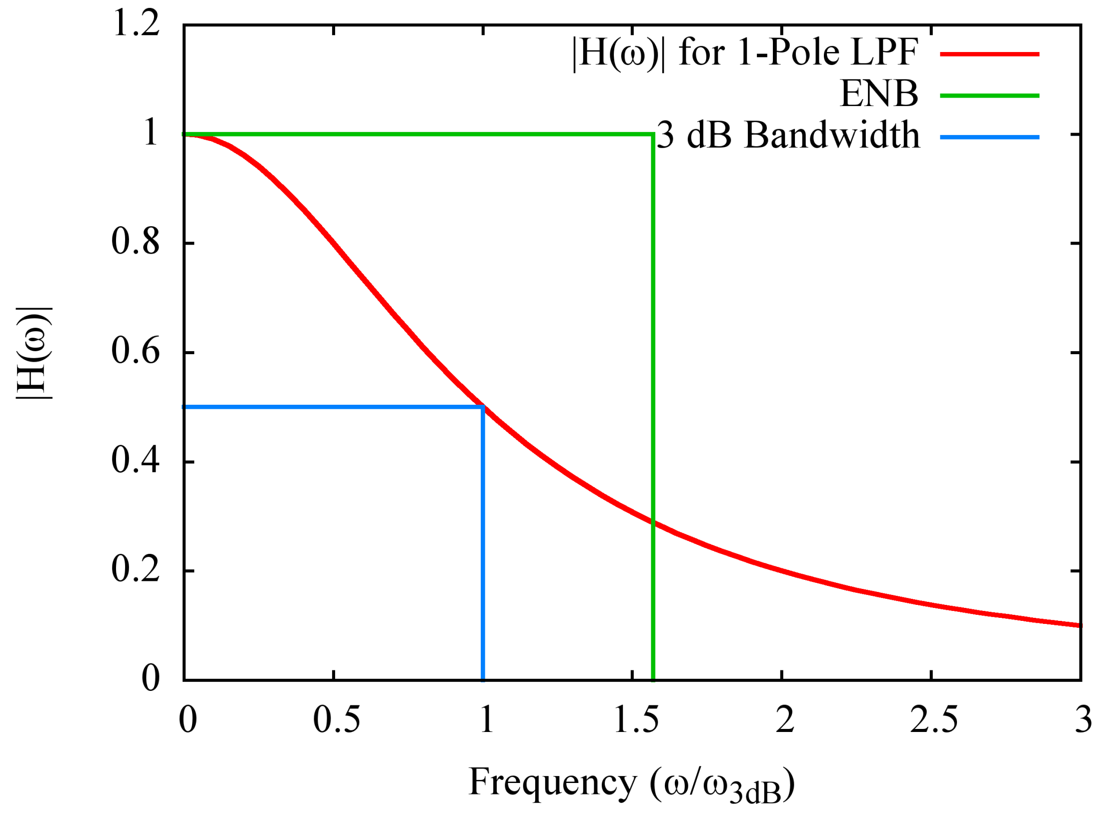


Figure 3-2. Definition of equivalent noise bandwidth for a single pole low pass filter.

$$A = \frac{1}{1 + \frac{2}{T} RC}, \tag{3.10}$$

and

$$B = \frac{1 - \frac{2}{T} RC}{1 + \frac{2}{T} RC}. \tag{3.11}$$

The transfer function in the z domain, Eq. 3.9, is in the form of

$$H(z) = \frac{Y(z)}{X(z)} = \frac{AN(z)}{1 + D(z)} \tag{3.12}$$

with

$$N(z)=1+z^{-1} \tag{3.13}$$

and

$$D(z)=B\,z^{-1}. \tag{3.14}$$

From Eq. 3.12,

$$Y(z)(1+D(z))=AX(z)N(z), \tag{3.15}$$

and it follows that

$$Y(z)=AX(z)N(z)-Y(z)D(z). \tag{3.16}$$

Substituting expressions for N(z) and D(z) and Eq. 3.16 we find

$$Y(z)=AX(z)(1+z^{-1})-BY(z)z^{-1}\ . \tag{3.17}$$

A difference equation representing Eq. 3.17 is

$$Y_n=A(X_n+X_{n-1})-BY_{n-1}\ . \tag{3.18}$$

Inserting equations for A and B into Eq. 3.18 gives

$$Y_n=\left(\frac{1}{1+\frac{2}{T}RC}\right)(X_n+X_{n-1})-\left(\frac{1-\frac{2}{T}RC}{1+\frac{2}{T}RC}\right)Y_{n-1}\ . \tag{3.19}$$

In order to implement this filter digitally the filter coefficients must be integers, and forcing the coefficients to be powers of two allows implementing the filter using simple binary shift operations. Requiring

$$1+\frac{2}{T}RC=2^m \tag{3.20}$$

for an integer m, the filter difference equation becomes

$$Y_n = \left(\frac{1}{2^m}\right)(X_n + X_{n-1}) - \left(\frac{2-2^m}{2^m}\right)Y_{n-1} \ , \tag{3.21}$$

or

$$Y_n = \left(\frac{1}{2^m}\right)(X_n + X_{n-1}) + Y_{n-1} - \left(\frac{1}{2^{m-1}}\right)Y_{n-1} \ . \tag{3.22}$$

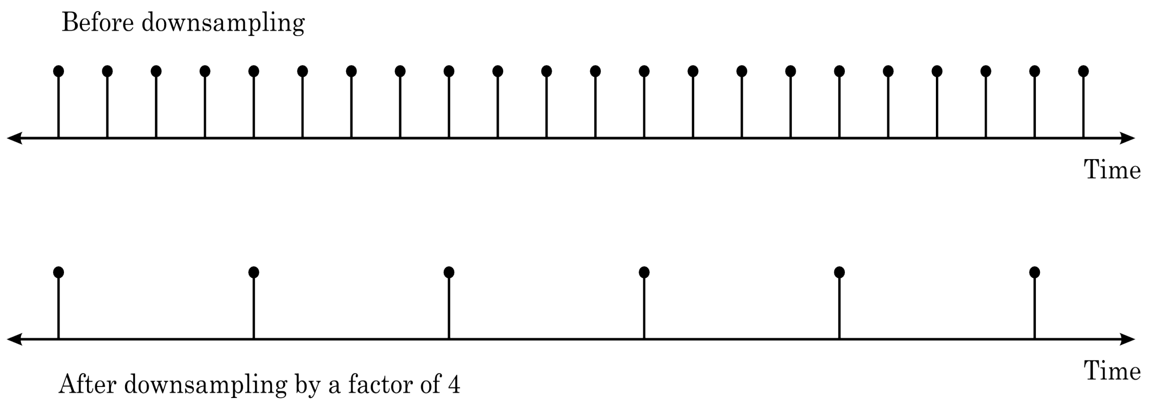
Eq. 3.22 consists of addition, subtraction, and right shift operations which are easily implemented with microprocessors.

3.3.1 Decimation

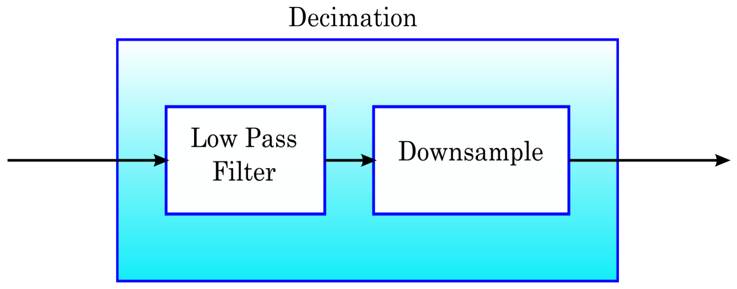
Decimation reduces the sample rate of a set of equally spaced samples by a factor of n. Decimation includes two processes: low pass filtering and downsampling. Downsampling by a factor of n selects every nth sample and discards the remaining samples, see Fig. 3-3(a). The Nyquist theorem says that the highest frequency that can be represented with a sampling frequency of f_{Sample} is $(f_{Sample})/2$. If the sampling frequency before downsampling is f_{Sample}, the sample frequency after downsampling is $(f_{Sample})/n$ and downsampled data must have bandwidth below $(f_{Sample})/2n$ to prevent aliasing requiring bandwidth reduction to less than or equal to $(f_{Sample})/2n$ before downsampling. Decimation, Fig. 3-3(b), includes the low pass filtering and downsampling operations.

3.3.3 Lock-in Amplifiers

A two phase lock in amplifier, see Fig. 3-4, measures the real and imaginary parts of a sinusoid input signal. The real component is in phase with the reference and the imaginary component is $\pi/2$ out of phase with the reference. One mixer multiplies the input signal,



(a)



(b)

Figure 3-3. Downsampling and decimation. (a) Downsampling. (b) Decimation.

$$V_{in} = A_{in}\cos(2\pi f_{in}t + \theta_{in}), \tag{3.23}$$

by the reference,

$$V_{ref} = A_{ref}\cos(2\pi f_{ref}), \tag{3.24}$$

to obtain the mixer output

$$V_{mixer1} = A_{in}A_{ref}\cos(2\pi f_{in}t + \theta_{in})\cos(2\pi f_{ref}). \tag{3.25}$$

Using the trigonometric identity

$$\cos(a)\cos(b) = \frac{1}{2}\cos(a+b) + \frac{1}{2}\cos(a-b) \tag{3.26}$$

Eq. (3.25) becomes

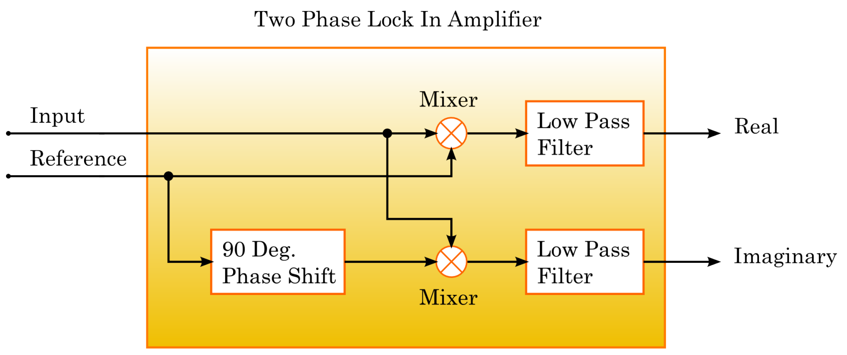


Figure 3-4. Two phase lock-in amplifier.

$$V_{mixer1} = \frac{1}{2} A_{in}A_{ref}\cos(2\pi(f_{in} + f_{ref})t + \theta_{in}) + \frac{1}{2} A_{in}A_{ref}\cos(2\pi(f_{in} - f_{ref})t + \theta_{in}). \tag{3.27}$$

With the assumption that $f_{in} = f_{ref}$, we find

$$V_{mixer1} = \frac{1}{2} A_{in}A_{ref}\cos(4\pi f_{ref}t + \theta_{in}) + \frac{1}{2} A_{in}A_{ref}\cos\theta_{in}\,. \tag{3.28}$$

The low pass filter allows the DC component to pass so that the 'Real' signal in Fig. 3-4 is

$$\text{Real} = \frac{1}{2} A_{in}A_{ref}\cos\theta_{in}. \tag{3.29}$$

Thus the lock in amplifier shifts the component of the input signal at the reference frequency to DC as shown in Fig. 3-5. To measure the imaginary portion of the input signal the reference frequency shifted by $\pi/2$,

$$V_{ref2} = A_{ref}\cos(2\pi f_{ref}t + \pi/2), \tag{3.30}$$

is mixed with the input yielding the second mixer output,

$$V_{mixer2} = A_{in}A_{ref}\cos(2\pi f_{in}t + \theta_{in})\cos(2\pi f_{ref}t + \pi/2). \tag{3.31}$$

Using the trigonometric identity, Eq. 3.26, the second mixer output becomes

$$V_{mixer2} = \frac{1}{2} A_{in}A_{ref}[\cos(2\pi(f_{in}+f_{ref})t+\theta_{in+}\pi/2)+\cos(2\pi(f_{in}- f_{ref})t+\theta_{in}-\pi/2)]. \tag{3.32}$$

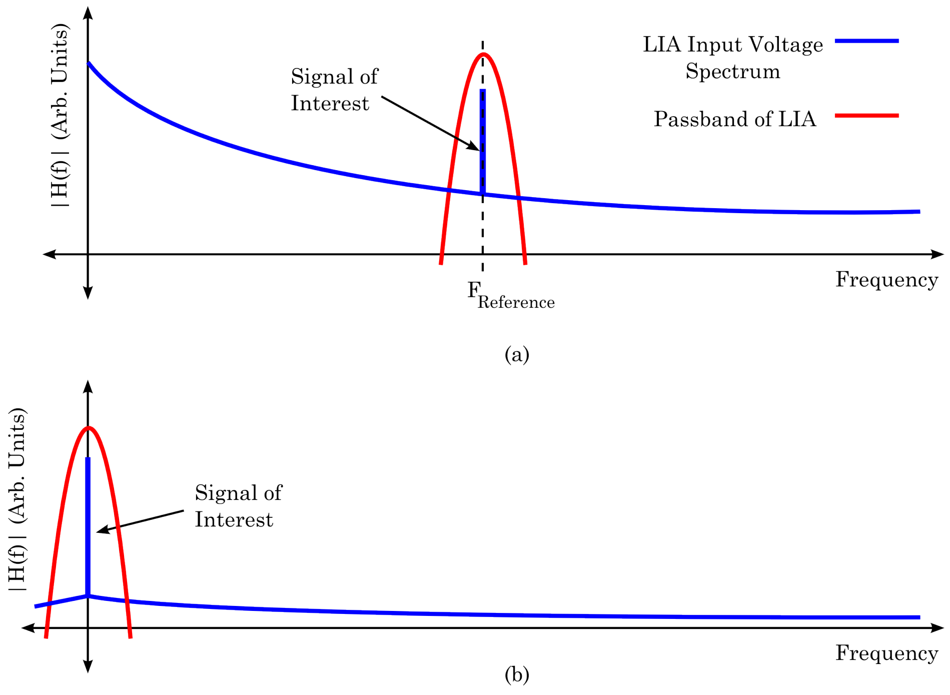


Figure 3-5. Qualitative description of lock-in amplifier operation. The signal at the reference frequency in (a) is shifted to DC as shown in (b) and undesired frequency components are attenuated by a low pass filter.

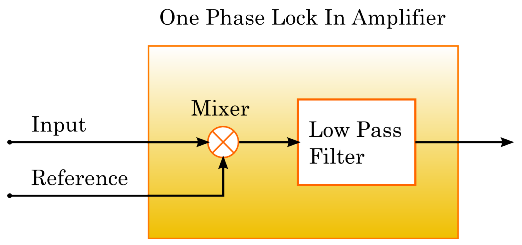


Figure 3-6. Single phase lock-in amplifier.

Assuming that $f_{in} = f_{ref}$, and applying the low pass filter we find the Imaginary output,

$$\text{Imaginary} = \frac{1}{2} A_{in}A_{ref}\cos(\theta_{in} - \pi/2) = \frac{1}{2} A_{in}A_{ref}\sin\theta_{in}. \qquad (3.33)$$

The single phase lock-in amplifier, Fig. 3-6. has only one mixer and one low pass filter. The single phase LIA output is equivalent to Eq. 3.29. With the additional assumption that

$$\theta_{in} = 0, \tag{3.34}$$

i.e., the input is in phase with the reference, the single phase LIA output is

$$\mathrm{LIA}_{\mathrm{Output}} = \frac{1}{2}\mathbf{A}_{in}A_{ref}. \tag{3.35}$$

3.4 Noise Model

3.4.1 Shot Noise

The noise model includes three sources of shot noise: photocurrent, current through R_{Shunt} due to op amp offset voltage (V_{IO}), and op amp input offset current (I_{BI}). Table II lists expressions for each of these noise sources. The shot noise sources are uncorrelated so the total shot noise current is

$$I_{Shot,Total} = \sqrt{I^2_{Shot,PC} + I^2_{Shot,EIO} + I^2_{Shot,IBI}}. \tag{3.36}$$

These noise currents flow through R_F creating the noise voltage

$$V_{Shot} = I_{Shot,Total} R_{F.} \tag{3.37}$$

at the TIA output.

Table II. Sources of shot noise considered by the noise model.

Shot Noise Source	**Value, Amps**
Photocurrent, I_{PC}	$\sqrt{2eI_{PC}\Delta f}$
E_{IO}	$\sqrt{2e(E_{IO}/R_{PD})\Delta f}$
I_{BI}	$\sqrt{2eI_{BI}\Delta f}$

3.4.2 ADC Noise

Sampling the TIA output with an ADC brings about ADC sampling noise and reference voltage noise. Successive approximation (SAR) ADCs and sigma-delta ADCs have very different noise characteristics. Standard SAR converters exhibit quantization noise due to the error from representing the input signal with a finite number of equally spaced bits. The voltage error for one sample for an ideal converter varies between zero and one LSB. A rough estimate of the standard deviation of this error is 1/6 LSB. Table III shows the quantization noise SNR for a standard SAR ADC with signal voltage 0.1V and voltage reference 4.096V using

$$SNR_{Quant\,,\,SAR} = 10\log\left(\frac{0.1\text{V}}{1/6\,LSB}\right). \tag{3.38}$$

where LSB = $V_{Ref}/2^N$. Calculations of quantization noise for several converter resolutions are shown in Table III, which shows that 20 bits are required to keep the quantization noise well below the desired DAU SNR of 40 dB for a standard SAR converter.

Table III. Quantization noise in dB for a standard SAR ADC. SNR is calculated using 10 log SNR.

ADC Resolution (Bits)	SAR ADC
12	28.2
16	40.2
20	52.3
24	64.3

Sigma-delta converters[39] exhibit reduced quantization noise compared to SAR converters. Sigma-delta converters increase the sampling rate, F_{Sample}, by the oversampling ratio, N, increasing the Nyquist rate to $NF_{Sample}/2$. The sigma-delta modulator moves quantization noise from the DC to $F_{Sample}/2$ range into higher frequencies where digital filters in the converter attenuate it before downsampling to F_{Sample}. Figure 3-7 depicts a block diagram of a delta-sigma converter, including the delta-sigma modulator, low pass filter, and downsampling operation. Inspection of Fig. 3-7 shows the modulator output, y_{mod}, is given by

$$y_{mod} = \frac{x - y_{mod}}{f} + Q\,, \tag{3.39}$$

where f is frequency, Q is the quantization noise, x is the converter input. Solving for y_{mod} gives

$$y_{mod} = \frac{x}{f+1} + Q\left(\frac{f}{f+1}\right), \tag{3.40}$$

which demonstrates the modulator low pass filters the input signal and acts as a high pass filter to the quantization noise. Figure 3-8 illustrates the modulator operation, removing quantization noise from low frequencies and passing the signal of interest in the low pass filter passband. The converter subsequently low pass filters the modulator output, removing quantization noise, and downsamples to the final output data rate of f_{sample}.

An estimate of the ADC voltage noise density is

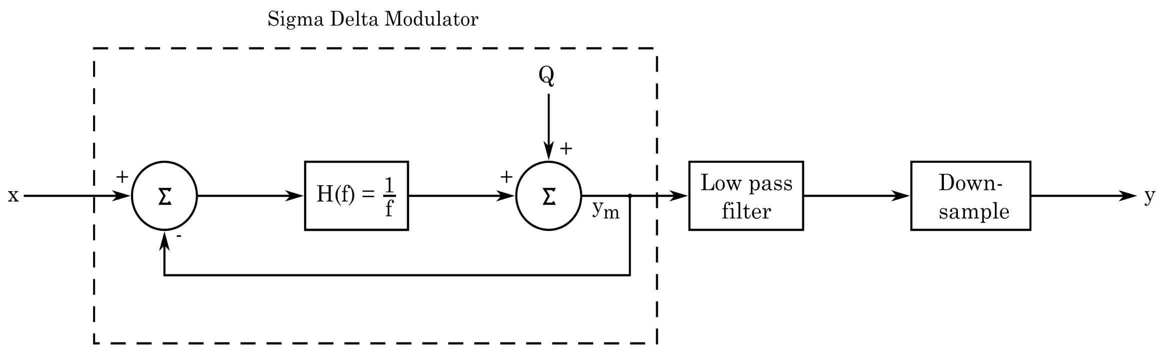


Figure 3-7. Block diagram of a first order sigma-delta converter.

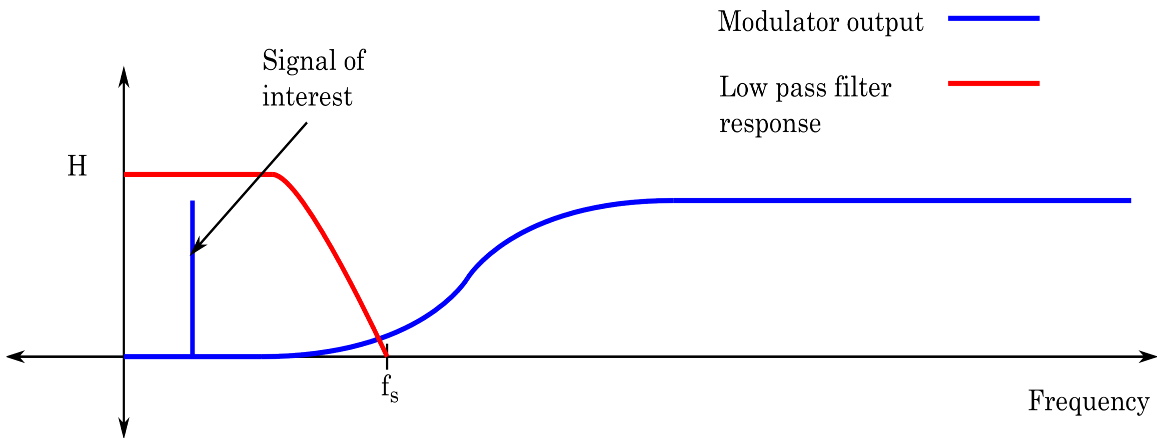


Figure 3-8. Output spectra of a first order sigma-delta modulator and passband of low pass filter.

$$E_{ADC} = \sqrt{\frac{V_{ADC}^{\;2}}{\Delta f_{ADC}}}, \tag{3.41}$$

where V_{ADC} is the ADC RMS noise voltage and Δf_{ADC} is the converter bandwidth. Table IV shows the SNR for a sigma delta converter for several oversampling ratios assuming a bandwidth of 0.018 Hz and calculating the ADC noise voltage with

$$V_{Noise} = E_{ADC}(0.018\text{ Hz})^{0.5}. \tag{3.42}$$

Table IV. ADC SNR for the ADS1278, a 24 bit sigma delta oversampling ADC, using $V_{Noise} = E_{ADC}(0.018\ \text{Hz})^{0.5}$ for several oversampling ratios.

Oversampling Ratio	SNR, dB
1	69.1
4	70.9
16	73.4
64	75.4

Two reference voltage sources provide positive and negative references for the ADC. Assuming the noise voltages from these sources are uncorrelated their sum is given by

$$\Delta V_{Ref,\,Total} = \sqrt{\Delta E_{Ref}^2 \Delta f + \Delta E_{Ref}^2 \Delta f} = \Delta E_{Ref} \sqrt{2 \Delta f}\,, \tag{3.43}$$

where ΔE_{ref} is the voltage reference noise voltage density in $V/\sqrt{Hz}$.

3.4.3 TIA Noise Model

Figure 3-9 shows a schematic of one photodiode channel with op amp noise sources, thermal noise from both resistors, and an equivalent circuit for the photodiode. A shunt resistance R_{Shunt} and a diode simulate the photodiode - the photodiode series resistance is omitted due to its relatively small magnitude. The op amp input noise voltage density, E_{NI}, and op amp input current noise density, I_{BI}, are included in the noise model. To find an mathematical expression for the noise voltage at the op amp output we incoherently sum the contribution of all noise sources at the TIA output to find[40]

$$E_{TIA} = \sqrt{(E_{NI} G_N)^2 \Delta f + (I_{BI} R_F)^2 \Delta f + 4\text{kTR}_F G_N \Delta f}\,. \tag{3.44}$$

Determination of the noise bandwidth, Δf, for a particular measurement method enables prediction of system noise performance with Eq. 3.44.

3.5 Average and Subtract Measurement Method

3.5.1 Description of Measurement Method

The average and subtract method of photocurrent measurement, Fig. 3-10, with the LED on and then averaging n measurements with the LED off. The difference between the average value with the LED on and with the LED off measures the desired signal voltage

$$V_{Signal} = \overline{ADC_{ON}} - \overline{ADC_{OFF}} = I_{PC} R_F \,, \tag{3.45}$$

where $\overline{ADC_{ON}}$ and $\overline{ADC_{OFF}}$ are the mean TIA voltages with the LED on and off. A low pass filter reduces the bandwidth of V_{Signal} to prevent aliasing during downsampling and increase the SNR. Spectral analysis of each component of the average and subtract method provides an estimate of its measurement bandwidth.

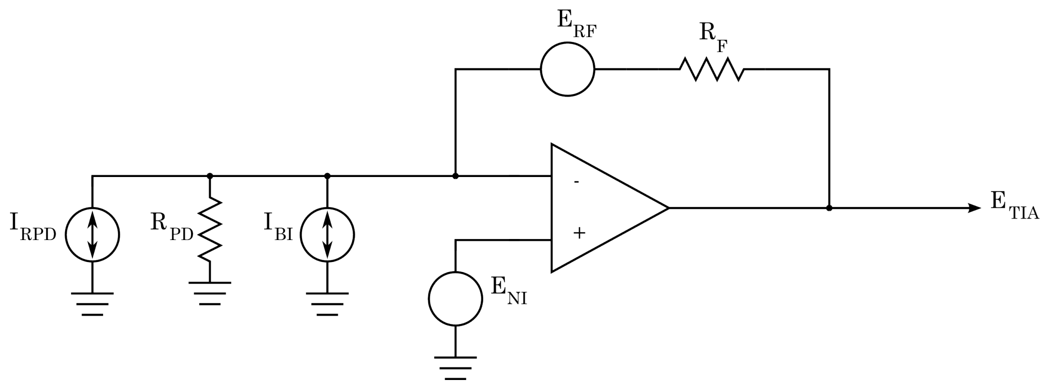


Figure 3-9. TIA noise model schematic including op amp noise sources.

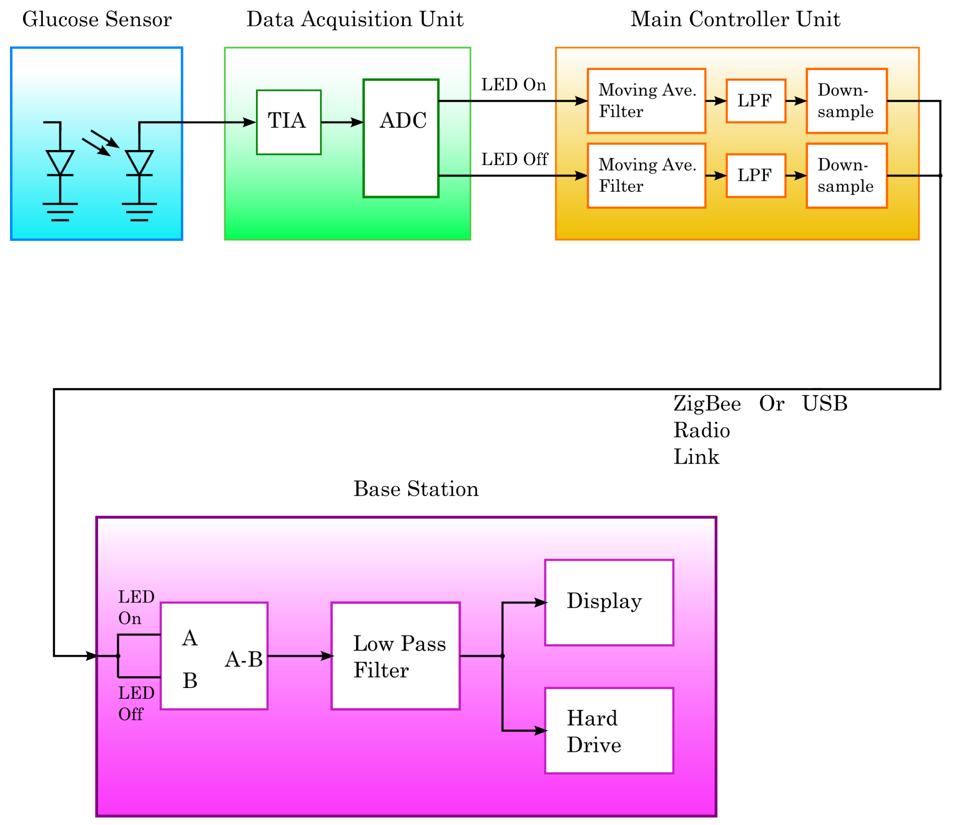


Figure 3-10. Block diagram of the average and subtract measurement method.

3.5.2 Spectral Analysis

3.5.2.1 TIA Bandwidth

Studies with the average and subtract method utilized a TIA with feedback resistor of 10 MΩ and feedback capacitance of 1000 pF. The feedback capacitor, C_F, limits the TIA bandwidth because as the frequency increases the capacitor impedance, 1/sC, reduces which shorts out the feedback resistor and limits TIA gain. Studies with the average and subtract method utilized an ADC sample rate of 2 kHz with each ADC sampling four photodiode channels in succession for a per-channel sample rate of 500 Hz. The component values indicated above result in a TIA 3dB bandwidth of

$$f_{TIA}=\frac{1}{2\pi R_F C_F}=16.7\,Hz. \tag{3.46}$$

3.5.2.2 ADC Bandwidth

The transfer function of the ADC on the DAU, the ADS1258, is given by[41]

$$|H(f)|=\left|\frac{\sin\left(\frac{128\pi f}{f_{Clk}}\right)}{64\sin\left(\frac{2\pi f}{f_{Clk}}\right)}\right|^5\left|\frac{\sin\left(\frac{128\pi\, Num_{Ave} f}{f_{Clk}}\right)}{Num_{Ave}\sin\left(\frac{2\pi f}{f_{Clk}}\right)}\right|, \tag{3.47}$$

where f_{Clk} = 16 MHz and Num_{Ave} = 16 for this method. A plot of the ADC transfer function, Fig. 3-11, shows the ADC 3 dB bandwidth is BW_{ADC} = 4687 Hz for the DAU ADC configuration.

3.5.2.3 Averaging Filter and Subtraction

Figure 3-12 shows a plot of the moving average filter transfer function for averaging 215 LED on samples and a derivation of the moving average filter transfer function is included in Appendix A. Numerically integrating the area under the magnitude of the filter transfer function shows the equivalent noise bandwidth for averaging 215 samples is

$$ENB_{N=215}=7.38\,Hz. \tag{3.48}$$

The standard deviation of the mean of n samples from a normal distribution with standard deviation σ and mean μ is

$$\sigma_n=\frac{\sigma}{\sqrt{n}}\,. \tag{3.49}$$

The standard deviation of the mean of n LED on and LED off values are

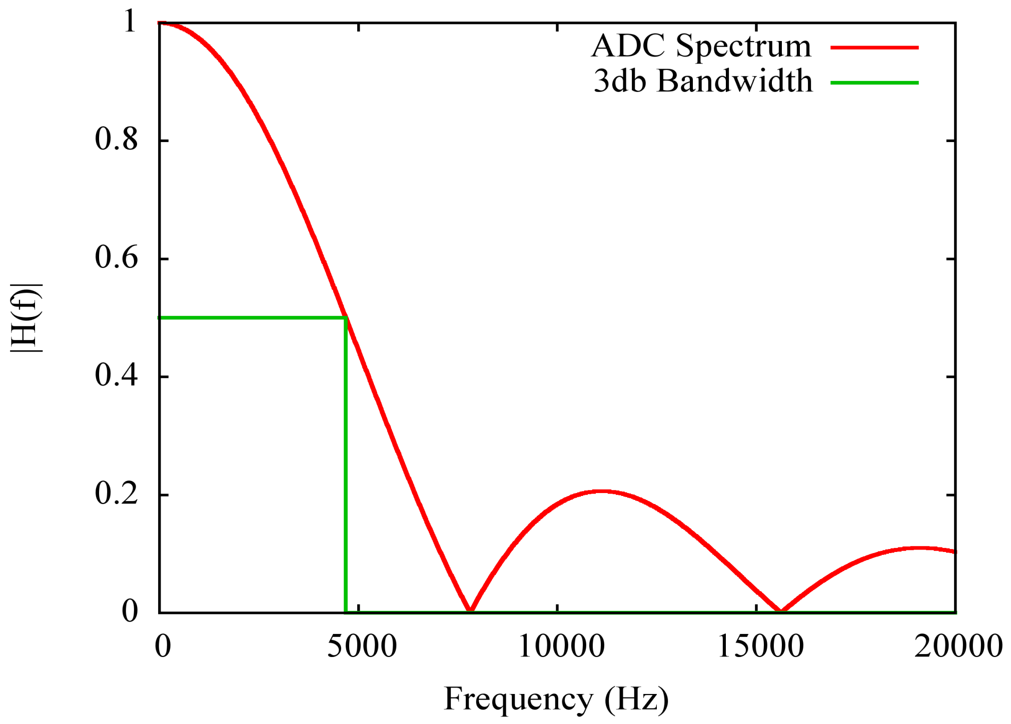


Figure 3-11. Texas Instruments ADS1258 transfer function for Num_{Ave}=16.

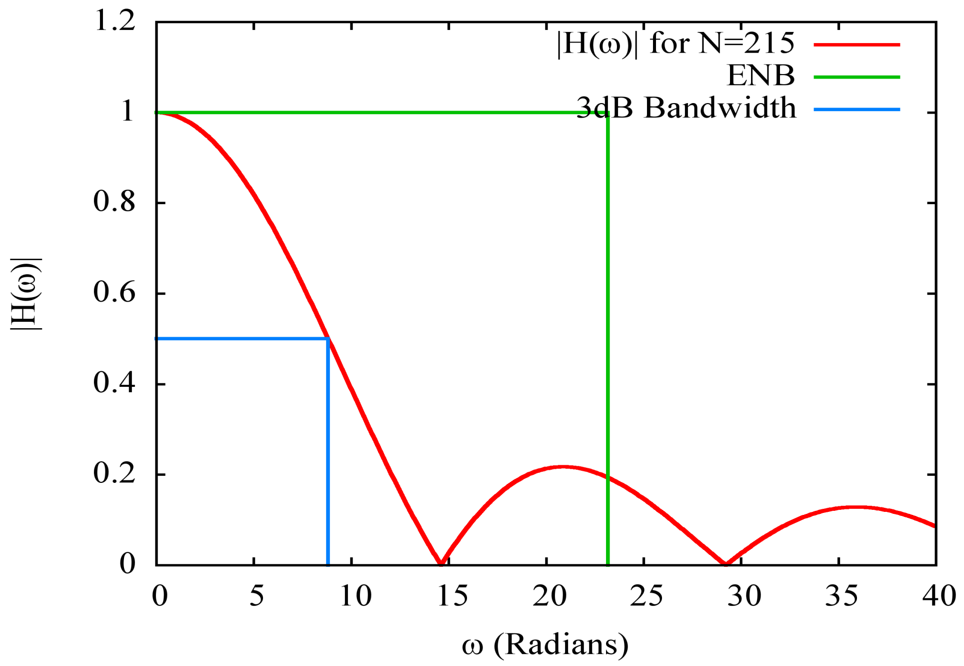


Figure 3-12. Spectrum of averaging process for N=215 showing 3dB bandwidth and equivalent noise bandwidth.

$$\sigma_{n,On}=\frac{\sigma_{On}}{\sqrt{n}}, \tag{3.50}$$

and

$$\sigma_{n,Off}=\frac{\sigma_{Off}}{\sqrt{n}}, \tag{3.51}$$

where $\sigma_{On(Off)}$ is the standard deviation of the LED on(off) samples.

Since we subtract the mean LED on and off voltages to find the signal voltage the standard deviation of the difference is the sum of the LED on and off standard deviations,

$$\sigma_{On-Off}=\sigma_{n,On}+\sigma_{n,Off}=\frac{\sigma_{On}+\sigma_{Off}}{\sqrt{n}}. \tag{3.52}$$

With the approximation that σ_{On} is approximately equal to σ_{Off},

$$\sigma_{On-Off}=2\left(\frac{\sigma_{On}}{\sqrt{n}}\right). \tag{3.53}$$

Since 10log(2) = 3.01 dB, the final SNR is 3.01 dB less than the LED on SNR.

3.5.2.4 Low Pass Filter

A first order Butterworth low pass filter with cutoff frequency 0.3 Hz after the averaging process prevents aliasing during downsampling. Software in the base station implements a low pass filter for a final bandwidth of 0.018 Hz.

3.5.3 Noise Model

The portion of the average and subtract method with the lowest bandwidth determines the measurement bandwidth. Thus the measurement bandwidth of the average and subtract method is 0.018 Hz, the low pass filter bandwidth, and one must subtract 3.01 dB from the noise model due to the subtraction operation. Estimating the op amp input voltage noise at DC by extrapolating the data sheet noise voltage plot to approximately 1 Hz results in 45 nV/ $\sqrt{(Hz)}$. A plot of the noise model for the Average and Subtract method is shown in Fig. 3-13 where the actual measurement bandwidth is the twice the ENB to account for the two-sided passband for all sources except shot noise.

Figure 3-13 shows the SNR for a 30 kΩ shunt resistance photodiode, 42.5 dB, meets the 40 dB SNR goal and the op amp input voltage noise dominates system noise at that resistance. The ADC noise, voltage reference noise, and op amp input current noise have little effect on the system SNR as their noise contribution is 10 to 30 dB less than the noise model near 10 kΩ to 30 kΩ.

3.6 LIA with Discretized Sinusoid Reference

3.6.1 Description of Measurement Method

Figure 3-14 shows the second proposed photocurrent measurement system which utilizes a single phase LIA with sinusoid reference signal to measure the magnitude of the photocurrent from each photodiode. Figure 3-15 illustrates the input (ADC samples), reference, mixer output, and LIA output as functions of time. The reference sinusoid has magnitude A_{Ref} and frequency

$$f_{Ref}=\frac{1}{t_{Cycle}} . \tag{3.54}$$

The input signal has the value of V_{On} for t=0 to t=t_{On} and the value V_{Off} for t from t_{On} to t_{Cycle}, with

$$t_{On}=t_{Off}=\frac{t_{Cycle}}{2} . \tag{3.55}$$

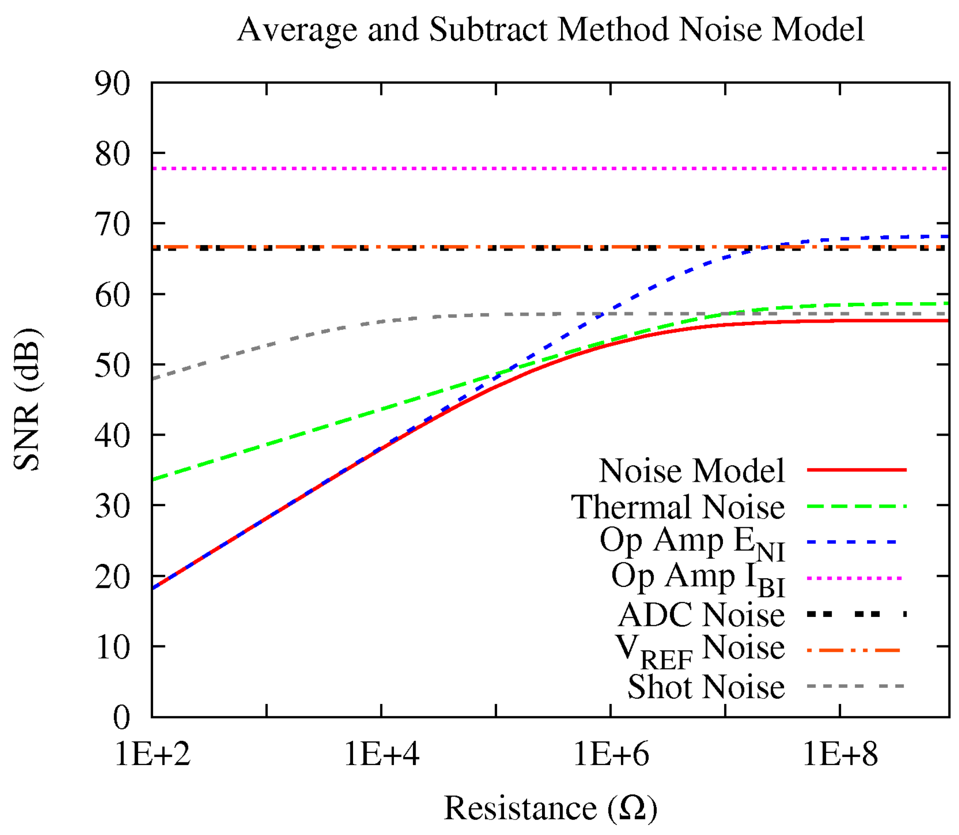


Figure 3-13. Noise model for the average and subtract method.

The mixer multiplies the input and reference signals and the low pass filter in the LIA passes the DC component of the mixer output. The low pass filter averages the mixer output over an entire lock-in cycle:

$$LIA_{Out}=\frac{1}{2}\left(\overline{Mixer_{On}}+\overline{Mixer_{Off}}\right), \tag{3.56}$$

where $\overline{Mixer_{On(Off)}}$ is the mean of the mixer output while the LED is on(off). Including expressions for the mean mixer outputs we have

$$LIA_{Out}=\frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n}Ref[i]\,ADC[i]+\frac{1}{n}\sum_{i=n+1}^{2n}Ref[i]\,ADC[i]\right), \tag{3.57}$$

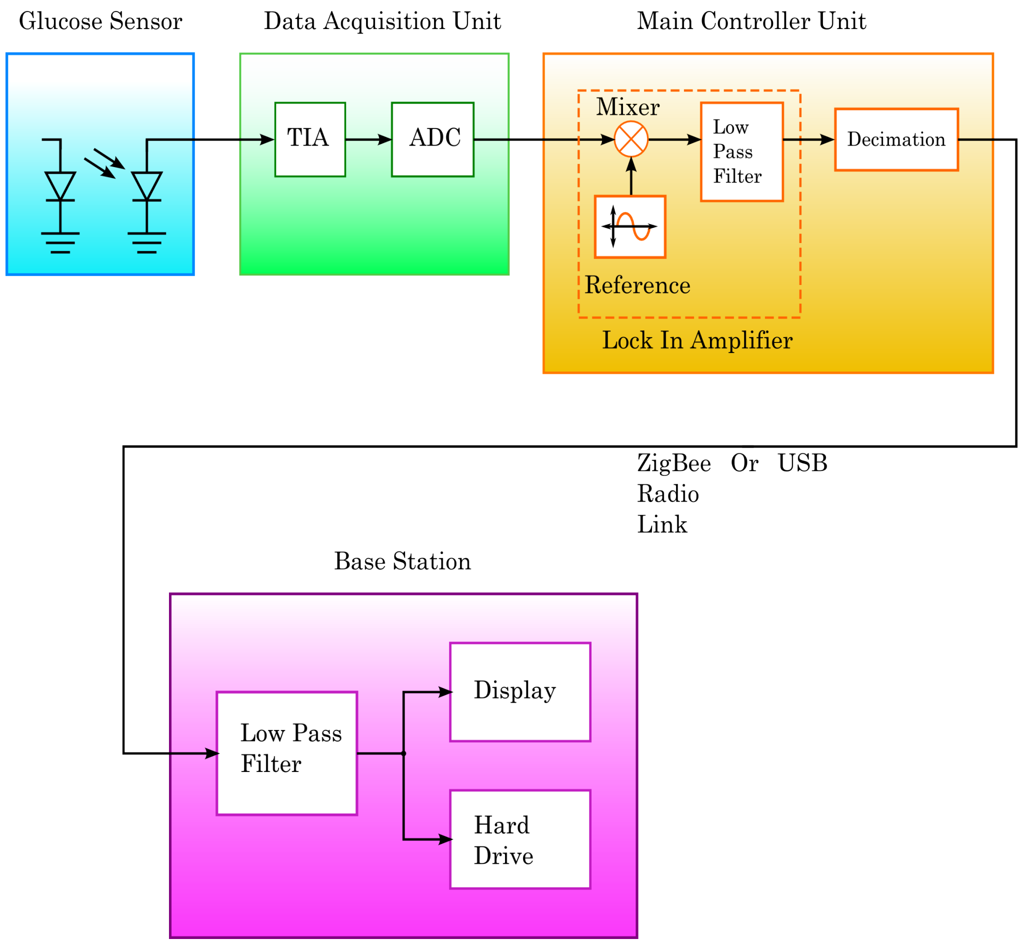


Figure 3-14. Block diagram of lock-in amplifier measurement system.

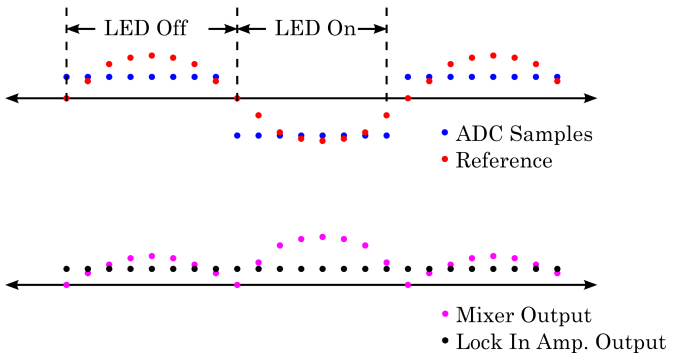


Figure 3-15. Operation of LIA with discretized sinusoid reference signal.

where Ref[i] is the digital representation of the reference signal and ADC[i] is the input signal comprised of ADC samples. Consolidating Eq. 3.57 into one sum the LIA output becomes

$$LIA_{Out} = \frac{1}{2\text{n}} \sum_{i=1}^{2\text{n}} Ref[i] ADC[i]. \tag{3.58}$$

3.6.1.1 Discretized Sinusoid Lock-in Implementation

The MCU utilizes 32 bit fixed-point integer math to implement the LIA. The reference sinusoid is comprised of Q0.31 format numbers while 24 bit two's complement ADC samples sign extended into Q0.31 format form the input signal.

Qm.n format numbers are binary numbers with 1 sign bit, m digits to the left of the decimal point, and n digits to the right of the decimal point. Q0.n numbers, also denoted Qn, are scaled so that the maximum and minimum numbers in this format are $1\text{-}2^{-n}$ and -1. The advantage of this convention is that overflow cannot occur as a result of a multiplication operation. To convert a floating point number between -1 and $1\text{-}2^{n}$ to Qm.n format, multiply by 2^{n} And to convert a Qm.n number back to floating point, divide by 2^{n}. Multiplying two Qm.n numbers requires adjusting the decimal point after multiplication. This is apparent if we convert two floating point numbers, $M_{1,\text{Float}}$ and $M_{2,\text{Float}}$ to Qm.n and multiply them:

$$M_{1,Float} 2^{n} M_{1,Float} 2^{n} = M_{1,Float} M_{2,Float} 2^{2\text{n}} . \tag{3.59}$$

This result has 2n bits to the right of the decimal point and must be divided by 2^{n} to obtain the result in Qm.n format. The multiplication operation for Qm.n is given by:

$$\frac{M_{1,Float}2^{n}M_{1,Float}2^{n}}{2^{n}}=M_{1,Float}M_{2,Float}2^{n}\ . \tag{3.60}$$

A processor easily performs the division by 2^n using a right shift of n bits. The mixer multiplies Q0.31 format ADC and Reference signals and the desired lock in output format is Q0.31, so a right shift of 62 – 31 = 31 provides Q0.31 format output.

Multiplication of two n bit binary numbers results in a 2n bit binary number. Thus multiplication of two Q0.31 numbers, which have 32 bits, requires a 64 bit wide accumulator. The mean LIA output considering the shift required for the multiplication and multiplication by 2^{31} to convert to floating point is

$$\overline{LIA_{Out}}=\frac{1}{2\mathrm{n}}(2^{31})\sum_{i=1}^{2\mathrm{n}}\left[\left(Ref_{Q0.31}[i]\,ADC_{Q0.31}[i]\right)\gg 31\right], \tag{3.61}$$

where the subscript Q0.31 indicates the quantity is in Q0.31 format.

The LIA must measure the signal voltage due to photocurrent, $I_{PC}R_f$. We must calculate the expected DC value of the LIA with our sinusoid reference signal and square wave input signal to properly scale the LIA output. First consider the first half of one cycle when the LED is on. With reference signal

$$Ref(t)=A_{Ref}\sin(\omega t), \tag{3.62}$$

where

$$\omega=2\frac{\pi}{t_{Cycle}}, \tag{3.63}$$

and A_{Ref} is the reference signal magnitude, the mean mixer output voltage is

$$\overline{Mix_{On}}=\frac{1}{t_{On}}\int_{0}^{t_{On}} V_{On} A_{Ref}\sin(\omega t)\,. \tag{3.64}$$

Performing the integration we find

$$\overline{Mix_{On}}=\frac{-A_{Ref} V_{On}}{\omega t_{On}}\left(\cos(\omega t_{On})-\cos(0)\right). \tag{3.65}$$

Since $\omega = 2\pi/t_{Cycle}$, $t_{On} = t_{Cycle}/2$, and $\omega t_{On} = \pi$ and we find

$$\overline{Mix_{On}}=\frac{-A_{Ref} V_{On}}{\pi}(\cos(\pi)-1)=\frac{2A_{Ref} V_{On}}{\pi}. \tag{3.66}$$

A similar process for the time when the LED is off shows that

$$\overline{Mix_{off}}=\frac{-2A_{Ref} V_{Off}}{\pi}. \tag{3.67}$$

The LIA output is the average of Eq.s (3.66) and (3.67):

$$LIA_{Output}=\frac{1}{2}\left(\overline{Mix_{On}}+\overline{Mix_{Off}}\right)=\frac{A_{Ref}}{\pi}(V_{On}-V_{Off}). \tag{3.68}$$

The signal voltage is then

$$V_{On}-V_{Off}=\frac{\pi}{A_{Ref}}LIA_{Out}. \tag{3.69}$$

The final equation for the LIA output considering Equations (3.61) is

$$V_{On}-V_{Off}=\frac{1}{2\text{n}}(2^{31})\frac{\pi}{A_{Ref}}\sum_{i=1}^{2\text{n}}\left[\left(Ref_{Q0.31}[i]ADC_{Q0.31}[i]\right)\gg 31\right]. \tag{3.70}$$

3.6.2 Spectral Analysis

We must determine the noise bandwidth of the discretized sinusoid LIA to plot the noise model. The TIA and ADC portions of the discretized sinusoid LIA system are the same as used in the average and subtract method so their bandwidth and noise contributions apply here as well. The noise bandwidth a

LIA is the combined bandwidth of all low pass filters after the mixer.

A plot of the moving average filter spectrum for the average of N=430 samples, Fig. 3-16, shows that the 3dB bandwidth for the average of one cycle of mixer output voltages is 0.7 Hz. Application of a low pass filter with cutoff frequency 0.3 Hz before decimation prevents aliasing of data sent to the base station. The base station low pass filter provides the final measurement bandwidth of 0.018 Hz for the discretized sinusoid LIA.

3.6.3 Noise Model

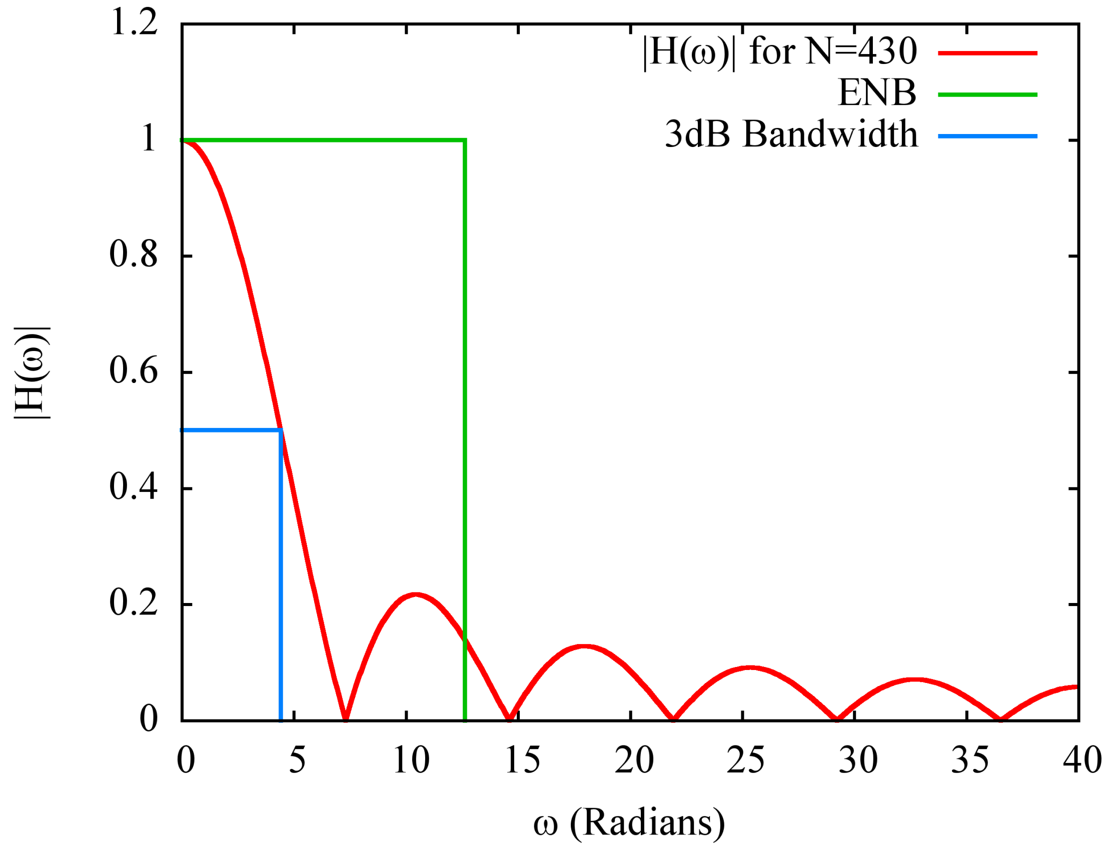


Figure 3-16. Output spectrum of the moving average filter for N=430.

A total of 430 samples in one reference signal cycle sets the reference frequency at

$$F_{Ref} = \Delta t_{Sample} N_{Samples} = \left(\frac{1}{500\,Hz}\right) 430 = 0.86\,Hz. \tag{3.71}$$

Extrapolating the input voltage noise plot in the op amp datasheet to 0.86 Hz shows input voltage noise is approximately 35 nV/ $\sqrt{(Hz)}$. The discretized sinusoid LIA contains no subtraction operation and does not require subtracting 3.01 dB from the noise model result. The discretized sinusoid LIA noise model is plotted in Fig. 3-17.

The noise model for the discretized sine wave LIA appears very similar to that of the average and subtract method, with SNR 41.8 dB at 30 kΩ and SNR determined by op amp input voltage noise near 30 kΩ. The ADC noise, voltage reference noise, and op amp input current noise have little effect on the system SNR for the discretized sinusoid LIA as well as for the average and subtract method.

3.7 Combined Lock-in Filter - ADS1258 Configuration

3.7.1 Description of Measurement Method

The combined lock-in filter method applies the lock-in amplifier described by Dorrington and Kunnemeyer[31]. This LIA utilizes a reference frequency equal to the Nyquist sampling rate and replaces the mixing operation with the inversion of every other sample. Figure 3-18 illustrates a block diagram of this method. Inverting every other sample is equivalent to multiplying the input signal by a cosine at the reference frequency. Since f(t)cos(ω_R) is a Fourier transform pair with ½[F(ω+ω_R) + F(ω-ω_R)] multiplying the LIA input by a cosine at the reference frequency with $f_{Reference} = f_{Nyquist}$ shifts the spectrum of the LIA input by the lock in reference frequency, which shifts at the reference frequency to DC, see Fig. 3-19. Then a low pass filter removes the remaining undesired frequency components. The procedure described by Dorrington and Kunnemeyer, which will be called the combined LIA filter method, implements both a single pole Butterworth low pass filter and the inversion process by alternately using the difference equations

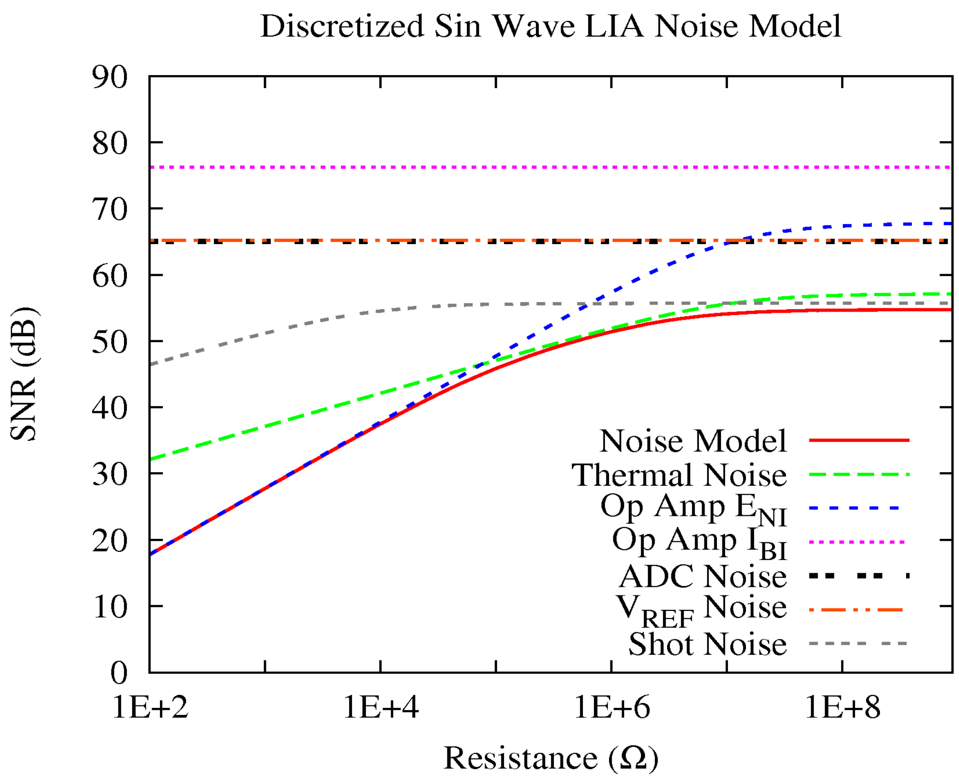


Figure 3-17. Discretized sinusoid LIA noise model.

$$y_n = \frac{1}{2^m}(x_n - x_{n-1}) + \left(1 - \frac{1}{2^{m-1}}\right) y_{n-1}, \tag{3.72}$$

and

$$y_n = \frac{1}{2^m}(-x_n + x_{n-1}) + \left(1 - \frac{1}{2^{m-1}}\right) y_{n-1}. \tag{3.73}$$

The ADS1258 has no latency assuming the ADC inputs are stable. When the ADC inputs change during an ADC sample cycle, the resulting output sample does not accurately represent the input signal. After the ADC inputs settle, the ADC must process an entire sample cycle with settled inputs

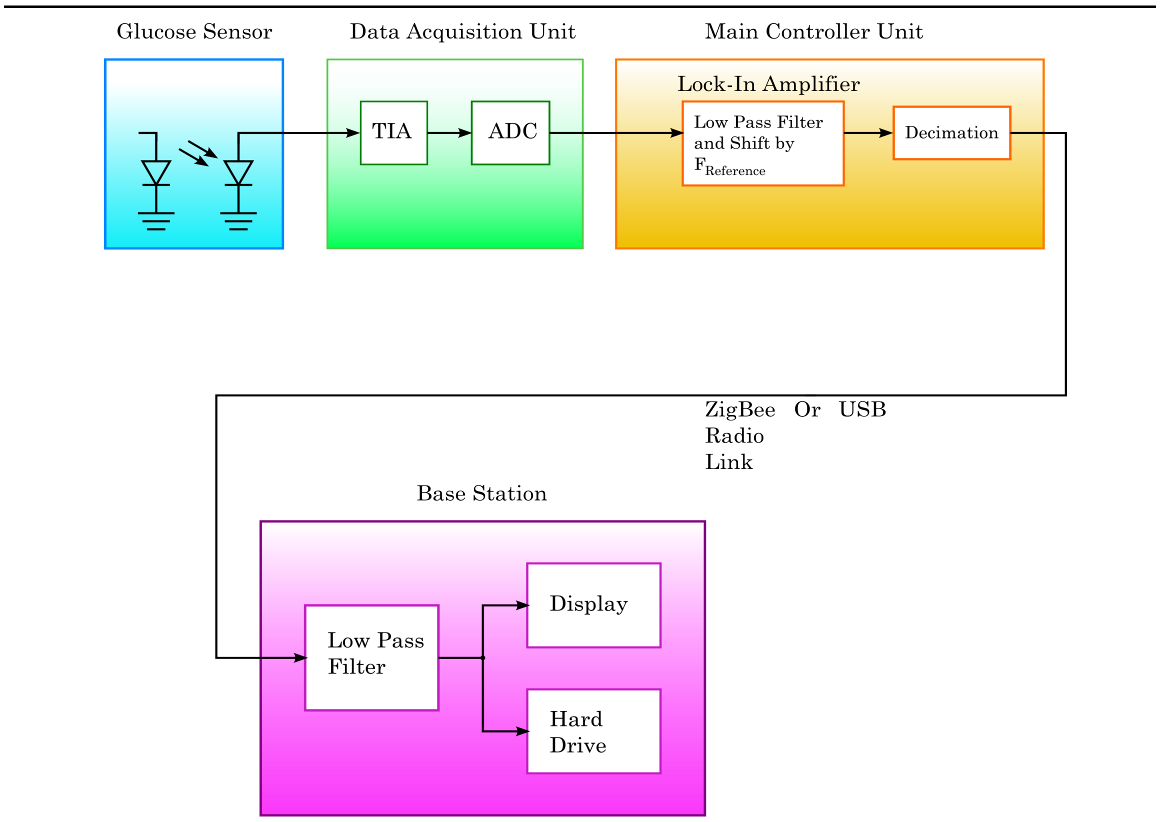


Figure 3-18. A block diagram of the combined lock-in amplifier measurement method.

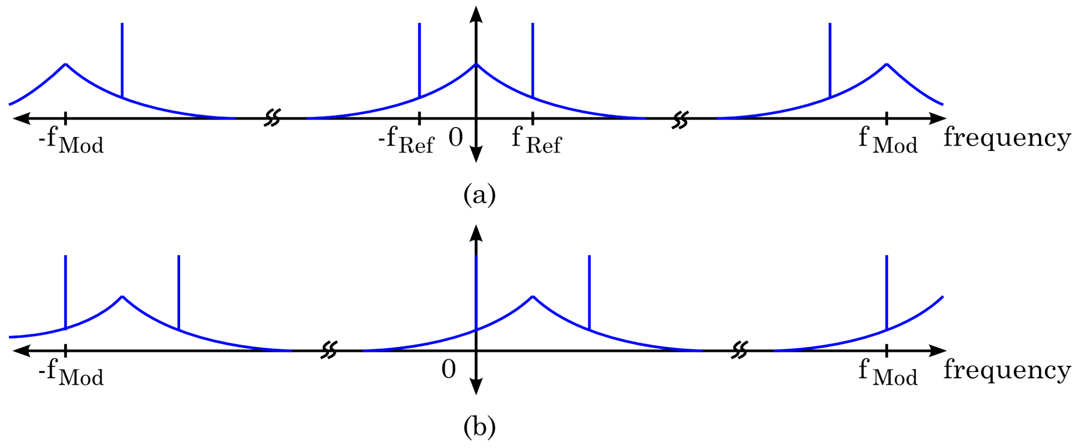


Figure 3-19. Combined LIA. (a) Input spectrum. (b) Output spectrum after shifting spectrum by $f_{Reference}$.

to accurately represent the input voltage.

3.7.2 Advantages of Combined LIA Filter

The combined LIA filtering method allows a higher reference frequency than is possible using a discretized sine wave for the reference signal assuming equal sampling frequencies and utilization of all samples. If an LIA uses a sine wave sampled 2^m times during one cycle for a reference signal, the two part LIA filter in Equations 3.72 and 3.73 requires only two samples resulting in a reference frequency 2^{m-1} times higher than the reference frequency of the discretized sine wave. This may help increase SNR in cases where a noise source with 1/f noise limits the SNR as is often the case for the input voltage noise of op amps. If the LIA reference frequency is just high enough that the op amp input voltage noise is dominated by the broadband noise, as in Fig. 3-5, the input voltage noise contribution to system SNR is minimized. Increasing the LIA reference frequency above the lowest frequency where the broadband noise dominates the 1/f noise has very small additional benefit since further increase in the reference frequency results in only a slight noise reduction.

3.7.3 Spectral Analysis

I chose a DAU ADC sample rate of 1.22 kHz for a lock-in reference frequency of 610 Hz. The DAU firmware decimates the sample data to produce a final DAU data rate of 1 Hz, requiring a low pass filter cutoff frequency below 0.5 Hz to prevent aliasing. The combined LIA filter algorithm in the DAU firmware is given by

$$y_n = \frac{1}{2^{10}}\left(x_n - x_{n-1}\right) + \left(1 - \frac{1}{2^9}\right) y_{n-1}, \tag{3.74}$$

and

$$y_n = \frac{1}{2^{10}}\left(-x_n + x_{n-1}\right) + \left(1 - \frac{1}{2^9}\right) y_{n-1}. \tag{3.75}$$

Application of the single pole Butterworth low pass filter

$$y_n = \frac{1}{2^{10}}\left(x_n + x_{n-1}\right) + \left(1 - \frac{1}{2^9}\right) y_{n-1} \tag{3.76}$$

reduces the bandwidth below 0.5 Hz to prevent aliasing during downsampling. The combined LIA and low pass filter could be combined but splitting them reduces possibility of rounding errors due to small filter coefficients as these calculations are performed using fixed-point arithmetic. The base station software contains a first order Butterworth low pass filter with variable cutoff frequency to further reduce measurement bandwidth and increase SNR. The variable cutoff frequency allow flexibility during experiments and can alternately be applied before data transmission to the base station. If the system bandwidth is too low the system will not respond to changes in glucose concentration. Glucose concentration in people changes over an approximate time period of a few minutes to ten or fifteen minutes so I chose an initial system bandwidth of 0.018 Hz to prevent the possibility of removing glucose concentration changes from the photocurrent spectra while maximizing SNR. Further studies included in Chapter VI explore determination of final system bandwidth based on measurements with the glucose sensor. The Butterworth low pass filter

$$y_n=\frac{1}{32}\left(x_n+x_{n-1}\right)+\frac{15}{16}y_{n-1} \tag{3.77}$$

in the base station sets the measurement bandwidth to 0.018 Hz as shown in a plot of the system noise spectrum in Fig. 3-20.

3.7.4 Noise Model

The low pass filters in the combined LIA filtering method pass signals within the filter bandwidth of the reference frequency so that the contribution of the op amp input voltage noise is the value of the op amp input voltage noise density at the reference frequency. The MAX4478 input voltage noise density at 610 Hz is 5.5 $\mu V/\sqrt{Hz}$. Figure 3-21 shows a plot of the noise model for the combined LIA filtering method.

Figure 3-21 shows the combined LIA method SNR, 44.4 dB, meets the goal of 40 dB SNR for 30 kΩ photodiode arrays since the reference frequency is high enough that broadband noise dominates op amp input voltage noise. Thermal noise restricts system SNR in the 10 kΩ to 1 MΩ range. ADC, voltage reference,

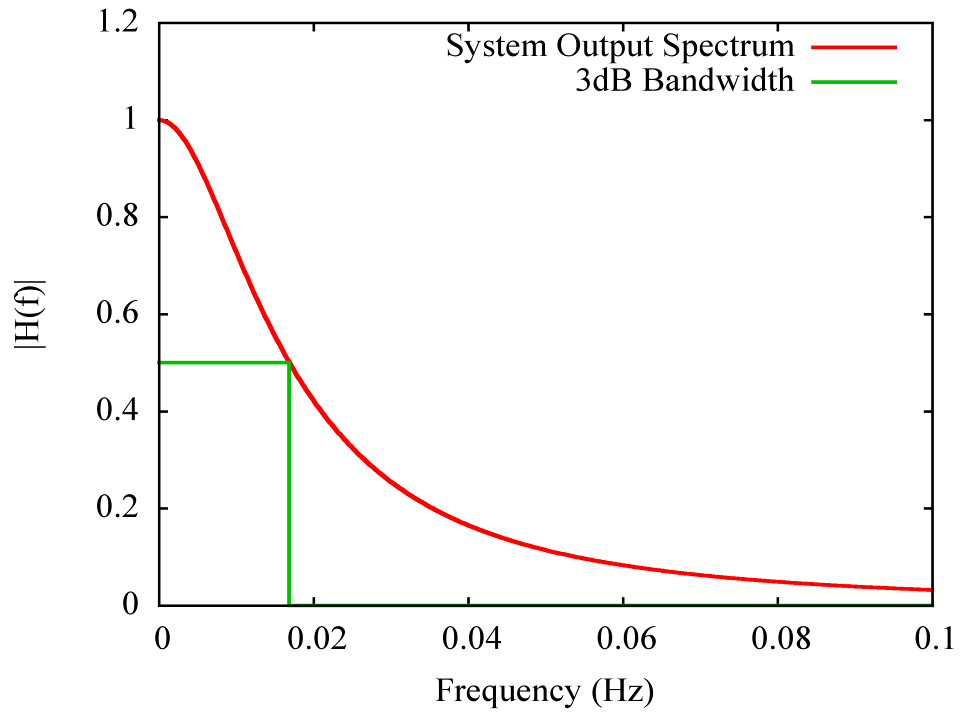


Figure 3-20. Spectrum of combined LIA measurement method, including low pass filters in MCU and base station, for the ADS1258 configuration.

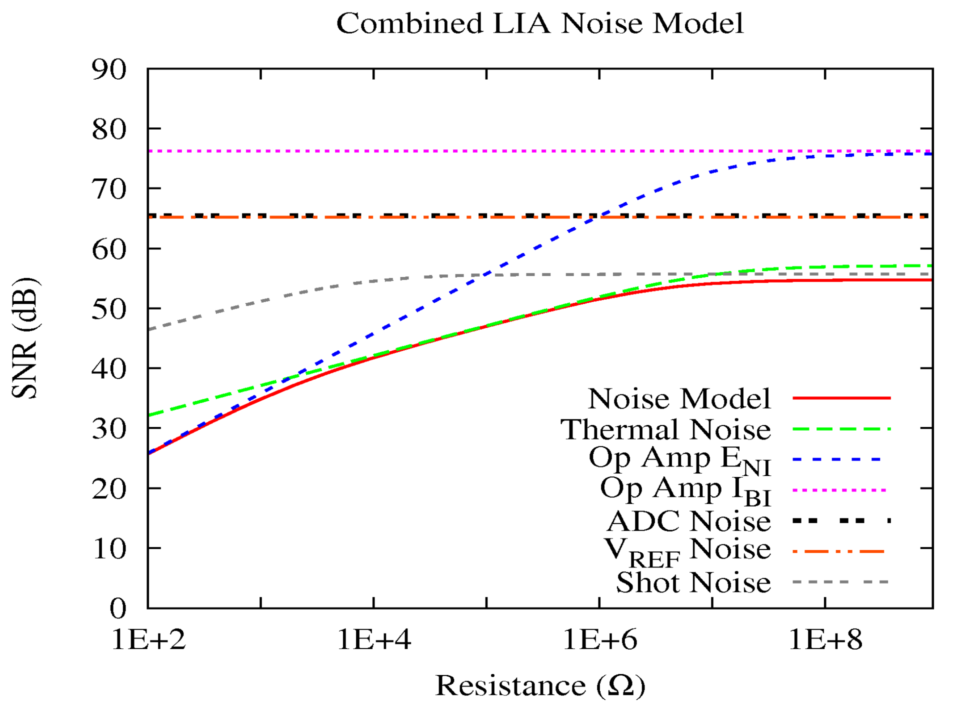


Figure 3-21. Noise model of combined LIA method.

and op amp input current noise have no effect on system SNR for this measurement method as their SNRs are 10 to 30 dB higher than the noise model.

3.8 Combined Lock-in Filter - ADS1278 Configuration

3.8.1 Description of Measurement Method

An alternate DAU configuration utilizes the ADS1278 ADC and the combined lock-in filtering method illustrated in Fig. 3-18. The ADS1278 input voltage range spans from ground to a positive supply voltage and therefore on the DAU the amp non-inverting inputs are connected to a reference voltage between the positive supply and ground. This enables the ADC to measure voltages falling below the reference voltage. An alternate biasing method utilizes 2.5V and 5V supplies with the op amp non-inverting inputs connected to 2.5V.

The ADS1278 exhibits a latency of 40 sample periods with a few percent of ripple voltage due to a step change in the analog inputs. The MCU software minimizes the effect of this effect by recording several samples for each LED configuration as discussed below.

3.8.2 Spectral Analysis

The transimpedance amplifier feedback capacitor of 100 pF limits the TIA output bandwidth to 159 Hz, well below half the ADC modulator frequency of 50 kHz preventing aliasing in the sampling process. The MCU digital filters reduce the lock-in amplifier bandwidth to below 0.5 Hz so aliasing does not occur during downsampling to 1 Hz.

The ADS1278 DAU configuration software uses the combined LIA filter

$$y_n = \frac{1}{2^8}\left(x_n - x_{n-1}\right) + \left(1 - \frac{1}{2^7}\right) y_{n-1}, \tag{3.78}$$

and

$$y_n = \frac{1}{2^8}\left(-x_n + x_{n-1}\right) + \left(1 - \frac{1}{2^7}\right) y_{n-1}, \tag{3.79}$$

in the MCU and the low pass filter

$$y_n = \frac{1}{32}\left(x_n + x_{n-1}\right) + \frac{15}{16} y_{n-1} \tag{3.80}$$

in the PC software to achieve a final system bandwidth of 0.017 Hz.

I configured the ADS1278 DAU to record ADC samples at 781 Hz. Sampling the analog inputs for four sets of ten samples with LED1 on, LEDs off, LED2 on, and LEDs off and filtering only the fifth sample of each set reduces the effect of ripple voltage due to the ADC latency. Also since the ADC delays the samples by 40 sample periods recording 40 samples for each lock-in cycle simplifies keeping track of which array to store the samples in. The resulting lock-in reference frequency is 39 Hz. Experiments with 1, 2, 4, 8, and 16 samples with stable ADC inputs showed the ripple voltage due to the ADC latency increased enough below 8 samples that the ADC samples did not represent the ADC input.

3.8.3 Noise Model

A few differences between the ADS1258 and ADS1278 configurations impact the nose model including the voltage reference circuit, reference frequency, and ADC noise. The ADS1278 configuration DAU requires only one voltage reference IC, the MAX6126, instead of two references required by the ADS1258 DAU. The MAX6126 broadband noise estimates the voltage reference noise contribution since the manufacturer does not include a plot of the noise spectrum in the datasheet and two RC filters on the DAU filter the reference voltage. The MAX6126 broadband noise specification is 45 nV/ $Hz^{-0.5}$ at 1 kHz. The MAX4478 op amp input voltage noise at the reference frequency of 38 Hz is 12 $nV/Hz^{-0.5}$. and the ADS1278 noise is 8 μV RMS. The ADS1278 configuration noise model for a bandwidth of 0.017 Hz is shown in Fig. 3-22, and the noise model software is included in Appendix F.

As the noise model plot shows, the relatively low reference frequency of 39 Hz adds more op amp input voltage noise to the model such that the thermal noise and op amp input voltage noise curves meet near 10 kΩ. For prototype PD arrays with shunt resistance 10 kΩ to 30 kΩ, the low reference frequency impacts the noise model by only a few tenths of a dB, and thus does not significantly degrade system performance. The noise model SNR of 44.8 dB at 30 kΩ shunt resistance meets the initial goal of 40 dB. The ADC noise, voltage reference noise, and op amp input current noise do not affect the noise model for this configuration.

3.9 Analysis and Summary

A comparison plot of all four photocurrent measurement methods presented in this chapter, Fig. 3-23, shows all filtering methods meet the SNR requirement, but the combined LIA filters have higher SNR by approximately 2

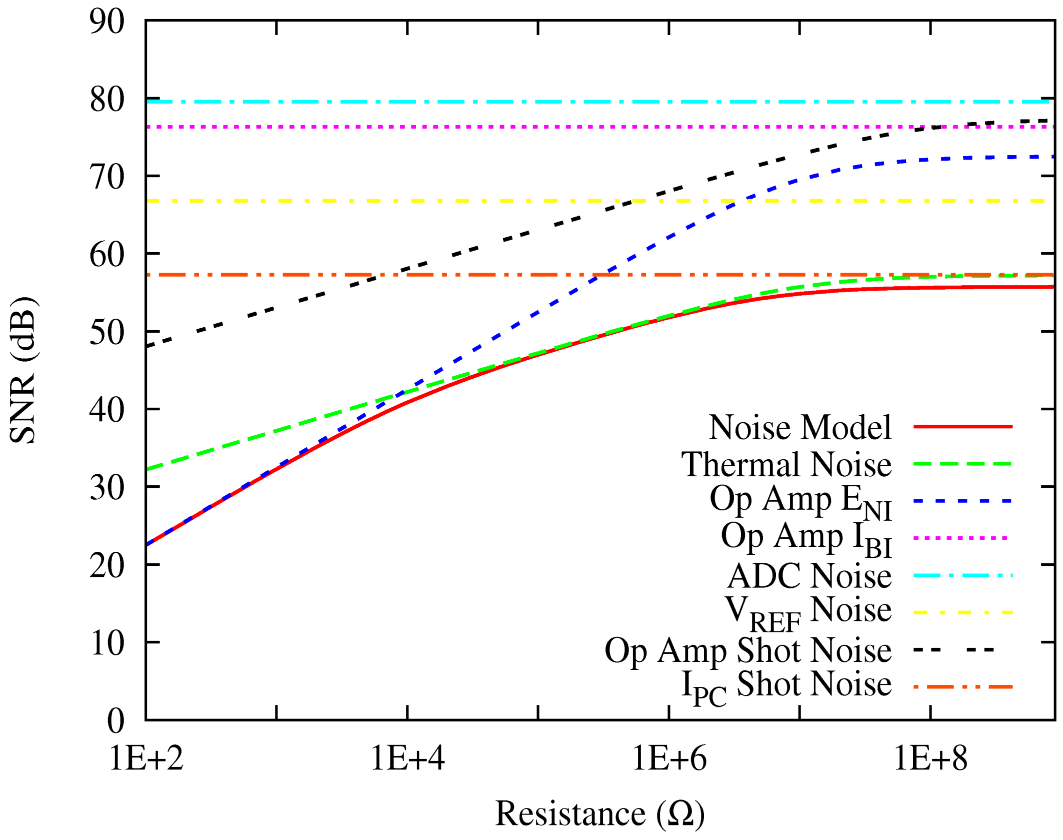


Figure 3-22. ADS1278 DAU noise model with system bandwidth of 0.017 Hz.

dB at 30 kΩ shunt resistance due to reduced contribution from the op amp input voltage noise enabled by a higher reference frequency. The ADS1258 model has a slightly higher SNR below 30 kΩ since the reference frequency is higher than that of the ADS1278 configuration. The average and subtract method exhibits a noise model nearly equal to that of the discretized LIA method. The remainder of experiments in this study employ the combined LIA filtering method. Further experiments in Chapter 5 select which DAU configuration provides acceptable performance for use in continuous glucose monitoring.

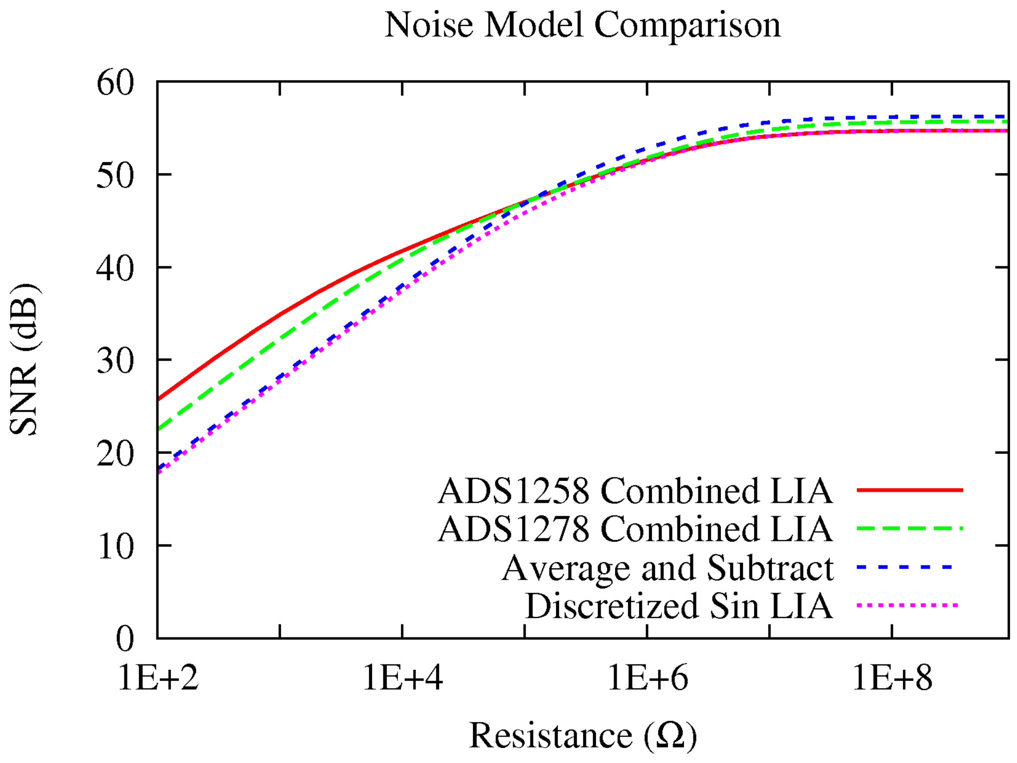


Figure 3-23. Comparison of all four measurement method noise models.

CHAPTER 4

DAU HARDWARE, FIRMWARE, AND SOFTWARE

This chapter documents design of DAU hardware and software enabling the DAU to meet system performance goals. DAU design must implement the combined LIA measurement method and meet several other design factors. DAU hardware, firmware, and software requirements are detailed before a discussion of the DAU design.

4.1 Requirements

4.1.1 Mechanical Requirements

- The DAU printed circuit card must interface with connectors on the end of the MCU card including a two pin 0.1 inch spacing connector and a 20 pin single row 0.1 inch spacing connector.
- The card must include a 40 pin DIP zero insertion force (ZIF) socket for glucose sensor installation.
- The sensor ZIF socket must be in the vertical orientation so that air bubbles will not collect in the glucose sensor fluid chambers.
- The MCU and DAU cards must connect together with the MCU card in the horizontal orientation and DAU in the vertical orientation with the glucose sensor installed on the side of the DAU card away from the MCU card. This configuration keeps tubing connected to the glucose sensor away from electrical components in case of a leak.

4.1.2 Electrical Requirements

- The DAU must interface with the connector on the edge of the MCU card.
- The DAU must provide LED current of 100 to 200 mA modulated with two enable lines from the MCU and include provision to alter the amount of LED current by changing a component value.
- The DAU must provide 32 transimpedance amplifiers to translate glucose sensor photocurrent into voltages for subsequent sampling by ADCs on the DAU.
- An SPI port must be used for transmitting photocurrent samples to the MCU.
- The DAU must generate any required voltage supplies from batteries.
- The measurement method must provide photocurrent measurement SNR greater than 40 dB where the signal to noise ratio is given by 10 log SNR for a photodiode with shunt resistance of 30 kΩ.
- Power draw measurements for the DAU and MCU shall be recorded to provide a starting point for future system miniaturization efforts.

4.1.3 Software Requirements

- The MCU firmware must be written in C using the Microchip IDE[42].
- The base station software must be written in Visual Basic 6.0.
- The system shall provide photocurrent measurements for all glucose sensor channels and for both LEDs at a rate of approximately 1 Hz.
- The software shall implement the combined LIA method.

- Software will modulate the LEDs and record samples with LED 1 on, no LEDs on, LED2 on, and no LEDs on.
- The system software shall provide provision for detection of transients due to air bubbles passing through the glucose sensor. An air bubble passing through the sensor causes a large transient in photocurrent measurements which invalidates the glucose concentration calculations.
- The base station software shall display and record all photocurrent measurements.

4.2 Hardware Design

Figure 4-1 illustrates a block diagram of the ADS1258 DAU, including the glucose sensor, TIAs, ADCs, as well as LED drivers, voltage regulators, and voltage references. The MCU LED control lines modulate the IR LEDs while TIAs translate photocurrent into voltages. The MCU obtains photocurrent samples from the ADCs via an SPI port. Each ADC monitors four sensor channels requiring a total of eight ADCs. A series of voltage regulators provide all necessary voltages for the DAU.

4.2.1 Component Selection

4.2.1.1 Operational Amplifier

The op amp in the TIAs must be available in a surface mount IC package and contain two or four op amps in one package. Utilizing surface mount devices reduces the board space required for the DAU circuitry and helps ensure long term availability as more through hole components become obsolete over time.

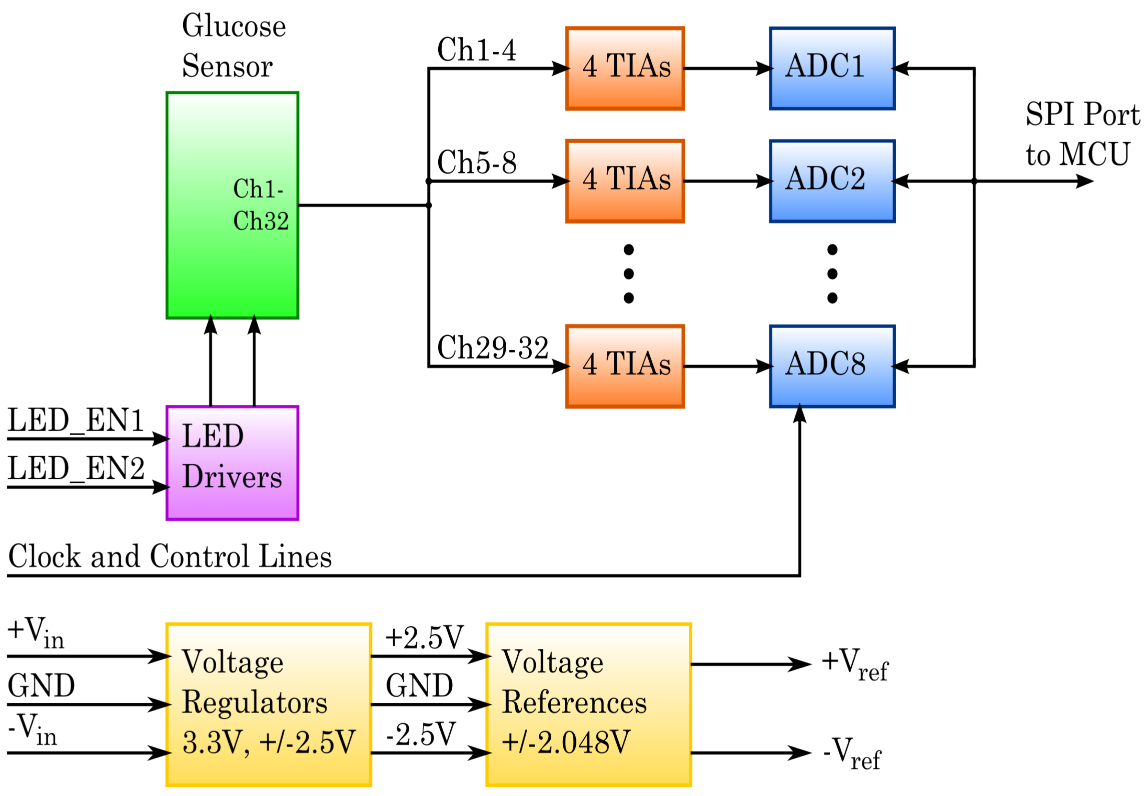


Figure 4-1. ADS1258 DAU block diagram.

The DAU requires 32 op amps and a quad op amp will reduce the board space required and minimize the number of components.

I selected the op amp which contributes the least amount of noise to the system. Table V shows technical data such as input voltage noise, input current noise, offset voltage, input offset voltage stability, and gain bandwidth product for a number of op amps available on the market. Calculations of noise from the shot noise on the offset current, input current noise density, and input voltage noise density, Table VI, assist in op amp selection. The offset current is

$$I_{Offset} = I_{InOffset} + V_{InOffset} / R_{PD}, \tag{4.1}$$

where $I_{InOffset}$ is the op amp input offset current, $V_{InOffset}$ is the op amp input offset voltage, and R_{PD} is the photodiode shunt resistance. Voltage noise density due to noise from the offset current referred to the TIA output,

$$E_{SHOT} = R_F \sqrt{2 I_{Offset} e}, \tag{4.2}$$

where *e* is the electron charge and Δf is the noise bandwidth, is of interest because this current flows through the photodiode causing shot noise. The input current noise density referred to the op amp output is

$$E_{I(BI)} = I_{BI} \; R_F, \tag{4.3}$$

where I_{BI} is the op amp input current noise density. The input voltage noise density referred to the op amp output is

$$E_{E(N\,I)} = E_{N\,I} G_N = E_{N\,I}\left(1 + R_F / R_T\right), \tag{4.4}$$

where E_{NI} is the op amp input voltage noise density. The uncorrelated sum of these noise sources,

$$E_{N,TOTAL} = \sqrt{(E_{SHOT})^2 + (E_{I(BI)})^2 + (E_{E(N\,I)})^2}, \tag{4.5}$$

provides the estimate of op amp voltage noise density. I selected the Maxim MAX4478 because it introduces the lowest noise contribution as shown in Table VI.

4.2.1.2 Feedback Resistor

The glucose sensor photocurrent design goal is 10 nA. The signal voltage $I_{PC}R_F$ must be within the input voltage rage of the ADC. The choice of 10 MΩ for R_F results in a signal voltage of

$$V_{Signal} = I_{PC} R_F = (10\,nA)(10\,M\,\Omega) = 0.1\text{V}, \tag{4.6}$$

Table V. Operational amplifier component data.

Device	Package	Op Amps Per Device	$V_{InOffset}$ (μV), Typ.	E_{IN}(nV/ $\sqrt{Hz}$) @ 1kHz, Typ.	E_{IN} (μV_{P-P}) 0.1-10 Hz, Typ.
AD8599	soic-8	2	10	1.07	-
ADA4004-4	soicn-14	4	40	1.8	-
AD8674	soicn-14	4	20	2.8	-
OP2177	msop-8	2	15	7.9	-
OP4177	tssop-14	4	25	7.9	-
ICL7650	so-14	1	0.7	-	2
LMP7732	msop-8	2	9	3	-
LT1125	sow-16	4	30	2.7	-
LT1127	sow-16	4	30	2.7	-
LT1114	soic-16	4	25	14	-
LTC1053	soicw-18	4	0.5	-	1.5
LT1028	Soic-8	1	20	1	-
LT1128	soic-8	1	20	1	-
LTC2052	soic-14	4	0.5	-	1.5
LT6005	tssop-16	4	190	325	-
LTC6082	tssop-16	4	-	13	-
MAX4208	umax-8	1	3	140	-
MAX4238	sot23-6	1	0.1	30	1.5
MAX4477	umax-8	2	70	4	-
MAX4478	tssop-14	4	70	4	0.26
OP27AFK	lccc-20	1	10	3	-
OPA2228	so-8	2	5	3	-
OPA4228	so-14	4	10	3	-
OPA2380	msop-8	2	4	200	-
OPA277	so-8	1	10	8	-
OPA2277	so-8	2	10	8	-
OPA4277	so-14	4	20	8	-
OPA376	sot23-5	1	5	7.5	-
OPA2376	msop-8	2	5	7.5	-
OPA4376	tssop-14	4	5	7.5	0.8

Table V continued.

Device	$I_{In.Offset}$ (nA), Typ.	I_{NI} (pA/ $\sqrt{Hz}$) @10Hz, Typ.	$V_{In.Offset}$ Stab. (µV/mo), Typ.	$V_{In.Offset}$ Drift (µV /Deg. C), Typ.	GBW (MHz), Typ.
AD8599	25	1.5	-	0.8	10
ADA4004-4	40	3.5	-	0.7	12
AD8674	6	0.3	-	0.3	10
OP2177	0.2	0.2	-	0.2	1.3
OP4177	0.2	0.2	-	0.3	1.3
ICL7650	0.0005	0.01	0.1/ $\sqrt{Mo}$	0.01	-
LMP7732	11	2.3	0.35	0.2	21
LT1125	7	1.3	0.3	0.4	12.5
LT1127	7	1.3	0.3	0.4	65
LT1114	0.06	0.03	0.3	0.4	0.75
LTC1053	0.03	0.0022	0.05/ $\sqrt{Mo}$	0.01	2.5
LT1028	18	1	0.3	0.2	75
LT1128	18	1	0.3	0.2	75
LTC2052	-	-	0.05/ $\sqrt{Mo}$	0.01	3
LT6005	0.005	0.0012	-	2	0.002
LTC6082	0.0001	-	-	0.2	3.6
MAX4208	0.001	-	-	0.1	0.75
MAX4238	0.002	-	0.05	0.01	1
MAX4477	0.001	0.0005	-	0.3	10
MAX4478	0.001	0.0005	-	0.3	10
OP27AFK	7	5	0.2	0.2	8
OPA2228	2.5	0.4	0.2	0.1	8
OPA4228	2.5	0.4	0.2	0.3	8
OPA2380	0.006	0.01	-	0.03	90
OPA277	0.5	0.2	0.2	0.1	1
OPA2277	0.5	0.2	0.2	0.1	1
OPA4277	0.5	0.2	0.2	0.15	1
OPA376	0.0002	0.0002	-	0.26	5.5
OPA2376	0.0002	0.0002	-	0.26	5.5
OPA4376	0.0002	0.0002	-	0.26	5.5

Table VI. Operational amplifier selection calculations.

Device	BW (kHz)	Ioffset (nA)	$V_{Shot(Ioffset)}$ ($\mu V/\sqrt{Hz}$)	$V_{N,I(NI)}$ ($\mu V/\sqrt{Hz}$)	$V_{N,V(NI)}$ ($\mu V/\sqrt{Hz}$)	Sum ($\mu V/\sqrt{Hz}$)
AD8599	30	25.33	0.90	15	0.4	15.0
ADA4004-4	36	41.33	1.15	35	0.6	35.0
AD8674	30	6.67	0.46	3.0	0.9	3.2
OP2177	3.9	0.7	0.15	2.0	2.6	3.3
OP4177	3.9	1.03	0.18	2.0	2.6	3.3
LT1125	37.5	8.00	0.51	13	0.9	13.0
LT1127	195	8.00	0.51	13	0.9	13.0
LT1114	2.25	0.89	0.17	0.30	4.7	4.7
LT6005	0.01	6.34	0.45	0.012	108.7	108.7
LTC6082	10.8	2.33	0.27	0.005	4.3	4.4
MAX4477	30	2.33	0.27	0.005	1.3	1.4
MAX4478	30	2.33	0.27	0.005	1.3	1.4
OP27AFK	24	7.33	0.48	50	1.0	50.0
OPA2228	24	2.67	0.29	4.0	1.0	4.1
OPA4228	24	2.83	0.30	4.0	1.0	4.1
OPA2380	270	0.14	0.07	0.10	66.9	66.9
OPA2277	3	0.83	0.16	2.0	2.7	3.3
OPA4277	3	1.17	0.19	2.0	2.7	3.3
OPA2376	16.5	0.17	0.07	0.002	2.5	2.5
OPA4376	16.5	0.17	0.07	0.002	2.5	2.5

within the input voltage range of the ADC. The feedback resistor must be a metal film resistor because in general metal film resistors exhibit lower noise than other resistor types[37].

4.2.1.3 Feedback Capacitor

The TIA feedback capacitor rolls off the TIA gain and also determines how fast the TIA responds to a step change in input current. The ADC passband repeats at multiples of 8 MHz requiring an anti-aliasing filter cutoff below 4

MHz to prevent aliasing. A standard SAR ADC would require anti-aliasing cutoff frequency of half the sample rate, but the oversampling sigma delta converter on the DAU effectively attenuates input signals between the ADC bandwidth and a frequency of 8 MHz minus the ADC bandwidth. The minimum C_F to prevent aliasing is

$$C_F = \frac{1}{2\pi R_F f_{Cutoff}} = \frac{1}{2\pi(10\text{M}\Omega)(4\text{MHz})} = 4\,fF\,, \tag{4.7}$$

much lower than stray capacitance of about 0.5 to 1 pF, ensuring aliasing is unlikely. Increasing C_F above this value improves rejection of noise outside the converter bandwidth, but the TIA must be able to respond to the step change in input current when the LED turns on or off. After a step change in input current the TIA output changes with a time constant R_FC_F - the longer the TIA settles the closer it approaches its final value. Allowing two time constants between toggling the LED and starting an ADC sample allows the TIA to reach 87% of its final value. Since one sample period is 0.41 ms the feedback capacitance must be

$$C_F = \frac{0.5\,t_{RC}}{R_F} = \frac{0.205\,ms}{10\,M\,\Omega} = 20\,pF \tag{4.8}$$

to allow two time constants during one ADC sample period. Experiments with an increased sample rate required reducing C_F to the final value of 10 pF for a TIA voltage bandwidth of

$$f_{TIA} = \frac{1}{2\pi R_F C_F} = \frac{1}{2\pi(10\,M\,\Omega)(10\,pF)} = 1590\,Hz. \tag{4.9}$$

4.2.1.4 Analog to Digital Converter

The ADC must meet the following constraints to be used on the DAU:

- Surface mount device
- 4/8/16 Channel device
- Sampling rate 200 to 2000 samples per second, minimum
- Serial port interface
- 20/24 bit resolution

Surface mount devices are required as with the op amps. The ADC will need to be able to measure several channels to minimize the parts count and board space. The MCU will send a set of data to the PC at approximately 1 Hz and record perhaps 50 to 500 samples for all 32 channels, with each LED on and off. The minimum sample rate without multiplexing ADC channels is then approximately 4 times 50 to 500 samples per second, or 200 to 2000 Hz without multiplexing. If multiplexing channels is required the minimum sampling rate increases accordingly. For example if there are four channels per ADC and the ADC has one ADC block, the minimum sample rate increases by a factor of four. The ADC will need a serial port to send sample data to the MCU since a limited number of digital I/O lines are available between the DAU and MCU. The ADC must have a minimum of 20 bits resolution to ensure the quantization noise is small relative to the system noise requirement.

Criteria for choosing the ADC are listed below.

- Maximum number of ADC blocks
- Ability to daisy-chain serial ports
- Maximum sampling speed

- Bipolar inputs

Technical specifications for several commercially available ADCs are listed in Table VII. The number of ADC blocks inside a converter affects the amount of time required to read all input channels. If each input has a dedicated ADC multiplexing is not required and system throughput is maximized. An SPI port requires a minimum of three pins including data out, data in, and serial clock. The addition of a chip select line for each device allows several integrated circuits to share the same SPI port. Some manufacturers allow several devices to share an SPI port using only one chip select - this capability minimizes the number of I/O pins required by the DAU. A faster sampling speed allows higher

Table VII. ADC component data.

Device	Package	Sample Rate	Channels	DS Blocks
AD7190	tssop-24	4800 SPS	4	1
ADS1211	ssop-28	1000	4	1
ADS1218	tqfp-48	390	8	1
ADS1258	qfn-48	23700	16	1
ADS1278	htqfp-64	128000	8	8
CS5528	ssop-24	617	8	1
LTC2408	ssop-28	7	8	1
LTC2418	ssop-28	7.5	16	1
LTC2444	qfn-38	8000	8	1
LTC2445	qfn-38	8000	8	1
LTC2448	qfn-38	8000	16	1
LTC2498	qfn-38	7.5	16	1
MAX11040	tssop-38	16000	4	4

sampling rate and LIA reference frequency which limits 1/f noise. Also higher sampling frequency can reduce the hardware complexity because a lower number of faster converters can process the same number of channels.

Since the photodiode array is in the common cathode configuration, the PD cathodes are at ground and photocurrent travels out of the PD anode, forcing the TIA output to fall below ground. The TIA amplifies the op amp input offset voltage and therefore the TIA output voltage may be positive or negative. A single ended bipolar input ADC easily handles this situation. Single ended unipolar or differential ADCs can be used with proper attention to supply and reference biasing.

I chose the ADS1258 for use on the DAU. This ADC has a 23.7 kHz sample rate, 16 inputs, one ADC block, and bipolar inputs. On the DAU each ADC measures four TIA channels which requires eight ADCs. I decided to use eight ADCs because with sixteen ADCs the parts cost and board complexity would be high and with four ADCs the reference frequency would be about 300 Hz, too low to reduce the effect of 1/f noise from op amp input voltage noise.

4.2.1.5 Voltage Reference

The voltage reference must be a surface mount device and have the minimum possible voltage noise. The present DAU requires positive and negative reference voltages. Table VIII presents several voltage references and their noise voltage from 0.1 to 10 Hz – the table shows that the ADR440 has the lowest noise voltage.

Table VIII. Voltage reference component data.

Device	Noise, µVp-p 0.1-10 Hz
REF3125	33
REF5025	7.5
VRE3025	1.5
MAX6126-2.5	1.45
ADR440	1

4.2.2 Printed Circuit Board Design

The DAU has six copper layers and measures approximately 4.25” by 4.25” with overall thickness 1/16”. Board layout design guidelines of minimum trace width 0.008”, minimum trace spacing 0.008”, and minimum via size 0.010” are within the capabilities of most printed circuit manufacturers. The layer stackup, Fig. 4-2, consists of a top layer for components and routing, layers 2 and 4 for ground planes, and layer 3, layer 5, and the bottom layer for routing signals. Layers 2 and 4 are identical, consisting of split analog and digital ground planes. The top layer contains pads for installing devices and space for routing analog and digital signals. The ADCs require 16 MHz clock sources - routing these signals on layer 3 between digital ground planes isolates them from the rest of the board. Locating the remainder of the digital lines below the digital ground planes separates then from analog signals on the top layer. See Appendix B for the ADS1258 DAU schematic. The parts placement process started with placing the glucose sensor in the center of the board, towards the top edge in the vertical orientation. This prevents air bubbles in the tubing from collecting inside the

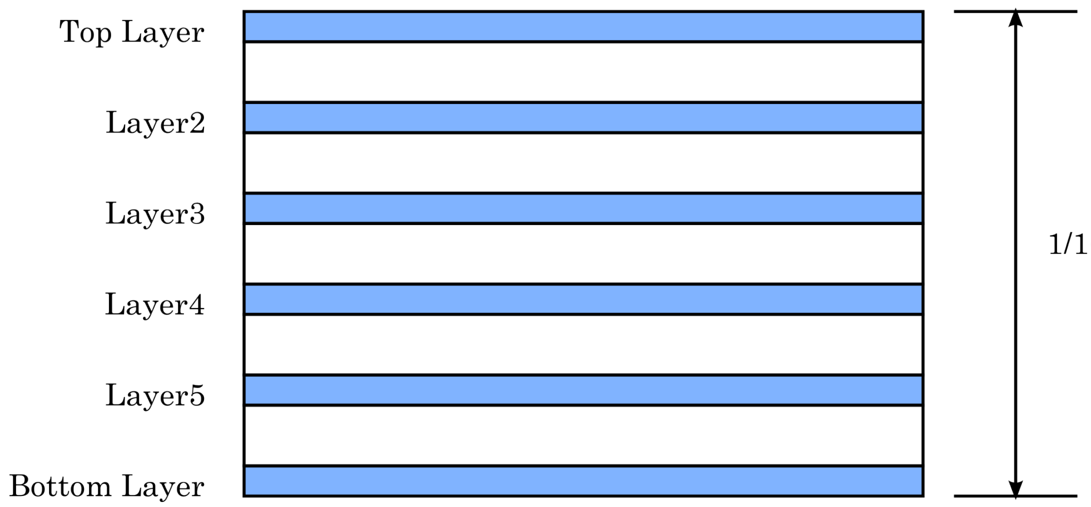


Figure 4-2. DAU printed circuit card layer definition.

sensor which causes large transients in the sensor data. A ZIF socket for the sensor allows easy removal and installation. Mounting the sensor on the bottom side reduces the possibility of getting fluid on the MCU in case of a leak. Locating the TIAs around the around the glucose sensor minimizes length of analog signal traces and allows creating the analog ground plane in the center of the board. The digital ground plane along the left, right, and lower edge of the board, providing room for LED drivers and other digital components.

4.2.3 Power Dissipation

Estimation of DAU power dissipation facilitates choice of voltage regulators and assists development of future photocurrent monitoring devices by providing a basis for designing future system configurations. Future versions of the monitoring device will operate on batteries and require re-charging perhaps once per day, requiring very low power operation. A power budget estimating the current required by the DAU is in Table IX. Actual current draw for the positive and negative supplies are 350 mA and 340 mA.

Table IX. Power budget for DAU3. All data are in mA.

Device	Quantity	I per Device	Positive Supply	Negative Supply
ADS1258, Analog	8	12	96	96
ADS1258, Digital	8	0.6	4.8	0
MAX4478	9	10.0	90	90
ADR440	2	3.75	7.5	7.5
MAX16803	2	3	6	6
LED	2	50	100	100
OPA2350	8	10.4	83.2	83.2
OPA365	8	5	40	40
Total	-	-	427.5	422.7

4.3 MCU Firmware

The MCU must read samples from ADCs on the DAU via the SPI bus, filter the samples, and send the resulting data points to the base station via USB or wireless link for further processing. The MCU software, written in C using the MPLAB IDE V8.1, is included in Appendix C. The software begins by initializing variables, the SPI port, and all eight ADCs, see the MCU firmware block diagram in Fig. 4-3. The main loop consists of processing samples with LED1 on, LED1 off, LED2 on, and LED2 off in succession and sending filtered data to the base station once per second.

A timing diagram for the sample acquisition process, Fig. 4-4, illustrates the MCU sampling procedure. Each ADC measures TIA outputs on four of its analog inputs, requiring the ADC multiplexer to change channels periodically. I included two sampling periods after channel switching to allow a brief ADC settling time. Then the ADC records photocurrent samples for LEDs 1 and 2 with the LEDs on and off. For each of these four sample categories, the ADC acquires two samples. The MCU discards the first sample and this time delay allows the TIA to approach its final output value for two $R_F C_F$ time constants. The lock-in amplifier filters the second sample using the combined lock-in filtering method. The firmware main loop cycles through all four analog inputs for eight ADCs for

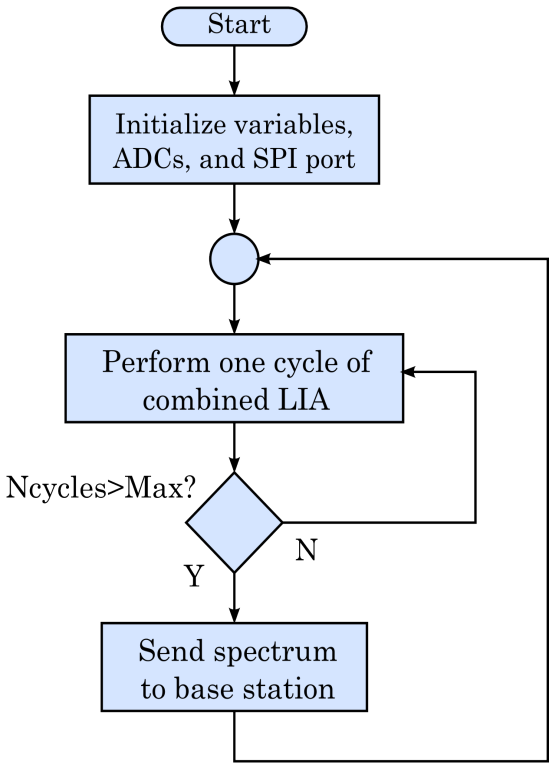

Start
Initialize variables, ADCs, and SPI port
Perform one cycle of combined LIA
Ncycles>Max?
N
Y
Send spectrum to base station

Figure 4-3. MCU firmware flowchart.

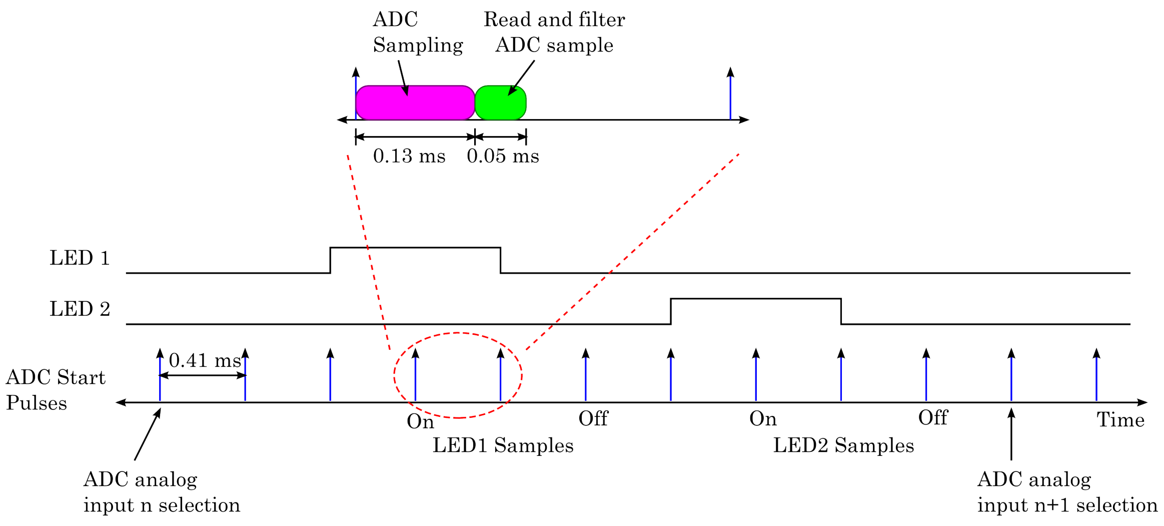
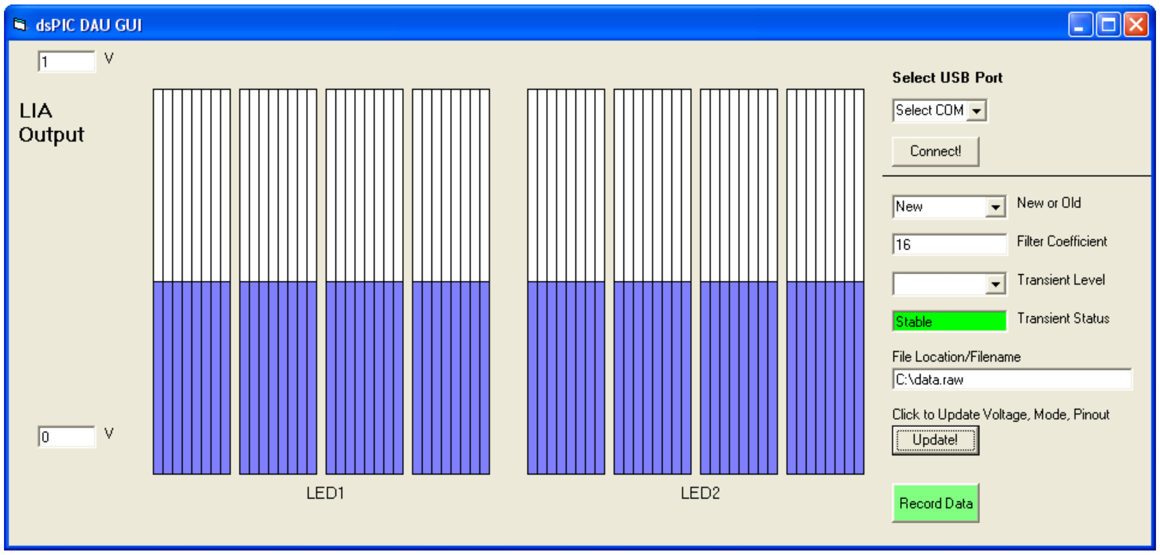


Figure 4-4. ADS1258 configuration firmware timing diagram.

one second and then sends the filtered photocurrent spectra to the base station.

4.4 Base Station Software

A graphical user interface written in Visual Basic 2006 reads glucose sensor data from the MCU USB or wireless port, displays it on the computer screen, and archives it for further processing. The base station software is included in Appendix D. Most of the user interface window, see Figure 4-5, consists of bar graphs displaying data points for all 32 channels and both LEDs. At any time the user adjusts the maximum and minimum voltages displayed by entering new maximum/minimum values on the left edge of the window and clicking the “Update” button. The software allows the user to select the MCU port, the sensor pinout (old or new), filter bandwidth, transient detection level, and output filename. After the MCU starts sending data the PC software enters a



dsPIC DAU GUI

1 V

LIA Output

0 V

LED1

LED2

Select USB Port

Select COM

Connect!

New — New or Old

16 — Filter Coefficient

Transient Level

Stable — Transient Status

File Location/Filename

C:\data.raw

Click to Update Voltage, Mode, Pinout

Update!

Record Data

Figure 4-5. The base station software window.

loop filtering, displaying, and recording sample data and transient indication.

4.5 ADS1278 DAU Configuration

The ADS1278 DAU configuration contains four ADS1278 ADCs and each ADC measures TIA output voltages for eight glucose sensor channels, see the block diagram in Fig. 4-6. Power conditioning circuitry generates a +2.5 V voltage reference and a +1.8 V power supply for the ADC core voltage. An SPI port transfers sample data to the MCU for filtering. A schematic diagram of the ADS1278 DAU is included in Appendix E. The ADS1278 ADC measures the differential voltage between each positive analog input pin and negative input pin relative to the differential voltage between the positive and negative reference voltage inputs but both analog input pin voltages must be above ground.

The ADS1278 ADC exhibits a latency, or delay, of 40 sample periods before the output samples accurately represent the input voltage. The MCU firmware accommodates this by recording ten samples with LED1 on, all LEDs off, LED2 on, and all LEDs off as shown in Fig. 4-7. The MCU firmware records the fifth sample of each group of ten samples with an ADC sample rate of 781.25 Hz which results in lock-in cycle length of 25.6 ms and a reference frequency of 39.1 Hz.

4.6 Design Notes

4.6.1 Metal Film Resistors

Metal film resistors exhibit less noise than other resistor types such as thick film and carbon composition resistors. Metal film resistors cost more than thick film resistors but where the SNR is a concern the use of metal film resistors is required. Thus the TIA uses metal film feedback resistors as any excess noise reduces the SNR.

4.6.2 Op Amp Input Voltage Range

Since all photodiodes have a common cathode the TIA output falls to $-I_{PC}R_F$ when the LEDs turn on. The op amp power supplies and input voltage range must not limit output voltage for the expected range of photocurrent.

4.6.3 Op Amp Non-inverting Input Voltage

The op amp non-inverting input voltage must be a low noise voltage source such as ground potential. If the non-inverting TIA input is driven at ½ the analog supply voltage for an op amp operating from +AVCC to ground, the TIA

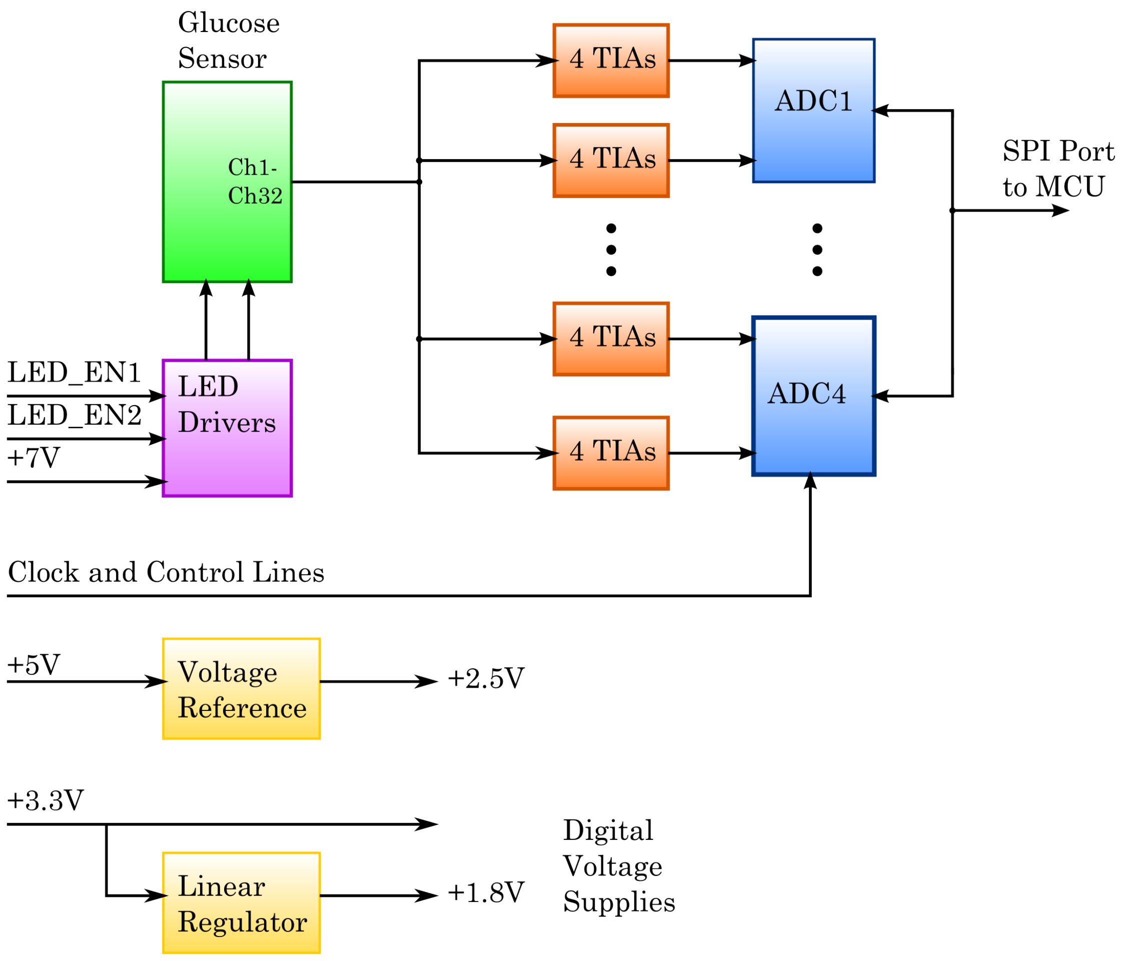


Figure 4-6. A block diagram of the ADS1278 DAU configuration.

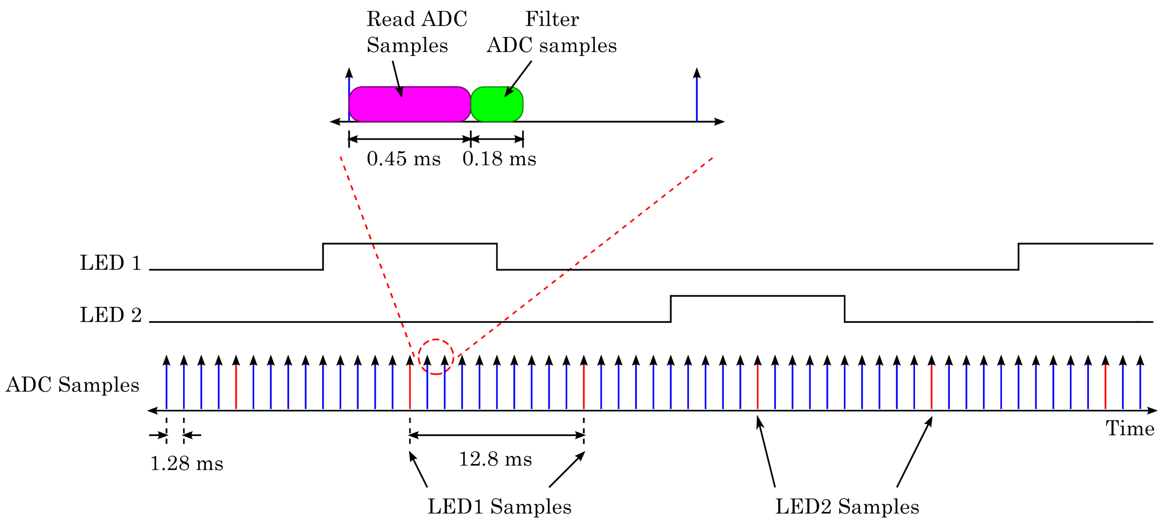


Figure 4-7. ADS1278 DAU configuration timing diagram.

amplifies noise present on the +AVCC/2 voltage. Two alternatives avoiding this problem are (1) using bipolar input op amps and ADCs with the non-inverting TIA input at ground and (2) using single ended input op amps and ADCs with two voltage sources supplying ground, +AVCC/2, and +AVCC with the non-inverting op amp input at +AVCC/2 and powering the ICs from ground and +AVCC.

4.6.4 Build a Demonstration System

Construction of a demonstration system has several benefits. It allows the designer to verify technical specifications of ICs, determine whether circuit configurations will operate correctly, and begin software development at an early stage in the design process.

4.6.5 Prevent Crosstalk

PC card designs must include provisions to prevent cross talk between signals. For example, high frequency digital signals and clocks must be separated from other signals including analog signals to prevent contamination of low level analog signals. Ground planes between signals, routing signals on adjacent planes at 90 degree angles, and increasing distance between traces prevent cross talk.

4.6.6 Prevent Ground Loops

Reducing the area of or elimination ground loops or current paths reduces the amount of noise induced into and radiated by the circuit path. Ensuring that the trace carrying current to a device is above the trace or plane carrying the return current from the device minimizes loop area of that current path reducing possibility of electromagnetic interference.

4.6.7 Lock-in Reference Frequency

Proper choice of the lock-in reference frequency enables reducing the effect of 1/f noise on system noise performance. The combined lock-in amplifier allows increasing the reference frequency high enough to reduce the contribution of 1/f noise in the op amp input voltage noise, increasing system SNR.

4.6.8 ADS1258 ADC configuration

The ADS1258 data sheet illustrates using a buffer between the multiplexer and the actual ADC modulator inputs and shows an example method for driving the voltage reference inputs with an op amp follower. Application of these additional components improved consistency of SNR across all DAU channels, but required an additional 24 op amps impacting the DAU board area and power requirements.

4.7 Summary

The DAU and MCU implement the combined LIA filtering method with firmware written in C and base station software written in Visual Basic 2006. DAU component selection minimizes electrical noise from each device - the device search included numerous op amp, voltage references, and ADCs. The six layer printed circuit card layout also minimizes noise by reducing crosstalk between digital and analog signals. Subsequent chapters document testing and evaluation of the DAU and MCU for continuous glucose monitoring.

CHAPTER 5

EXPERIMENTS AND ANALYSIS

Experimental measurements in this chapter evaluate the capability of the monitoring system to provide low noise measurements acceptable for use with the glucose monitoring system. Initial measurements with metal film resistors in place of photodiodes, see Fig. 5-1, enables measurement of offsets, system SNR, channel isolation, and electronics stability. Deviation from zero Volts present in lock-in amplifier outputs without illumination could be tolerated if the offsets do not change, but offsets that vary with time or from experiment to experiment cannot occur as time varying offsets appear as changes in analyte concentration. The measurement system must provide at least 40 dB SNR with 30 kΩ shunt resistance photodiodes providing 10 nA of photocurrent. Crosstalk between DAU channels indicates poor channel to channel isolation - investigation of

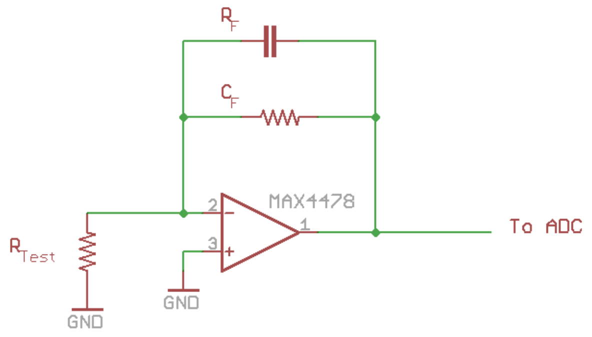


Figure 5-1. Utilization of metal film resistors to simulate photodiodes.

correlation coefficients for all channel pairs ensures independent data channels. Recording system outputs over several hours determines whether the electronics induce drift over time. Subsequent experiments with IR LEDs, variable filters, and photodiode arrays with shunt resistance of 10 kΩ to 30 kΩ prove the system ability to measure photocurrent with acceptable SNR.

5.1 DC Offsets Using Metal Film Resistors

The lock-in amplifier output voltage is proportional to $I_{PC}R_{F,}$ therefore when there is no photocurrent the LIA output must be 0 Volts, ignoring noise. Thus any significant DC offset voltage represents measurement error as the LIAs pass only the portion of the input signal modulated at the reference frequency. A DC shift in any sensor channel with no photocurrent sends erroneous spectra to the calibration routine. The ADS1258 DAU lock-in amplifier channels exhibited DC offsets of 50 to 500 μV which changed with each power cycle, see Fig. 5-2(a), rendering the configuration unacceptable for measuring absorption spectra.

Significant effort expended to eliminate ADS1258 offsets achieved little success. The ADC contains a multiplexer before the actual delta-sigma ADC with the multiplexer outputs and ADC inputs available on ADC pins. Advice from the device manufacturer on obtaining specified device performance includes the addition of an external buffer between the multiplexer and ADC and buffering the voltage reference to the ADC with another external buffer. Neither the addition of single ended buffers nor differential buffers between the multiplexer and ADC inputs removed the offset error. Buffering the voltage reference as indicated did not remove the measurement error provided by the ADS1258 configuration. Schematics of the single ended and differential buffers between the multiplexers and ADC inputs as well as the buffers for each ADS1258 voltage reference input are depicted in Fig. 5.3. Other attempts to improve offset performance of the ADS1258 configuration include:

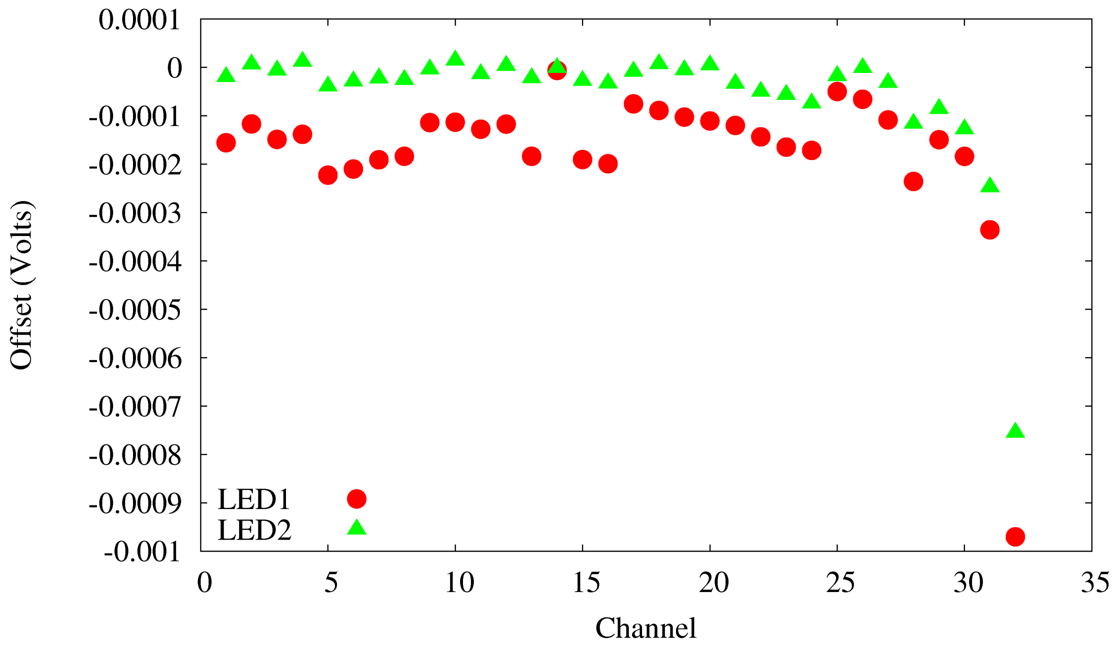
- Use low noise laboratory power supply to provide voltage reference.
- Utilize low noise laboratory supply to supply +/-2.5 Volt analog supplies.
- Increase RC time constant of RC filters in voltage reference, reduce resistance in RC filters ensuring the ADC1258 voltage reference current draw does not load filters.
- Investigate use of alternate component values for the 47 Ohm resistor and 2200 pF capacitor between the voltage reference buffer and ADC voltage reference pins. The capacitor helps the ADC input capacitor charge up during the acquisition time and the resistor isolates the capacitor from the op amp output - these components are not designed to be an RC filter.
- Measure another set of samples after LED1 and LED2 samples and subtract this data from LED1 and LED2. The DC offsets can change from power cycle to power cycle but are relatively stable during each data collection period. Subtraction of the third data set did not eliminate the offsets.
- Monitor the internal ADS1258 voltage reference. Periodic fluctuations on the external voltage reference were noted - fluctuations in voltage

reference between LED on and LED off samples with constant voltages at ADC analog inputs cause the offsets in lock-in amplifier data. Neither modifying the reference voltage circuit nor modifying the software to alter the data collection periodicity significantly reduced the offsets. The ADC does not allow use of the internal reference for recording samples.

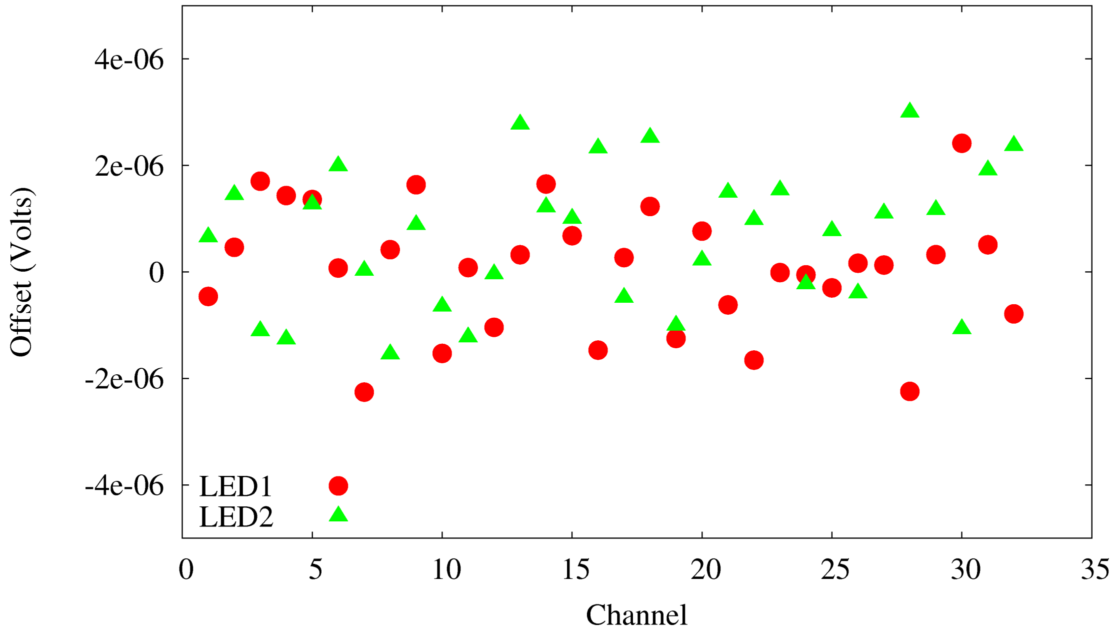
As shown in Fig. 5-2(b) the ADS1278 configuration offsets are within a few μV of ground, much lower than for the ADS1258 configuration with offset errors of a few hundred μV to nearly 1 mV. therefore the ADS1278 configuration was selected for use with the measurement system. The ADS1278 configuration has an ADC clock configuration of 2 MHz, much lower than the 16 MHz ADC clock of the ADS1258 configuration. Eight ADCs on the ADS1258 configuration require the ADC clock to function and although these clock lines sit between two digital ground planes, routing the ADC clock lines to all eight ADCs could cause excess 16 MHz noise on the ADS1258 PC card. The ADC allows a 16 MHz signal to pass and alias since its passband repeats at multiples of 8 MHz, half the ADC clock frequency. With only four ADCs and a lower ADC clock frequency the ADS1278 configuration has less susceptibility to ADC clock noise passing through the ADC.

5.2 Signal to Noise Ratio Measurement with Metal Film Resistors

Measurement of the system SNR with metal film resistors in place of the photodiode array verifies whether the combined LIA measurement method follows the noise model prediction. A PC recorded at least 1000 monitoring



(a) ADS1258 Configuration



(b) ADS1278 Configuration

Figure 5-2 Offsets. (a) ADS1258 configuration. (b) ADS1278 configuration.

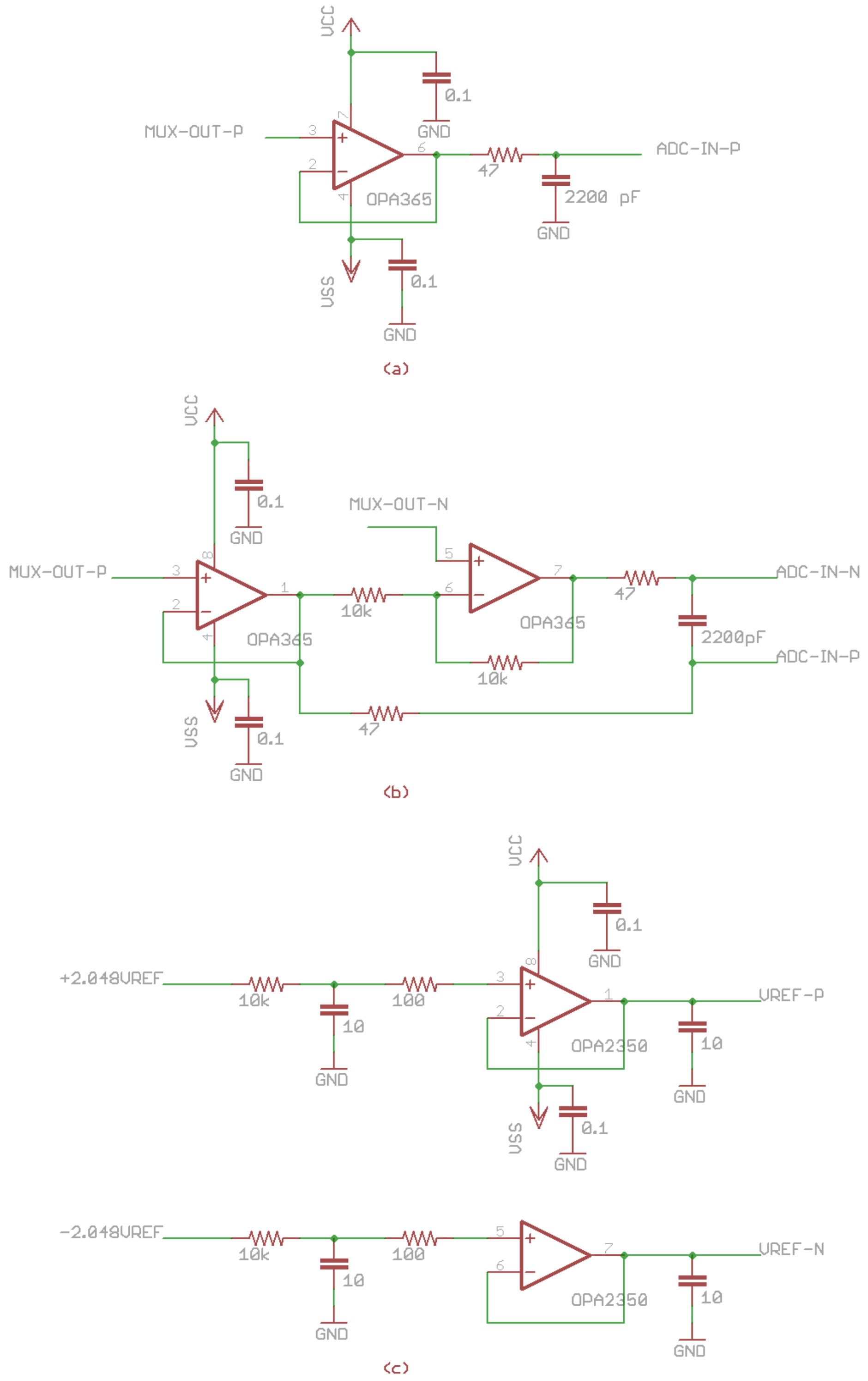


Figure 5-3. Circuitry included with ADS1258 DAU configuration. (a) Single ended and (b) differential buffers between each ADC multiplexer output and ADC input. (c) Buffers for voltage reference inputs.

system samples with R_{Test} values of 10 kΩ, 30 kΩ, 100 kΩ, 300 kΩ, 1 MΩ, and an open circuit (approximately 1E9 Ω) for all channels. A warm up time of at least 30 minutes with the monitoring system fully operational before retaining data allows temperature stabilization of the DAU and MCU.

Figure 5-4 shows a plot of mean experimental SNR measurements for the ADS1278 configuration with SNR in dB calculated using

$$SNR_{dB} = 10\,log SNR = 10\log\frac{I_{PC}R_F}{\sigma}, \tag{5.1}$$

where σ is the sample standard deviation of 1000 data points. The experimental SNR data with metal film resistors match the noise model well for R_{Test} of 300 kΩ and below, but the SNR drops slightly for R_{Test} = 1 MΩ and drops about 3 dB for an open circuit. One noise source not considered in the noise model is the 2.5V reference voltage applied to the non-inverting op amp input in the TIA, but this noise source would appear as additional op amp input voltage noise and reduce SNR for lower R_{Test} values before affecting that of larger R_{Test} and cannot account for the excess noise. Modifying the noise model to include a n 0.8 MΩ resistor in parallel with the test resistor, shown in Fig. 5-4 as the modified noise model, minimized the mean error relative to the metal film resistor experimental data. Thus a leakage resistance, $R_{Contamination}$, due to flux residue or other contaminants reduces the effective value of R_{Test}. For R_{Test} below 1MΩ the parallel combination of R_{Test} and the contamination resistance,

$$R_{Test,\,Effective} = \frac{R_{Test}\,R_{Contamination}}{R_{Test} + R_{Contamination}}, \tag{5.2}$$

is close to R_{Test} but it is limited to $R_{Contamination}$ with high R_{Test}. Including guard rings around the op amp inputs will reduce susceptibility to leakage currents.

5.3 Stability Investigation Using Metal Film Resistors

Stability of DAU photocurrent measurements over time ensures glucose concentration measurements are free of electronics drift. Ambient air temperature changes can cause electronics drift in the monitoring system due to the temperature coefficient of the voltage reference. Heating and cooling systems modulate air temperature with periods of tens of minutes possibly affecting

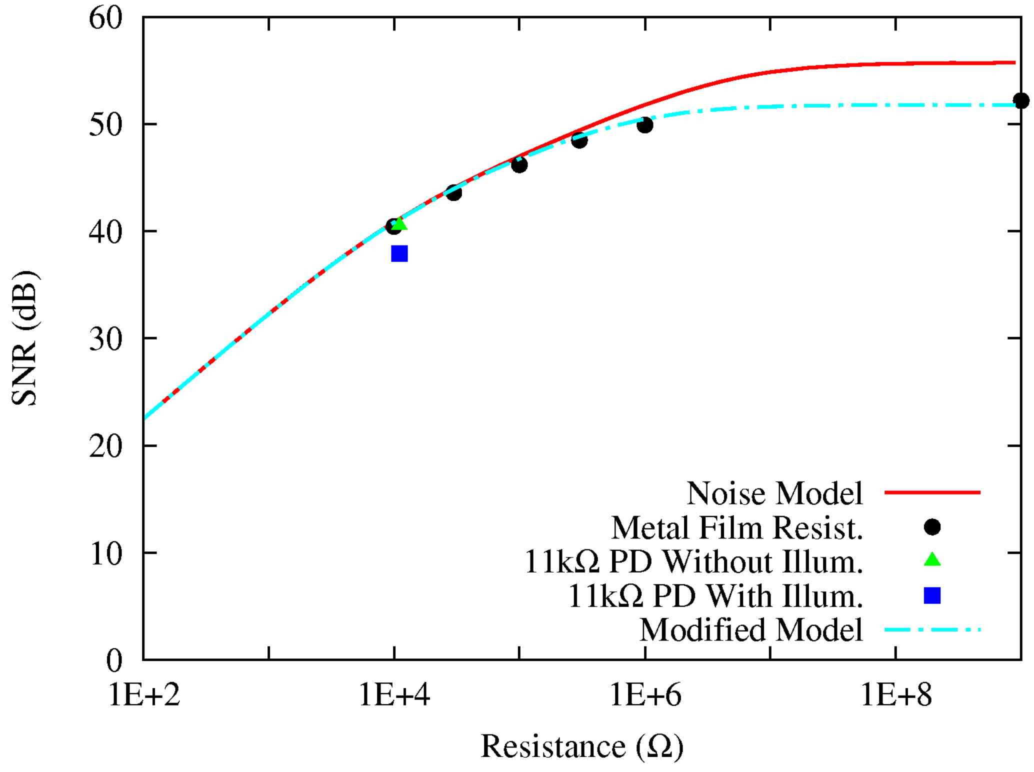
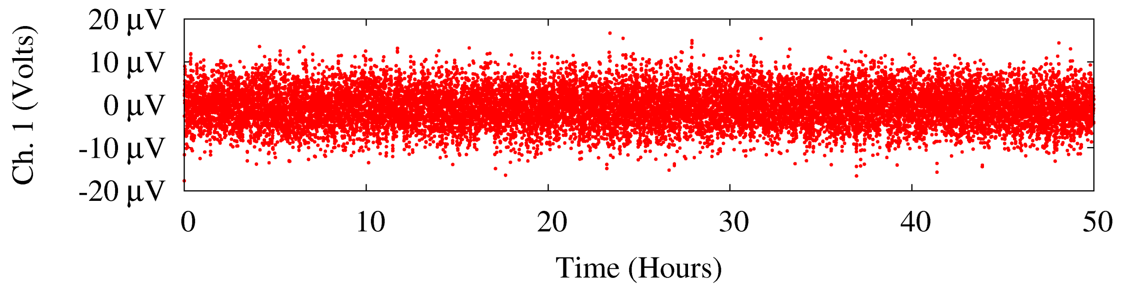


Figure 5-4. Experimental SNR results and noise model for ADS1278 DAU.

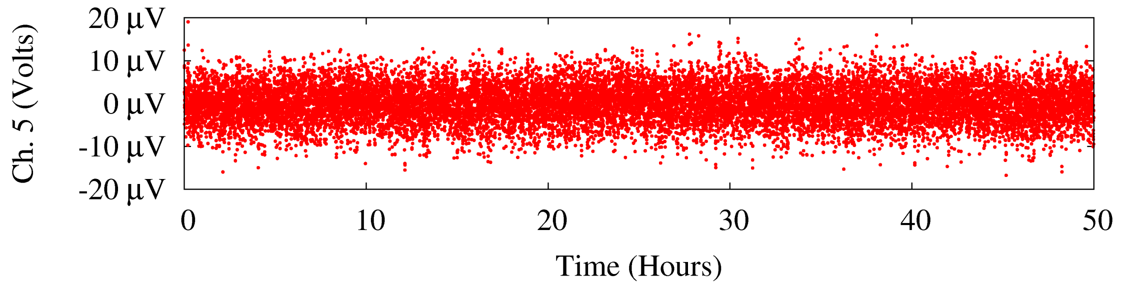
the ADC samples over time. Factors such as op amp offset voltage and voltage reference magnitude can shift over weeks or months, but techniques such as periodic system calibration, lock-in detection, and ratiometric measurement reduce or eliminate these effects. A plot of DAU sample data over two days evaluates whether drift is present in the DAU electronics. Fig. 5-5 shows plots of several representative LED1 channels using 30 kΩ metal film resistors for R_{Test} and a 0.017 Hz bandwidth. The plots show a mean SNR of 43.4 dB relative to 0.1V signal without significant drift, matching the noise model prediction of 44.0 dB very well. Therefore the monitoring system electronics do not exhibit drift and do maintain SNR performance over a period of days. Also the electronics do not require periodic calibration while in operation - the measurement system ran continually over the two day period without interrupting data logging to record or reset any software data or any type of calibration data.

5.4 Cross-correlation with Metal Film Resistors

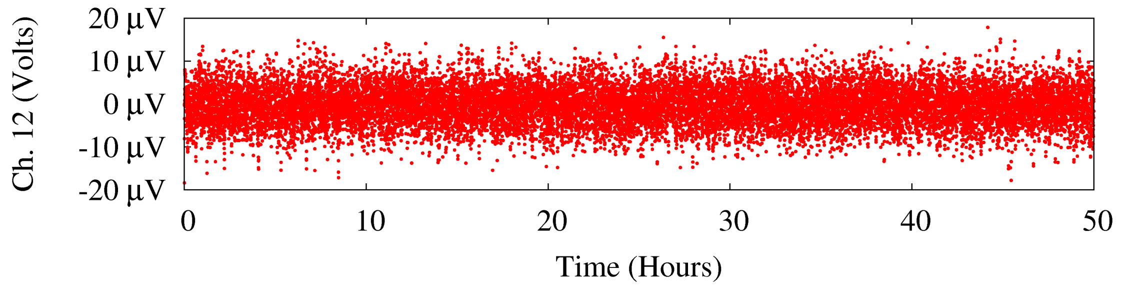
In the ideal multi-channel data acquisition system samples from each channel are completely independent and therefore uncorrelated. The ADS1278 DAU configuration contains four ADCs and eight op amp ICs with separately filtered voltage references for each op amp IC and ADC and hence some similarities between channels over time are possible. Fluctuations in LED output intensity over time affect all photodiodes simultaneously and also contributes to correlations between photodiode channels. Calculation of the Pearson correlation coefficient[43] between two vectors X and Y,



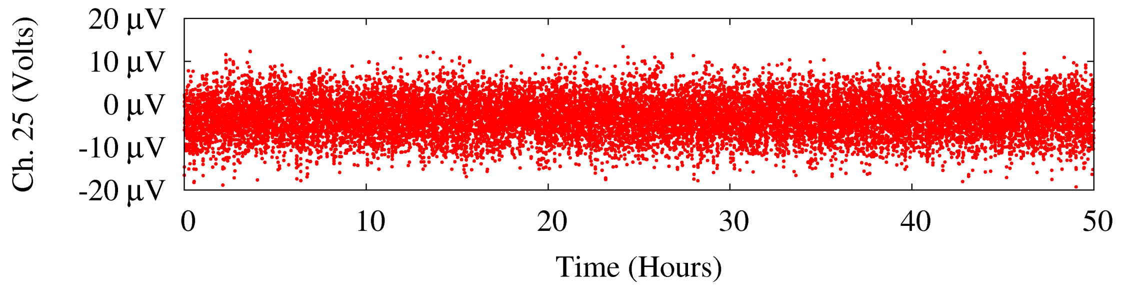
(a) Channel 1. σ = 4.32 μV.



(b) Channel 5. σ = 4.33 μV.



(c) Channel 12. σ = 4.49 μV.



(d) Channel 25. σ = 4.53 μV.

Figure 5-5. Plot of several representative DAU channels with R_{Test} = 30 kΩ and bandwidth of 0.017 Hz over two days showing lack of electronics drift.

$$r_{X,Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}, \tag{5.3}$$

where $\mu_X(\mu_Y)$ is mean of X(Y), $\sigma_X(\sigma_Y)$ is the standard deviation of X(Y), and E[a] is the mean value of a, estimates similarity of the two vectors over time. The correlation coefficient of a vector with itself equals 1 since the standard deviation is

$$\sigma_X = \sqrt{E(X-\mu_X)^2}. \tag{5.4}$$

A three dimensional plot of r_{AB}, the correlation coefficient, versus channel A and channel B where A and B include all LED1 and LED2 photodiode channels, Fig. 5-6, shows correlation coefficients from approximately -0.4 to 0.4. Two plots of channel pairs with the highest correlation coefficients, Fig. 5-7, shows these channels do exhibit periods with similar gradual changes but the channels do not closely follow each other for extended amounts of time. Perturbations of the voltage reference likely causes the correlations between channels on the card without illumination. Comparing correlations between LED1 and LED2 for each channel shows no high correlations. Thus the monitoring system channels provide independent samples. The personal computer software for calculation of the cross-correlation coefficients is included in Appendix G.

5.5 Signal to Noise Ratio Measurement with Variable Filter, PD Array, and IR LEDs

A 40 pin DIP IC header with 11 kΩ shunt resistance photodiode array and linear variable bandpass filter facilitated SNR experiments while completed glucose sensors were under development. The mean SNR for all channels without

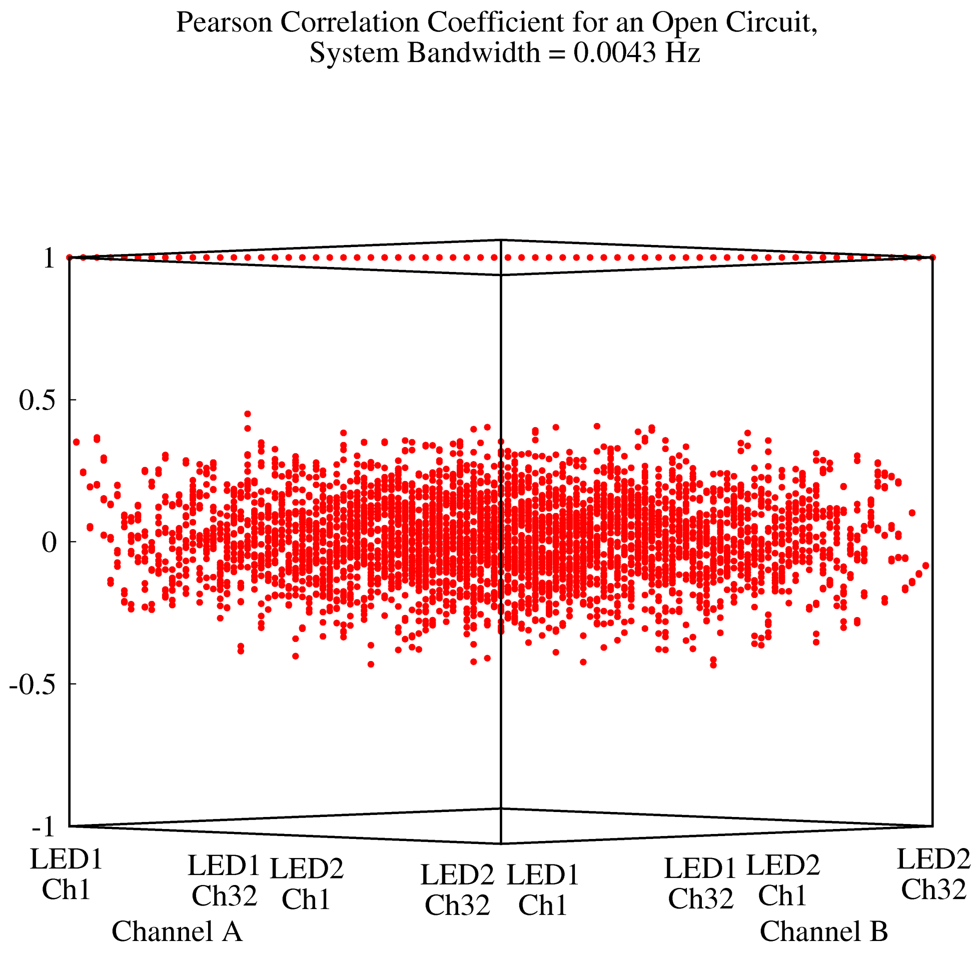
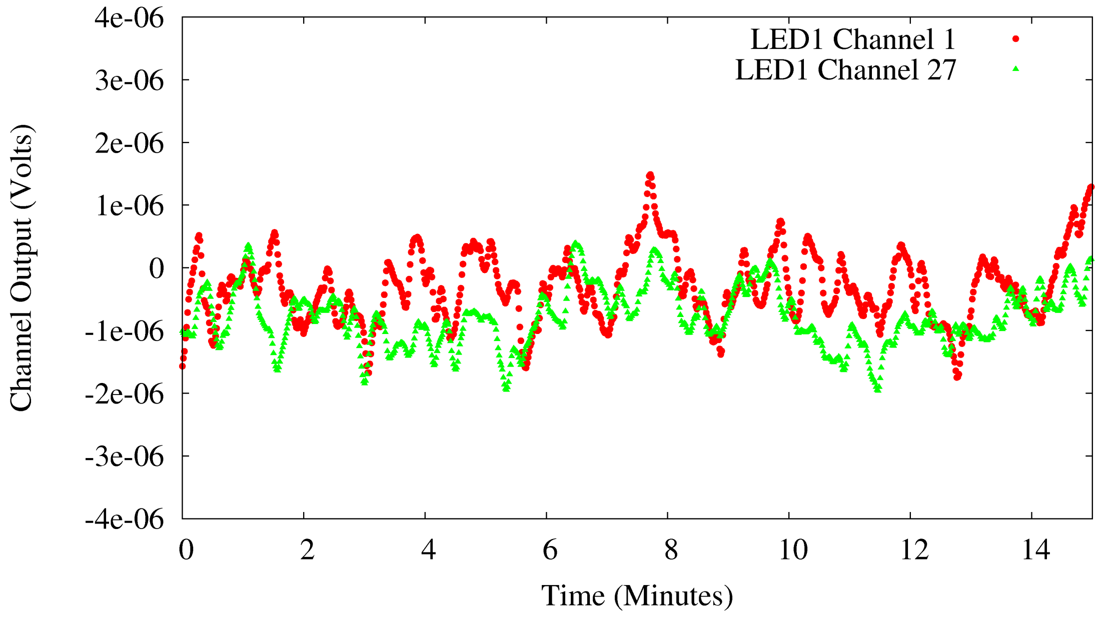
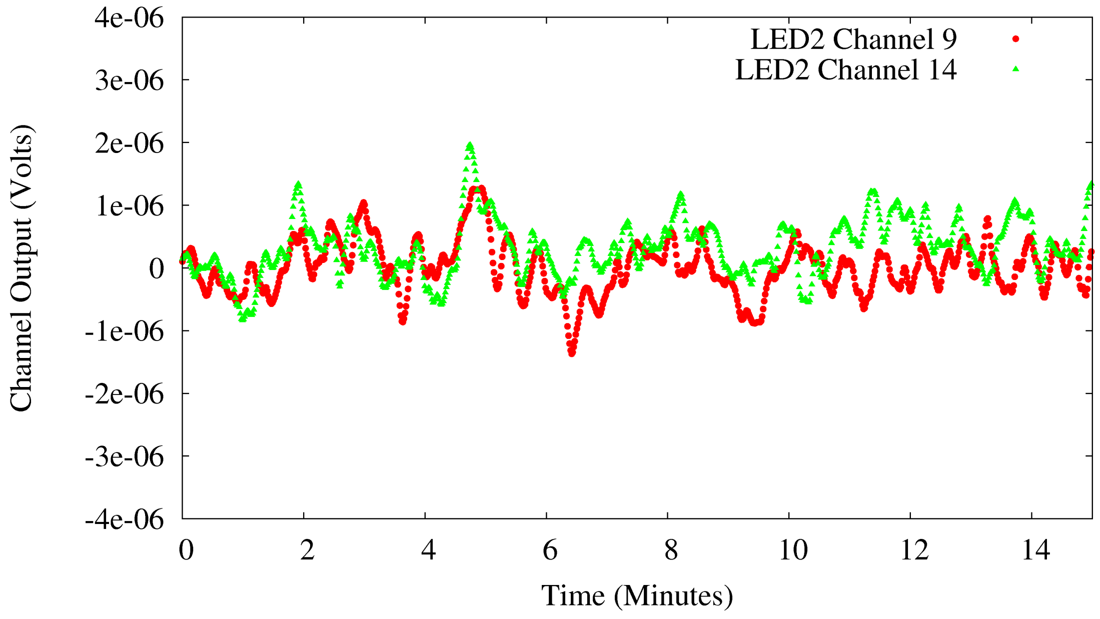


Figure 5-6. Three dimensional plot of Pearson correlation coefficient, r_{AB}, versus channel A and channel B, where A and B include all possible LED1 and LED2 glucose sensor channels. For this plot, R_{Test} is an open circuit, system bandwidth equals 0.0043 Hz, and each channel includes 1000 consecutive samples.



(a) LED1 Channels 1 and 27. $r_{LED1Ch1,LED1Ch27} = 0.425$



(b) LED2 Channels 9 and 14, $r_{LED2Ch9,LED2Ch14} = 0.397$

Figure 5-7. Pairs of DAU channels with highest correlation. (a) LED1 channels 1 and 27 and (b) LED2 channels 9 and 14. Slight correlations over time are likely due to changes in the voltage reference.

illumination, 40.5 dB, matches the experimental SNR of 40.4 dB for 10 kΩ metal film resistors as shown in Fig. 5-4. The noise model SNR prediction for 10 kΩ is 40.8 dB.

A pair of infrared LEDs providing illumination covering the 2.2 to 2.4 micron wavelength range aligned over the variable bandpass filter and 11 kΩ photodiode array provide an apparatus for measuring SNR with the photodiodes illuminated with IR light. The distance between the LEDs and photodiodes determines the photocurrent magnitude - I aligned the LEDs to obtain a lock-in output voltage of 0.1 V, corresponding to the 10 nA glucose sensor photocurrent design goal. The SNR result for the 11 kΩ photodiode array with IR illumination is 37.9 dB as shown in Fig. 5-4, approximately 2.5 dB below the experimental SNR for 10 kΩ R_{Test}.

The mean SNR for the case of the un-illuminated PD array matches the experimental metal film resistor results very well, but SNR falls with illumination. Possible sources of the excess noise include the LED driver circuitry and the LED itself. Measurement of the standard deviation of the LED driver output voltage with the LEDs replaced with a metal film resistor with resistance equivalent to that of the LEDs results in a SNR of 60 dB for a 0.017 Hz measurement bandwidth, nearly 20 dB above the experimental measurements with metal film resistors or photodiode arrays and therefore the LED driver is not likely the source of the excess noise. Studies show LEDs exhibit low frequency noise in their output light intensity[44]. Noise from this source may explain the additional system noise with illuminated photodiodes.

5.6 Conclusions

Due to the offsets present with the ADS1258 configuration the ADS1278 configuration was selected for the remainder of experiments. The signal to noise ratio of the photocurrent monitoring system follows the noise model calculation very well with metal film resistors in place of the photodiodes. The DAU channels provide independent measurements of the photodiode array channels and the measurement system maintains stable operation over a period of days without calibration. Experiments with IR LEDs, the linearly variable bandpass filter, and the 11 kΩ photodiode array show the noise model predicts the experimental SNR without illumination but with illumination the SNR drops by 2 dB. Investigation of the LED driver noise indicates it is not the source of the extra noise but low frequency sources of noise common to LEDs may cause the excess noise.

CHAPTER 6

GLUCOSE CONCENTRATION MEASUREMENT

6.1 Introduction

After a brief discussion of some spectroscopy terms evaluation of the measurement system applied to glucose monitoring begins with determination of the systems lower glucose concentration detection limit. Experiments with chemical solutions of varying glucose content exhibit system noise and elicit the system limit of detection - the minimum glucose concentration the system can detect. Further experiments with several analytes with near IR absorption features show system specificity or the ability to measure glucose concentration in the presence of other chemicals. Final experiments are planned utilizing a lab animal to demonstrate the system with live subjects. Detection of large transients in photocurrent spectra as occur when air passes through the sensor fluid chambers allows identification of these events and prevent the associated faulty glucose concentration predictions.

In spectroscopy the relationship between wavelength of light, λ, and the spectroscopic wavenumber, k, is given by

$$\lambda = \frac{1}{k}. \tag{6.1}$$

In this paper λ is measured in microns and wavenumber is in units of cm^{-1}. The light intensity, I, at one wavelength transmitted by an absorbing medium is

$$I = I_0 10^{-\alpha}, \tag{6.2}$$

where I_0 is the initial light intensity and α is the absorbance. The medium absorbance is found to be

$$\alpha = -\log \frac{I}{I_0}, \tag{6.3}$$

where log is the base 10 logarithm and α is in absorbance units (A.U.). The Beer-Lambert law,

$$\alpha = \varepsilon\, l\, c, \tag{6.4}$$

where ε is the molar absorptivity in 1/(M cm), l is the optical path length through the medium, and c is the molar concentration, states that absorbance is proportional to analyte concentration at one wavelength.

Figure 6-1 illustrates the laboratory apparatus for experiments with chemical solutions. Two manual syringes pull fluid through the glucose sensor sample and reference channels. The reference channel always contains a pH buffer solution.

6.2 Transient Detection

Since water attenuates IR light propagating through the fluid chamber more than air, large photocurrent transients occur when air bubbles pass through the glucose sensor. The resulting fluctuation in photocurrent spectra invalidate the glucose concentration calculation and therefore software to detect and flag large transients in photocurrent samples prevents air bubble transients from causing undue alarm.

Low pass filters reduce the effect of noise spikes so the software must check for large shifts in the input samples before any filtering. The MCU

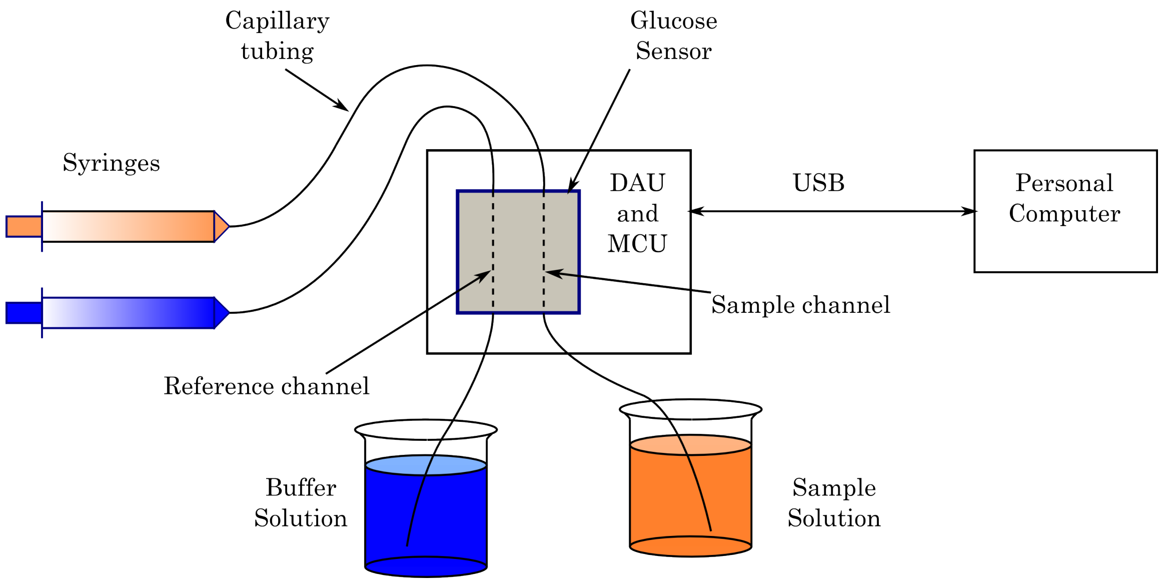
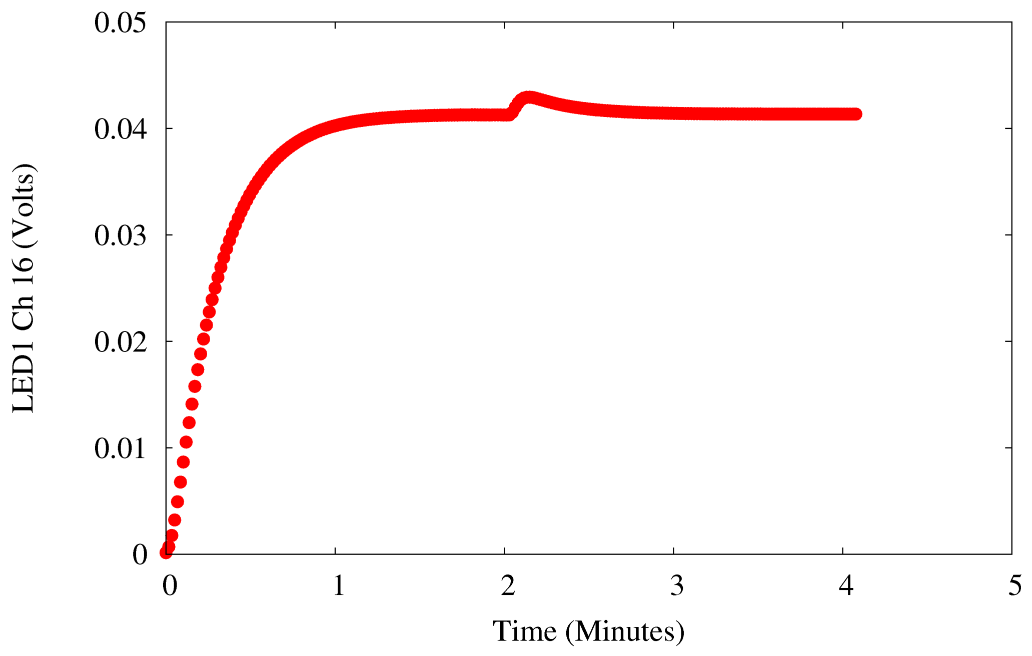


Figure 6-1. Apparatus for concentration measurement experiments.

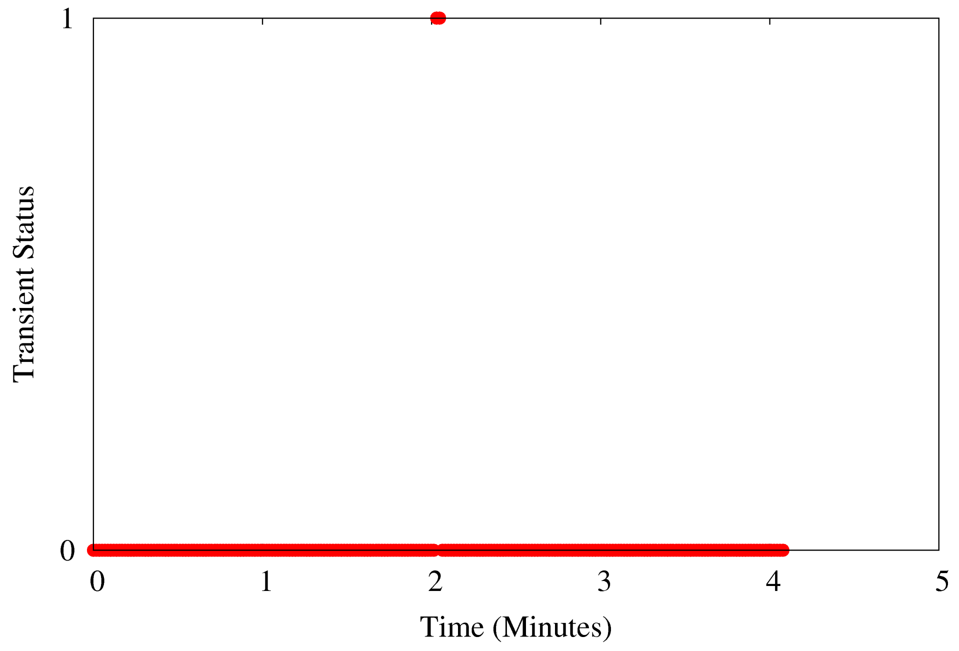
firmware checks whether the magnitude of the difference between the new LED on minus LED off voltage and the previous lock-in result exceeds a threshold set by the user. Figure 6-2 illustrates a demonstration of the transient detection routine with the glucose sensor when a small air bubble passed through the signal channel. The transient monitor status, Fig. 6-2(b), shows the software correctly found the transient just after a time of 2 minutes. For this example the transient threshold was 0.313 Volts and the system bandwidth was 0.017 Hz.

6.3 Baseline Spectrum

The glucose monitoring system measures glucose concentration based on the difference between absorbance spectra from a blank solution flowing in the reference channel and a fluid sample flowing through the sample channel. Since the light intensity reaching a photodiode element is represented by



(a) LED1 Ch 16



(b) Transient Indication

Figure 6-2. Demonstration of transient detection with the glucose sensor. (a) Plot of channel 16 versus time. (b) Status of transient monitor.

$$I = I_0 T\, 10^{-\alpha}, \tag{6.5}$$

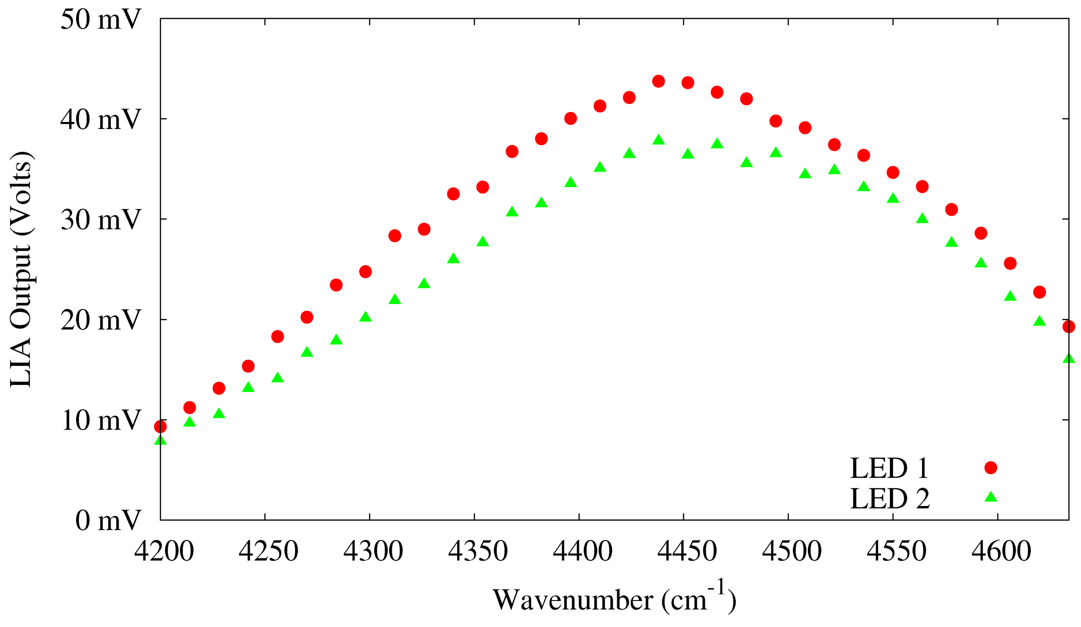
where I is the light intensity reaching the photodiode, I_0 is the initial light intensity, T is the transmission coefficient of the instrument for that photodiode, and α is absorbance, the instrumentation adds a baseline spectrum to the absorbance spectrum measurements. The baseline spectrum includes effects due to channel to channel differences in the electronics and glucose sensor channels. The baseline spectrum is determined by recording reference and sample spectra with blank solutions flowing through both fluid chambers and calculating the absorbance for photodiode n using

$$A_n = -\log_{10}\frac{S_n}{R_n}, \tag{6.6}$$

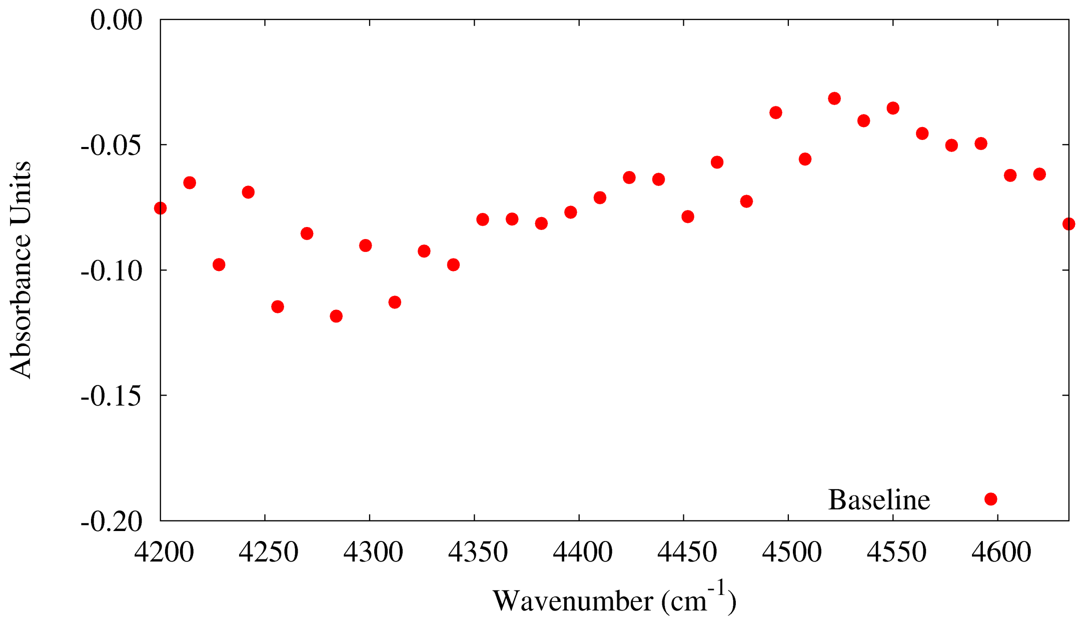
where S_n (R_n) is the nth sample lock-in output voltage. Subtracting the baseline spectrum from subsequent measurements with sample fluid in the sample channel gives the absorbance spectrum for the sample fluid. Plots of the sample and reference spectra with buffer solution in each fluid chamber and the resulting baseline spectrum are shown in Fig. 6-3.

6.4 Absorption Spectrum

To calculate the absorption spectrum of a sample fluid containing analytes one measures the sample spectrum using the procedure for determination of the baseline spectrum in the previous section and subtracts the baseline spectrum from the sample spectrum. The absorbance spectrum plotted in Fig. 6-4(a) shows experimental measurements exhibit glucose absorption bands. Comparison with a reference glucose absorption spectrum recorded with an FTIR spectrometer[35],



(a) Photocurrent spectra for LED1 and LED2.



(b) Baseline spectrum.

Figure 6-3. Example of baseline spectrum calculation. (a) Reference and sample spectra with buffer solution in both fluid chambers. (b) Baseline spectrum calculated from the spectra in (a).

Fig. 6-4(b), indicates spreading of the peaks near 4300 cm^{-1} and 4400 cm^{-1} as well as increased absorption below 4300 cm^{-1}, likely due to a relatively large optical filter bandwidth of the glucose sensor compared to the FTIR instrument. For this example the average of 1000 consecutive buffer spectra formed the baseline spectrum and the average of 1000 50 mM glucose spectra provided the glucose sample spectrum.

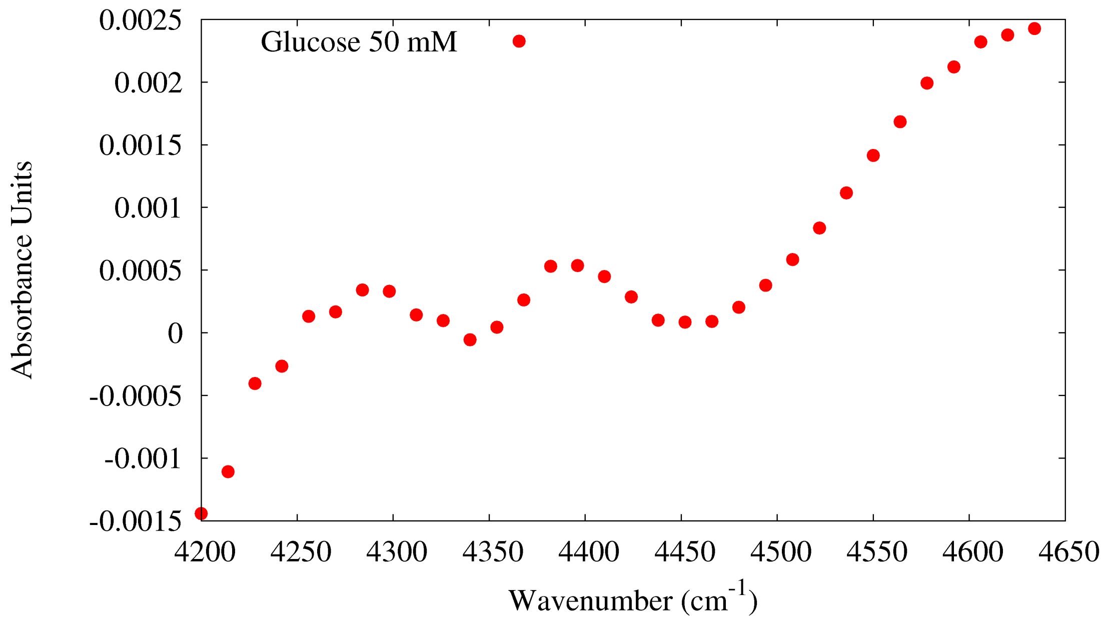
6.5 Limit of Detection

Quantification of the change in the absorption spectrum for several concentrations of glucose allows estimation of the lower limit of glucose concentration detection. The limit of detection,

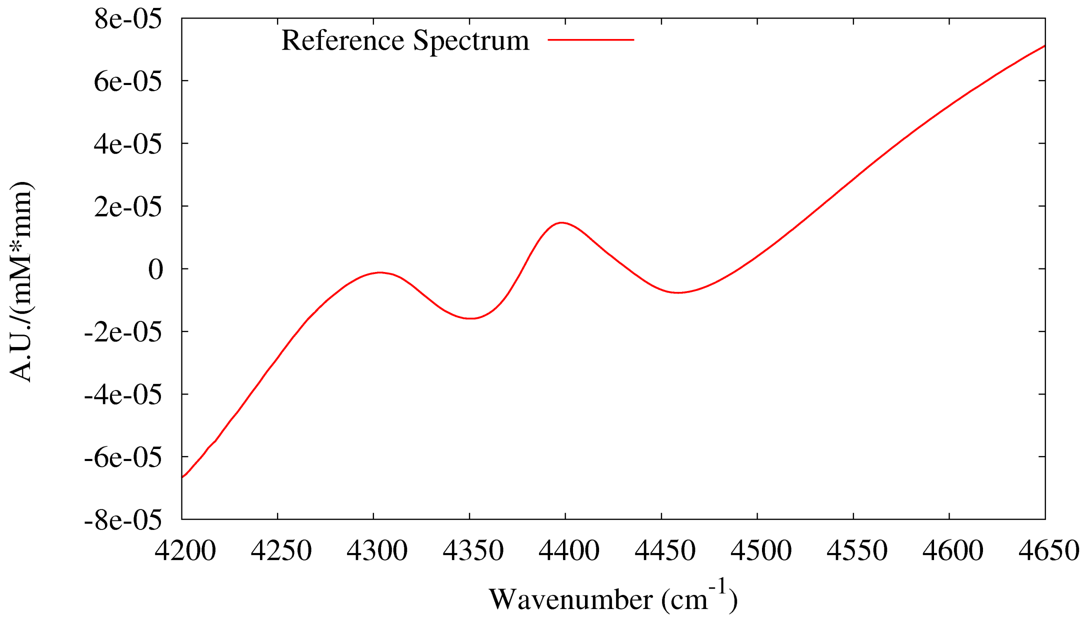
$$LOD = 3\,\sigma_B, \tag{6.7}$$

where σ_B is the standard deviation of glucose concentration for a blank sample, estimates the lower bound of analyte concentration the system determines with a degree of certainty.

We calculated an approximate estimate of the LOD by considering the response from only one glucose sensor channel near the peak of a glucose absorption band. Figure 6-5 shows a plot of the response of channel 16 with wavenumber 4410 cm^{-1} while flowing buffer and the glucose concentrations indicated with a system bandwidth of 0.00108 Hz. The standard deviation of the blank sample absorbance is 27 μA.U. A plot of absorbance vs. glucose concentration, Fig. 6-6, shows there are approximately 75000 mM per A.U. and a rough estimate of the LOD is 6.1 mM for this data set. Performing this analysis



(a)



(b)

Figure 6-4. Glucose absorption spectra. (a) Absorption spectrum of 50 mM glucose sample measured by DAU/MCU with system bandwidth of 0.00108 Hz. (b) Reference glucose absorption spectrum measured with an FTIR instrument[35].

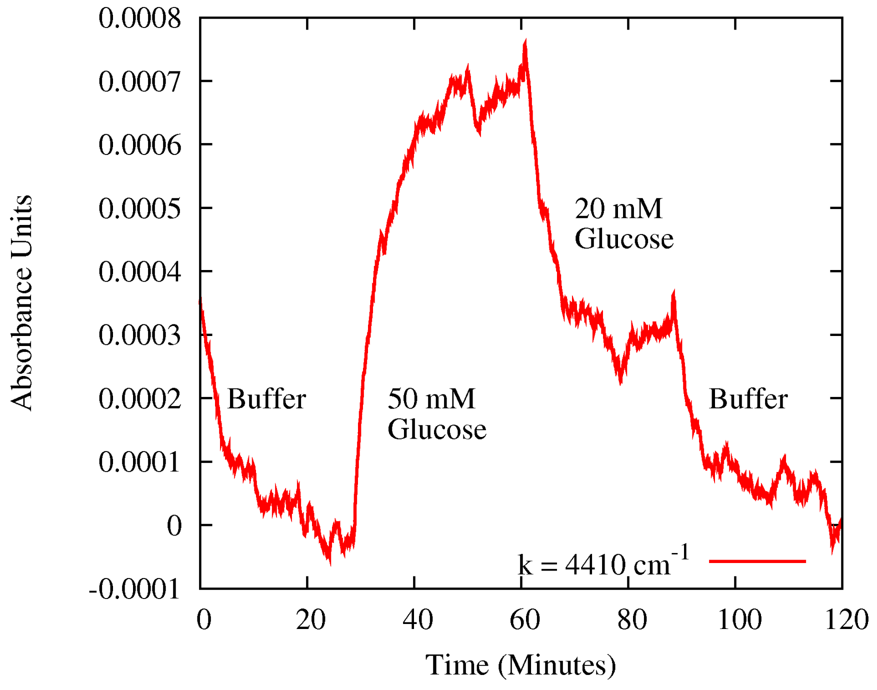


Figure 6-5. Response of glucose sensor channel 16 to several glucose concentrations.

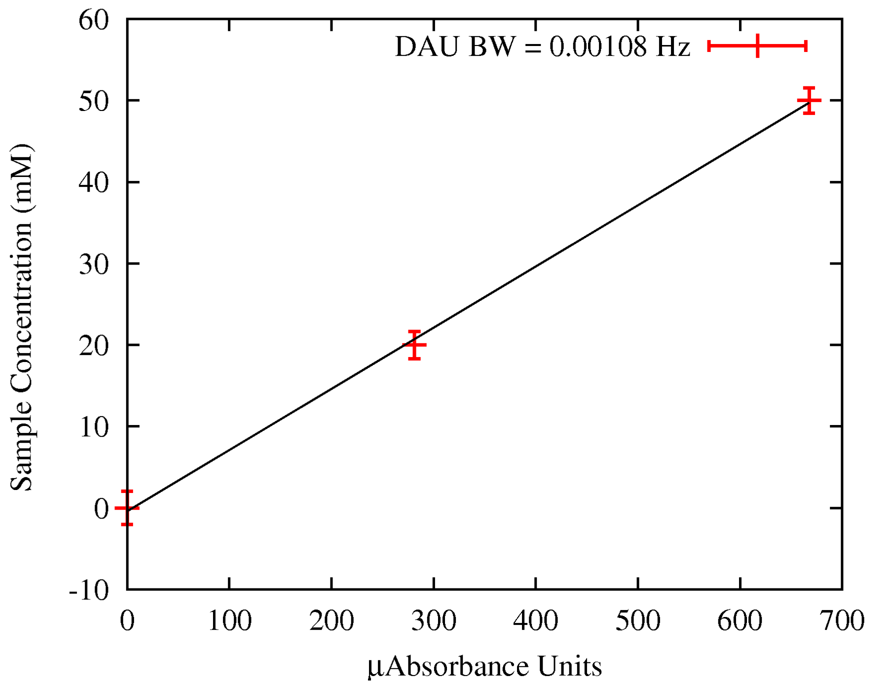


Figure 6-6. Plot of glucose concentration versus absorbance for 20 and 50 mM solutions recorded using the DAU with bandwidth of 0.00108 Hz.

for a range of system bandwidths, Fig. 6-7(a), shows the LOD decreases with decreasing system bandwidth as one would expect since reduction in bandwidth reduces system noise levels and thus reduces the standard deviation of spectral data. Fig. 6-7(b) shows a plot of the signal to noise ratio with a lock-in amplifier output voltage approximately 50 mV using only an IR LED and an 11 kΩ photodiode array without fluid chambers showing the SNR increases as expected for bandwidth extending down to 0.5 mHz. A system bandwidth of 0.5 mHz corresponds to an exponential time constant of approximately 5.3 minutes. Since the glucose concentration changes over a period of perhaps a few tens of minutes and ISF glucose concentration presents physiological delays relative to blood

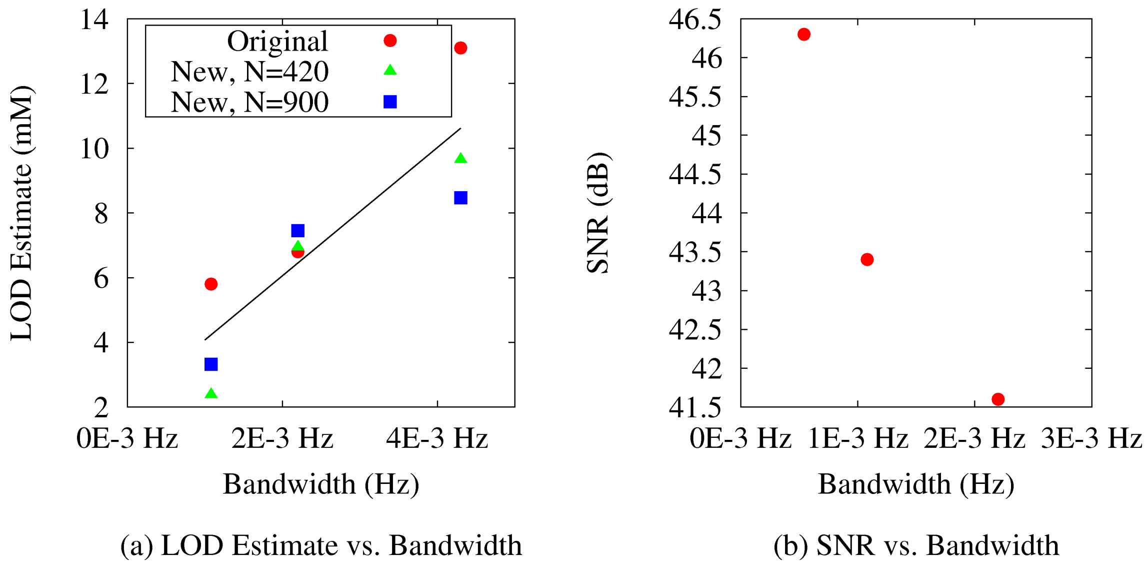


Figure 6-7. LOD with one wavelength and SNR versus bandwidth.

glucose concentration change, low system bandwidths prevent timely system response to glucose concentration and suggests the need for a minimum system bandwidth requirement.

During glucose sensor construction one aligns the LEDs with the PD array assisted by a photocurrent monitoring system[35] consisting of a printed circuit board with several transimpedance amplifiers, a commercially available ADC card, and LED driver demonstration boards. Data I recorded with this system in Fig. 6-8 shows the response of Channel 16 with wavenumber 4410 cm^{-1}, buffer solution in the reference channel, and glucose solution concentration indicated in the sample channel. The transimpedance amplifier board contains OPA27 operational amplifiers in the TIA configuration, Fig. 3-1, with feedback capacitance provided only by stray capacitance. The system recorded samples every 9.5 seconds, thus the system bandwidth must be below 0.053 Hz to prevent aliasing. Analysis of the filtering and downsampling processes utilized by the alignment system shows the filter bandwidth is 0.08 Hz. The standard deviation of absorbance with the blank sample is 148 μA.U. and a plot of solution concentration versus absorbance, Fig. 6-9, exhibits a slope of 76000 mM per A.U. Therefore a rough LOD estimate for this monitoring system with one wavelength is 33.7 mM.

6.6 Minimum System Bandwidth

A minimum system bandwidth which allows nearly all spectral energy of typical glucose concentration transients to pass through the system filter

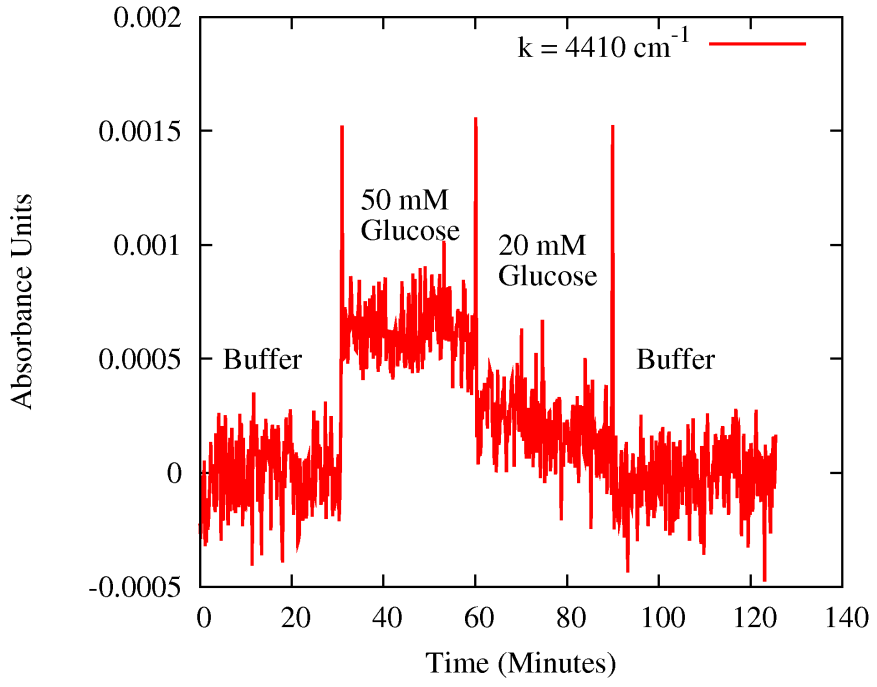


Figure 6-8. Response of one glucose sensor channel measured by monitoring system used for sensor construction to glucose concentrations indicated.

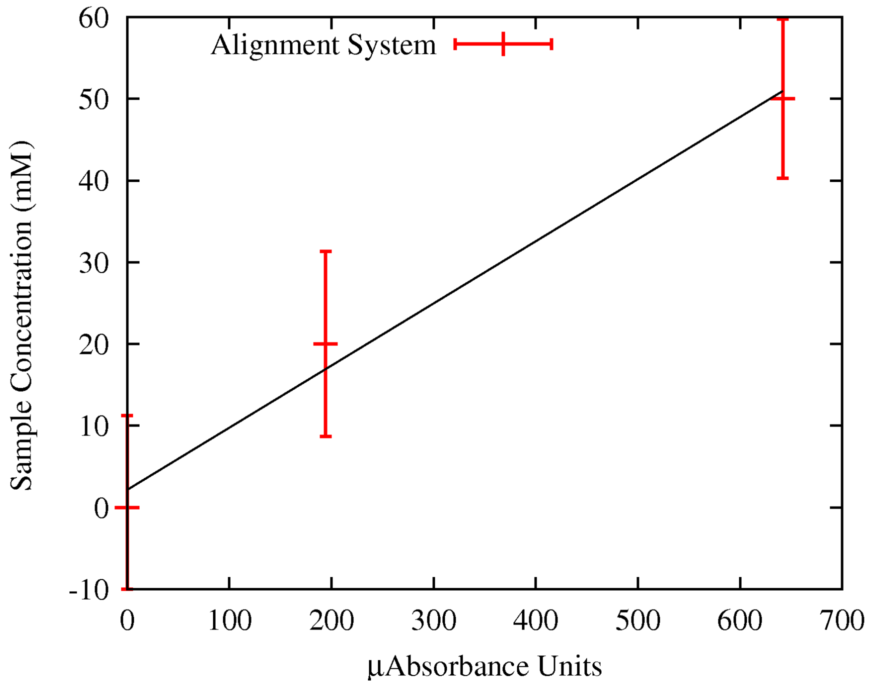


Figure 6-9. Plot of glucose concentration versus absorbance for 20 and 50 mM solutions recorded using the sensor alignment system.

maximizes system SNR while allowing the monitoring system to maintain accuracy as the subjects glucose concentration changes over time. Fourier analysis of a reproduction of a fast meal transient from a patient[45], Fig. 6-10, shows the transient has frequency content up to 0.13 mHz. Figure 6-11(a) includes a plot of the fraction of spectral energy passed by an RC low pass filter versus filter bandwidth, found by integration of the Fourier transform of the transient up to the filter bandwidth. A bandwidth of 1 mHz allows nearly all the transient energy to pass through the filter. The group delay of a filter, given by $-d\phi(\omega)/dt$ where $\phi(\omega)$ is the filter phase, is the amount of time the input signal is delayed as a function of frequency. The group delay of an RC low pass filter,

$$\frac{-d\phi(\omega)}{d\omega}=\frac{RC}{(1+\omega^2R^2C^2)}, \tag{6.8}$$

with bandwidth of 1 mHz, see Fig. 6-11(b), is relatively constant over the transient frequency range of 0.13 mHz, and therefore a bandwidth of 1 mHz allows nearly all transient energy to pass through the filter without distorting the signal and the system will accurately measure the glucose concentration transient.

6.7 Multivariate Calibration

In practice, multivariate calibration methods such as partial least squares (PLS)[46] and the net analyte signal method (NAS)[47,48] use all sensor photodiode channels to predict analyte concentration. The partial least squares method seeks to describe spectral variation in a set of vectors or spectra from

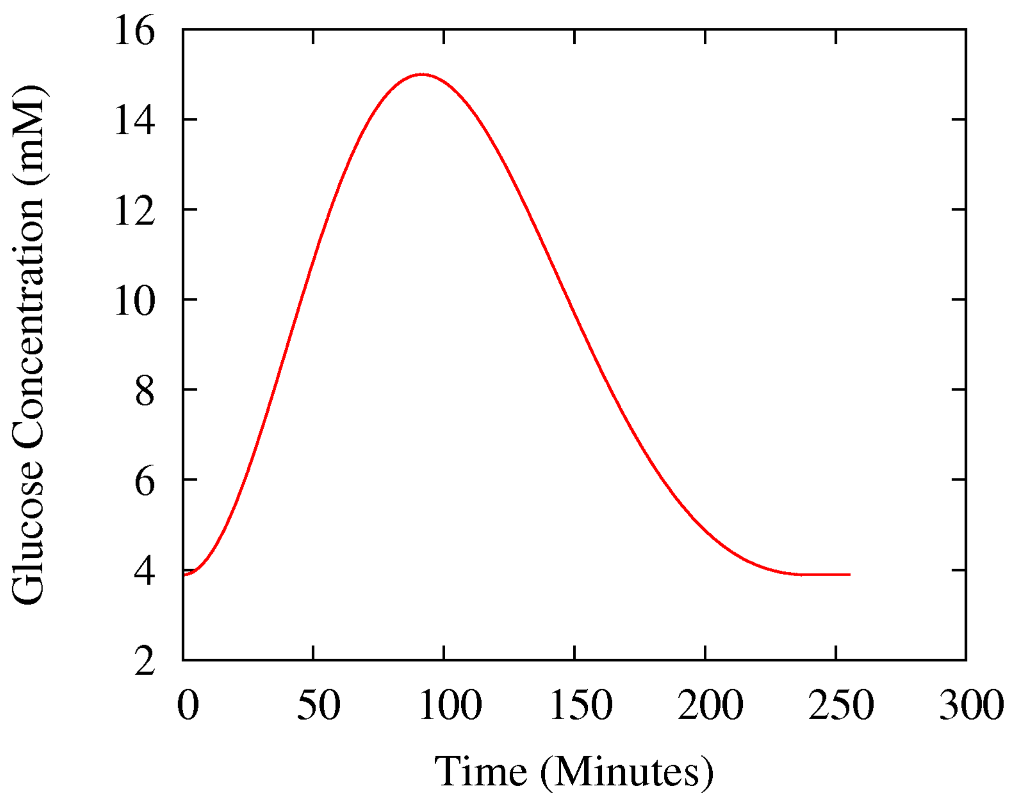
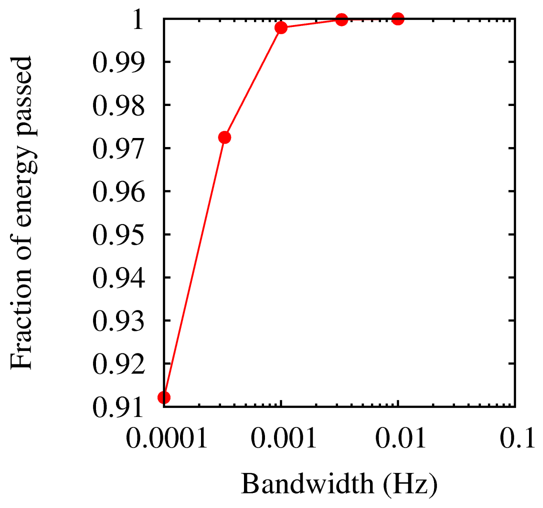
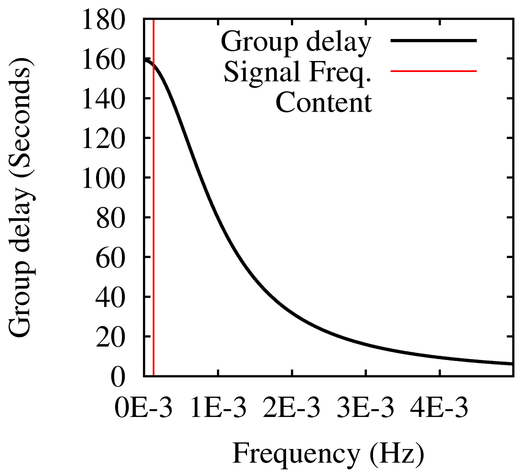


Figure 6-10. A reproduction of a fast meal response glucose concentration transient from patient data.[45]



(a) Freq. content vs. Bandwidth



(b) Group delay vs. Frequency Filter BW = 0.001 Hz

Figure 6-11. Fraction of energy passed by low pass filter vs. bandwidth and group delay for bandwidth of 0.001 Hz.

several samples with a range of analyte concentrations while including concentration information by using correlations between analyte concentration and sample spectra. The process decomposes a spectral matrix X using

$$X = T\,P^T + E\,, \tag{6.9}$$

where X contains sample spectra, T is a matrix of scores, and P^T is the transpose of P, a matrix of spectral loadings. The PLS process determines the first factor through an iterative procedure, removes the corresponding variation in the system, and then finds the next factor.

For one analyte, the procedure begins by calculating a loading weight set

$$w_1 = \frac{X^T\,y}{\|X^T\,y\|}, \tag{6.10}$$

where X is an m by n matrix of m spectra of length n and y is an n by 1 vector of analyte concentrations. The corresponding score is given by

$$t_1 = X\,w_1\,, \tag{6.11}$$

and the spectral loading is

$$p_1 = \frac{X\,t_1}{t_1^T\,t_1}. \tag{6.12}$$

One then uses

$$t_{1,New} = \frac{X\,p_1}{p_1^T\,p_1} \tag{6.13}$$

to calculate a new value for the first score, compares it with the first score estimate, and repeats the procedure until the score stabilizes. The concentration loading is calculated with

$$q_1 = \frac{t_1^T y}{t_1^T t_1}, \tag{6.14}$$

and the system is deflated with

$$X = X - t_1 p_1 \tag{6.15}$$

and

$$y = y - t_1 q_1 . \tag{6.16}$$

One finds additional factors by repeating the above process on the new system. The prediction of sample concentration is given by

$$c = Xb, \tag{6.17}$$

where the calibration vector, b, is given by

$$b = W(P^T W) q, \tag{6.18}$$

where the columns of W are the weights, the columns of P are the spectra loadings, and q is a vector containing the concentration loadings.

To use the NAS method for a particular analyte one first collects a set of background spectra in the absence of the analyte. A method such as singular value decomposition determines several spectral shapes or factors summarizing the variability of the background spectra. The first background factor describes the largest portion of the spectral variation of the background spectra and subsequent factors represent the largest component of the spectral variation not due to previous factors. The net analyte signal for the analyte of interest is the portion of the pure component spectrum orthogonal to the background factors. The net analyte signal for a sample spectrum x is given by

$$A=\left(I-M\,M^{t}\right)x\,, \tag{6.19}$$

where M^{t} is the pseudo-inverse of M, a matrix of background factors. The NAS vector length depends on the concentration of analyte in the pure component solution and must be normalized such that the final NAS for one analyte is

$$NAS=\frac{A}{\|A\|}, \tag{6.20}$$

and the prediction of analyte concentration for a sample spectrum, x, is given by

$$c=NAS\,x. \tag{6.21}$$

Sample spectra were recorded for PLS and NAS calibration models[49]. The PLS calibration method utilized sixty solutions of glucose, urea, and lactate, in random concentrations varying from 4 mM to 50 mM. Forty-eight spectra were used for the calibration model and the remaining 12 spectra formed the prediction spectra. After optimizing the number of factors and the spectral range for the calibration, the standard deviation of errors of prediction and calibration were calculated for 200 different combinations of 48 calibration spectra and 12 prediction spectra. Results of the PLS analysis, Table X, show the mean of all the standard errors of prediction and calibration, MSEP and MSEC, for glucose are 1.02 mM and 1.12 mM, respectively. Also shown in Table X are the MSEP and MSEC results for urea and lactate. A NAS calibration analysis included 22 buffer spectra and 50 mM pure component spectra from glucose, urea, and lactate. The standard error of prediction of glucose with the NAS model is 1.88 mM, see Table X. Euglycemic levels are in the range of 3.9 to 5.5 mM glucose while hyperglycemia occurs at 11 mM, therefore a SEP of 1 to 2 mM acceptably detects hyperglycemia. Low blood glucose, hypoglycemia, is the condition where glucose concentration falls below approximately 3 mM, only a few mM below normal glucose levels, thus the SEP results may not be accurate enough to reliably detect hypoglycemia. Perhaps the larger SEP for the NAS method arises from the use of only 22 background spectra and including a larger number of background spectra in the NAS calibration can reduce the SEP.

Table X. PLS and NAS calibration results.[49]

Analyte	PLS MSEP(mM)	NAS MSEP(mM)
Glucose	1.12	1.88
Urea	0.62	1.69
Lactate	1.7	2.57

Finally, future application of the glucose monitoring system to continuous glucose monitoring with a laboratory animal will exhibit use of the instrument in the actual application. During the experiment the animal will be anesthetized while modulating blood glucose levels and monitoring ISF glucose concentration, similar to the procedure in Ref. 50.

6.8 Summary

The measurement system detects the absorption of glucose with a minimum LOD of 5.8 mM using a rough estimate of LOD with only one wavelength and the glucose sensor alignment apparatus LOD is 33.78 mM for one wavelength. PLS and NAS calibration models exhibit glucose SEP of 1.02 and 1.88 mM, respectively, acceptable for use in detection of hyperglycemia but not quite acceptable for use monitoring hypoglycemia. The system has a transient detection feature for flagging large fluctuations due to the presence of air in the sensor fluid chambers. Experiments with a live laboratory animal will exhibit system performance in the continuous glucose monitoring application.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

The measurement system provides low noise near IR absorption spectra with single wavelength LOD estimate of 5.8 mM, using near IR absorption spectrosopy, a direct glucose concentration measurement. Experiments performed by Joo-Young Choi utilizing PLS and NAS calibration models show the system exhibits glucose SEP of 1.02 and 1.88 mM, showing the system is acceptable for use in detection of hyperglycemia but not quite acceptable for use monitoring hypoglycemia. Utilization of direct glucose measurement methods ensures the measurement system responds to actual changes in glucose concentration instead of chance correlations with other characteristics such as refractive index.

The glucose sensor forms a spectrometer sensitive to a range of wavelengths near peaks in the glucose absorption spectrum. The monitoring system, comprised of a data acquisition unit and main controller unit, samples photocurrent from the glucose sensor and constructs photocurrent spectra for the calibration process. The measurement system noise properties closely follow numerical models predicting system performance. Investigation of characteristics such as drift, correlation, transient detection, and LOD proved system ability to measure glucose concentration using near IR spectroscopy.

If desired, further improvements to DAU electronics may bring improvement in LOD including use of a low noise current amplifier to amplify photocurrent. A current amplifier before the transimpedance amplifier will lift the SNR and help improve LOD. Future work also includes miniaturization of the MCU and DAU into a credit card sized unit utilizing ultra-low power components and design methods. One design suggestion for the miniaturized system is the use of a 20 bit ADC - this reduces required die area and power dissipation without impact to the noise model. Reduction in the number of samples recorded per unit time allows use of lower switching frequency microcontrollers and therefore reduction in power consumption. Analog switches multiplexing the photodiodes with a limited number of transimpedance amplifiers and ADCs also reduces system power requirements.

REFERENCES

1. Diabetes Fact Sheet No. 312. World Health Organization: Geneva, Switzerland, www.who.int (accessed 3/21/2012).

2. Cengiz, E.; Tamborlane, W. V. A Tale of Two Compartments: Interstitial Versus Blood Glucose Monitoring. *Diabetes Tech. Ther*. 11, S-11, 2009.

3. Arnold, M. A.; Small, G. W. Noninvasive Glucose Sensing. *Anal. Chem*. **77**, pp. 5429-5439, 2005.

4. Bai, C.; Graham, T. L.; Arnold, M. A. Assessing and Advancing Technology for the Noninvasive Measurement of Clinical Glucose. *Anal. Chem*. **41**, pp. 2773-2793, 2008.

5. Chia, C. W.; Saudek, C. D. Glucose sensors: toward closed loop insulin delivery. *Endocrinol Metab Clin N Am* **33**, pp. 175-195, 2004.

6. Olesberg, J. T.; Cao, C.; Yager, J. R.; Prineas, J. P.; Coretsopoulos, C.; Arnold, M. A.; Olafsen, L. J.; Santilli, M. Optical Microsensor for Continuous Glucose Measurements in Interstitial Fluid. *Proc. Of SPIE* **6094**, 609403, 2006.

7. Wu, P.; He, Y.; Wang, H. F.; Yan, X. P. Conjugation of Glucose Oxidase onto Mn-Doped ZnS Quantum Dots for Phosphorescent Sensing of Glucose in Biological Fluids. *Anal. Chem*. **82**, pp. 1427-1433, 2010.

8. Billingsley, K.; Balaconis, M. K.; Dubach, J. M.; Zhang, N.; Lim, E.; Francis, K. P.; Clark, H. A. Flourescent Nano-Optodes for Glucose Detection. *Anal. Chem.* **82**, pp. 3707-3713, 2010.

9. Yan, Q.; Peng, B.; Su, G.; Cohan, B. E.; Major, T. C.; Meyerhoff, M. E. Measurement of Tear Glucose Levels with Amperometric Glucose Biosensor/Calillary Tube Configuration. *Anal. Chem.* **83**, pp. 8341-8346.

10. Sekretaryova, A. N.; Vokhmyanina, D. V.; Chulanova, T. O.; Karyakina, E. E.; Karyakin, A. A. Reagentless Biosensor Based on Glucose Oxidase Wired by the Mediator Freely Diffusing in Enzyme Containing Membrane. *Anal. Chem.* **84**, pp. 1220-1223, 2012.

11. Tsai, T. W.; Heckert, G.; Neves, L. F.; Tan, Y.; Kao, D. Y.; Harrison, R. R.; Resasco, D. E.; Schmidtke, D. W. Adsorption of Glucose Oxidase onto Single-Walled Carbon Nanotubes and Its Application in Layer-By-Layer Biosensors. *Anal. Chem*. **81**, pp. 7917-7925, 2009.

12. Ke Ma, K.; Yuen, J. M.; Shah, N. C.; Walsh, J. T., Jr.; Glucksberg, M. R.; Van Duyne, R. P. Spatially Offset Raman Spectroscopy: Multiple Rats, Improved Hypoglycemic Accuracy, Low Incident Power, and Continuous Monitoring for Greater than 17 Days. *Anal. Chem*. **83**, pp. 9146-9152, 2011.

13. Yuen, J. M.; Shah, N. C.; Walsh, J. T., Jr.; Glucksberg, M. R.; Van Duyne, R. P. Transcutaneous Glucose Sensing by Surface-Enhanced Spatially Offset Raman Spectroscopy in a Rat Model. *Anal. Chem*. **82**, pp. 8382-8385, 2010.

14. Ward Muscatello, M. M.; Stunja, L. E.; Asher, S. A. Polymerized Crystalline Colloidal Array Sensing of High Glucose Concentrations. *Anal. Chem* Vol. **81**, pp. 4978-4986, 2009.

15. Olesberg, J. T.; Arnold, M. A.; Mermelstein, C.; Schmitz, J.; Wagner, J. Tunable Laser Diode System for Noninvasive Blood Glucose Measurements. *Appl. Spectros*. **59**, pp 1480-1484, 2005.

16. Shen, Y. C.; Davies, A. G.; Linfield, E. H.; Elsey, T. S.; Taday, P. F.; Arnone, D. D. The use of Fourier-transform infrared spectroscopy for the quantitative determination of glucose concentration in whole blood. *Phys. Med. Biol*. **48**, pp. 2023-2032, 2003.

17. Abookasis, D.; Workman, J. J. Direct measurements of blood glucose concentration in the presence of saccharide interferences using slope and bias orthogonal signal correction and Fourier transform near-infrared spectroscopy. *J. Biomed. Opt*. **16**, 027001, 2011.

18. Heise, H. M.; Damm, U.; Bodenlenz, M.; Kondepati, V. R.; Kohler, G.; Ellmerer, M. Bedside monitoring of subcutaneous interstitial glucose in healthy individuals using microdialysis and infrared spectrometry. *J. Biomed. Opt*. **12**, 024004, 2007.

19. Amin-Akhlaghi, Z.; Cooley, D. W.; Andersen, D. R. Study of an Infrared Glucose Sensor and its Noise Model. FFH2011, Vienna, Austria, 2011.

20. Cooley, D. W.; Andersen, D. R. Low Noise Measurement of Photocurrent in Low Impedance Photodiodes. EIT2010, Normal, Illinois, 2010.

21. Cooley, D. W.; Andersen, D. R. Low Noise Measurement of Photocurrent for Continuous Glucose Monitoring. BIODEVICES 2010, Valencia, Spain, 2010.

22. Olesberg, J. T.; Arnold, M. A; Hu, S. Y. B. Temperature-Insensitive Near-Infrared Method for Determination of Protein Concentration during Protein Crystal Growth. *Anal. Chem*. **72**, pp. 4985-4990, 2000.

23. Martens, H.; Naes, T. *Multivariate Calibration;* Wiley: New York, 1989.

24. Kanukurthy, K. S. Wireless controller for a near infrared multichannel optical glucose sensor. Ph.D. Dissertation, University of Iowa, Iowa City, IA, 2007.

25. Hobbs, P. C. C. *Building Electro-Optical Systems*, *2nd Ed.*; Wiley: New York, 2009.

26. Graeme, J. *Photodiode Amplifiers:* McGraw Hill: New York, 1996.

27. Milnes, A. G.; Polyakov, A. Y. Gallium antimonide device related properties. *Solid State Electron*. **36,** pp. 803-818, 1993.

28. Sze, S. M. *Physics of Semiconductor Devices;* Wiley: New York, 1981.

29. Dicke, R. H. The Measurement of Thermal Radiation at Microwave Frequencies. *Rev. Sci. Instrum*. **17**, pp. 268-275, 1946.

30. App. Note 3. Stanford Research Systems: Sunnyvale, California, www.thinksrs.com (accessed 3/21/2012).

31. Dorrington, A. A.; Kunnemeyer, R. A simple microcontroller based digital lock-in amplifier for the detection of low level optical signals. DELTA'02 0-7695-1453-7/02, 2002.

32. Prineas, J. P.; Olesberg, J. T.; Yager, J. R.; Cao. C.; Coretsopoulos, C.; Reddy, M. H. M. Cascaded active regions in 2.4 μm GaInAsSb light-emitting diodes for improved current efficiency. *Appl. Phys. Lett*. **89**, 211108, 2006.

33. Pierret, R. F. *Semiconductor Device Fundamentals;* Addison-Wesley: Reading, Massachusetts, 1996.

34. S1133 Datasheet. Hamamatsu Photonics, K. K.: Hamamatsu City, Japan, www.hamamatsu.com (accessed 3/21/2012).

35. Private communication with Jon Olesberg, March 2012.

36. Saleh, B. E. A.; Teich, M. C. *Fundamentals of Photonics;* Wiley: New York, 1991.

37. Motchenbacher, C. D.; Connelly, J. A. *Low-Noise Electronic System Design;* Wiley: New York, 1993.

38. Antoniou, A., *Digital Filters: Analysis, Design, and Applications*; McGraw Hill: New York, 1993.

39. App. Note AN-283. Analog Devices, Inc.: Norwood, Massachusetts, www.analogdevices.com (accessed 3/21/2012).

40. App. Report SBOA066A. Texas Instruments, Inc.: Dallas, Texas, www.ti.com (accessed 3/21/2012).

41. ADS1258 Datasheet. Texas Instruments, Inc.: Dallas, Texas, www.ti.com (accessed 3/21/2012).

42. *MPLAB IDE V8.10;* Microchip Technology, Inc.: Chandler, Arizona, www.microchip.com (accessed 3/21/2012).

43. Everitt, B. S.; Skrondal, A. *Cambridge Dictionary of Statistics, 4th Ed.*; Cambridge University Press: Cambridge, United Kingdom, 2006.

44. Rumyantsev, S. L.; Shur, M. S.; Bilenko, Y.; Kosterin, P. V.; Salzberg, B. M. Low frequency noise and long-term stability of noncoherent light sources. *J. Appl. Phys*. **96**, pp. 966-969, 2004.

45. Keenan, D. B.; Mastrototaro, J. J.; Voskanyan, G.; Steil, G. M. Delays in Minimally Invasive Continuous Glucose Monitoring Devices: A Review of Current Technology. *J. Diabetes Sci. Tech*. **3**, pp. 1207-1214, 2009.

46. Bai, C. Noninvasive Near Infrared Spectroscopy on Living Tissue with Multivariate Calibration Approaches. Ph.D. Dissertation, University of Iowa, Iowa City, IA, 2010.

47. Lorber, A.; Faber, K.; Kowalski, B. R. Net Analyte Signal Calculation in Multivariate Calibration. *Anal. Chem*. **69**, pp. 1620-1626, 1997.

48. Private communication with Gary W. Small, April 2012.

49. Private communication with Joo-Young Choi, March 2012.

50. Olesberg, J. T.; Liu, L.; Van Zee, V.; Arnold, M. A. In Vivo Near-Infrared Spectroscopy of Rat Skin Tissue with Varying Blood Glucose Levels. *Anal. Chem*. **78**, pp. 215-223, 2006.

APPENDIX A

MOVING AVERAGE FILTER BANDWIDTH

The frequency response of linear, time invariant system is

$$H(\omega)=\sum_{m=-\infty}^{\infty} h(m)\mathrm{e}^{-j\omega m}. \tag{A.1}$$

The N-sample moving average is represented by

$$h(n)=1/N\,,\ for\ n=0,1,2,\ldots.N-1. \tag{A.2}$$

The frequency response of $h(n)$ is then

$$H(\omega)=\frac{1}{N}\sum_{m=0}^{N-1}\mathrm{e}^{-j\omega m}=\frac{1}{N}\left(\frac{1-\mathrm{e}^{-j\omega N}}{1-\mathrm{e}^{-j\omega}}\right), \tag{A.3}$$

since

$$\sum_{m=N}^{M} a^m=\frac{a^N-a^{M+1}}{1-a}. \tag{A.4}$$

APPENDIX B

NOISE MODEL CALCULATION SOFTWARE

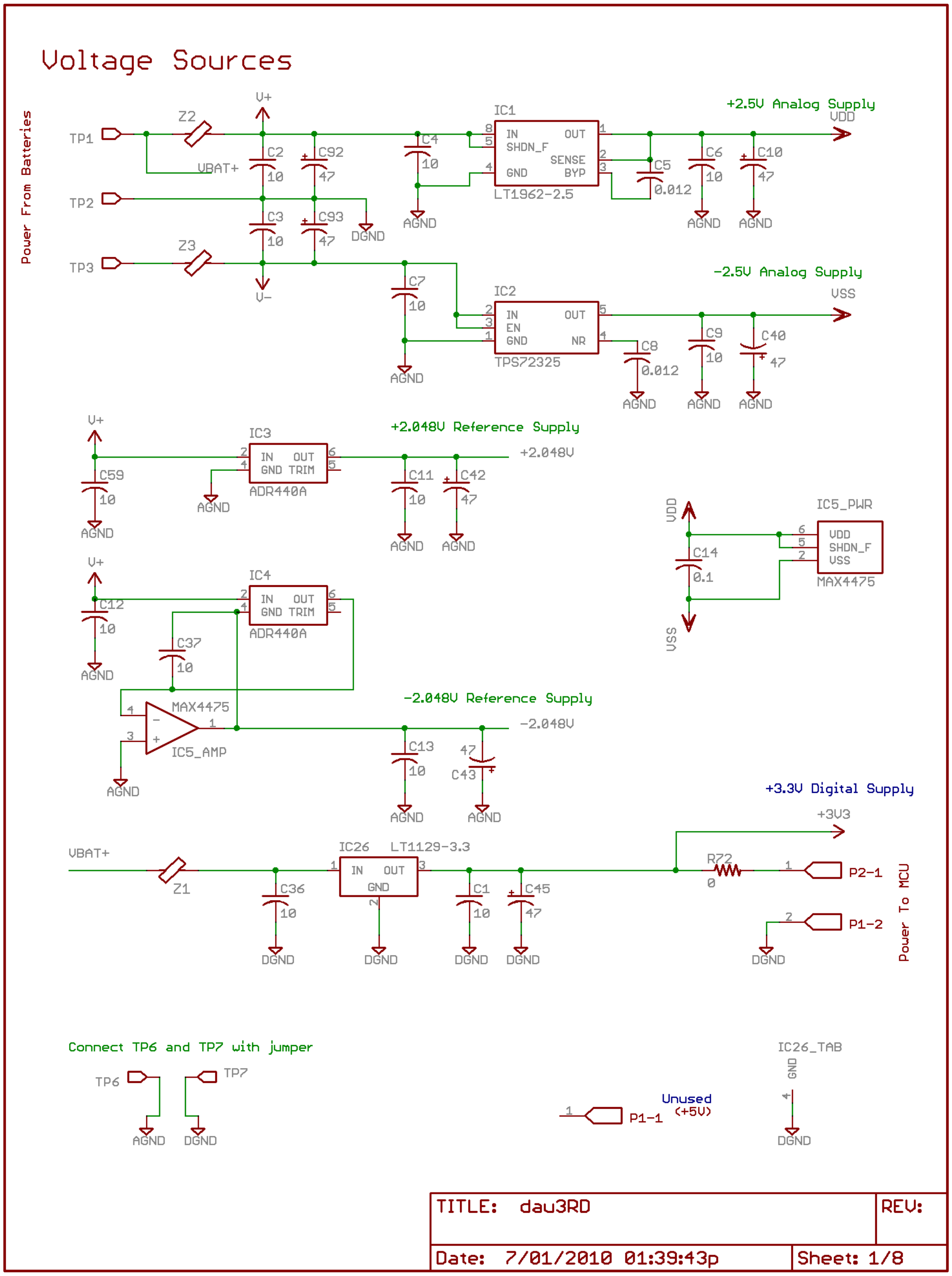
Filename: noise.c.

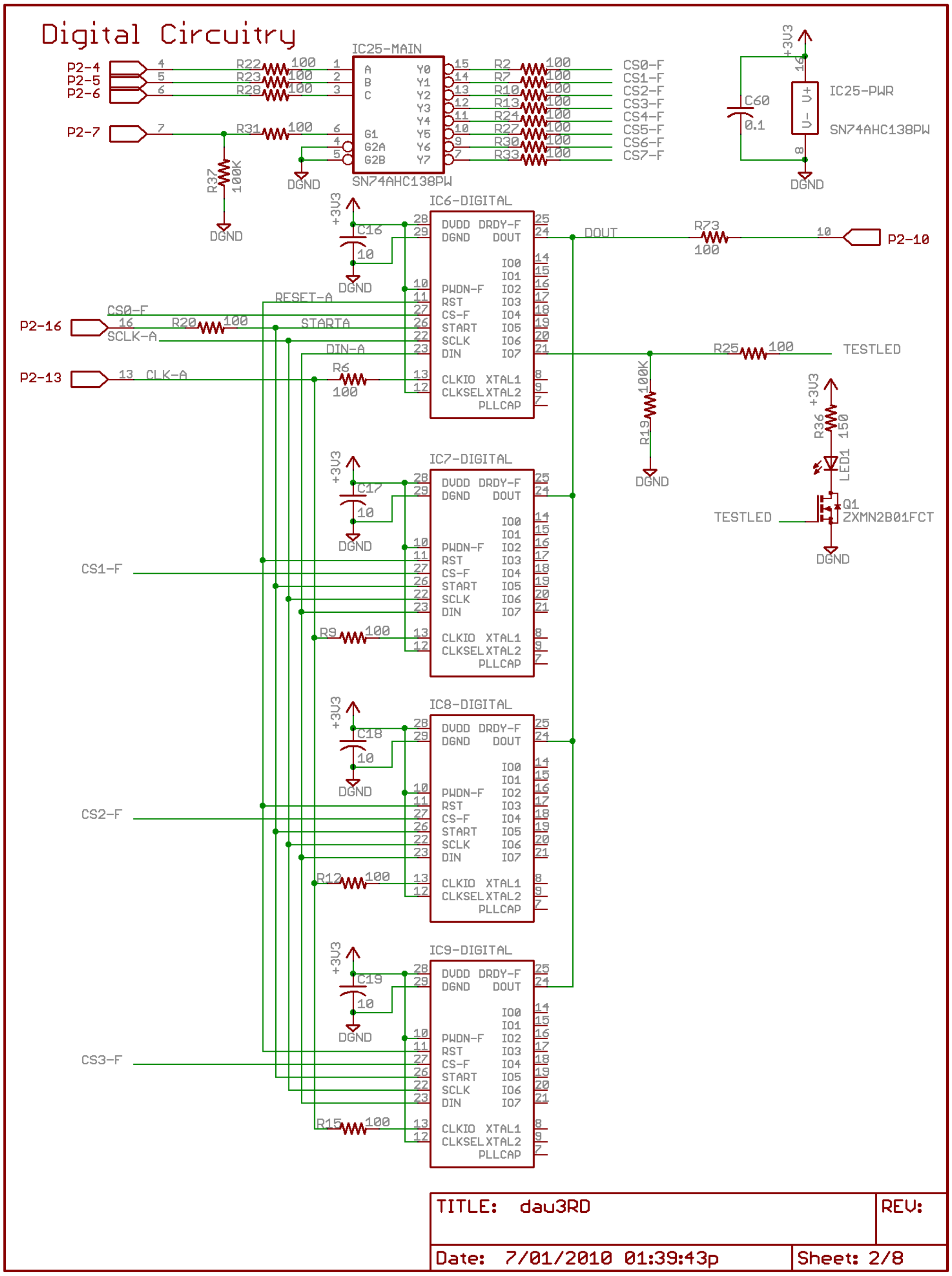
```
#include <stdio.h>
#include <math.h>

int main()
{

/* March 4 2012 */
/* Noise model calculation for Lock-In Amplifier */
/* BW in input file is single ended BW, ENB = 2 times BW times pi/2 */
/* Eni is value of Eni from datasheet at the ref. frequency 39.1 Hz */
/* Assume ADC noise is distributed over the ADC passband */
/* ADC noise is ADC noise times ENB / (2 * ADC BW) */
/* deltav is ADC noise in microVolts RMS */
/* Uses broadband noise for Vref */
/* Assumes shot noise, 2eI, is for double sided spectrum */

  int i, j;
  double eni, eni2, ibn, ers, ibi, rf, rpd, temp, avenum, isig, gain, bw, bw2;
  double vn, snr, snrdb, deltav;
  FILE *idfPtr; /* idfPtr = input.dat file pointer */
  FILE *odfPtr; /* odfPtr = output.dat file pointer */

  ibn = 0.0; /* noninverting input current noise */
  ers = 0.0; /* noninverting source resistor voltage noise */
  ibi = 0.5e-15; /* inverting input current noise density */

  if ( ( idfPtr = fopen( "input.dat", "r" ) ) == NULL )
    printf( "No input file!\n" );
  else {
    printf( "Reading input data\n" );
    fscanf( idfPtr, "%lf%lf%lf%lf%lf%lf%lf", &rf, &temp, &avenum, &isig, &bw,
&deltav, &eni);
    printf( "%e %e %e %e %e %e %e\n", rf, temp, avenum, isig, bw, deltav, eni);
    fclose( idfPtr );
    printf( "Got the input data okay\n" );
  }
```

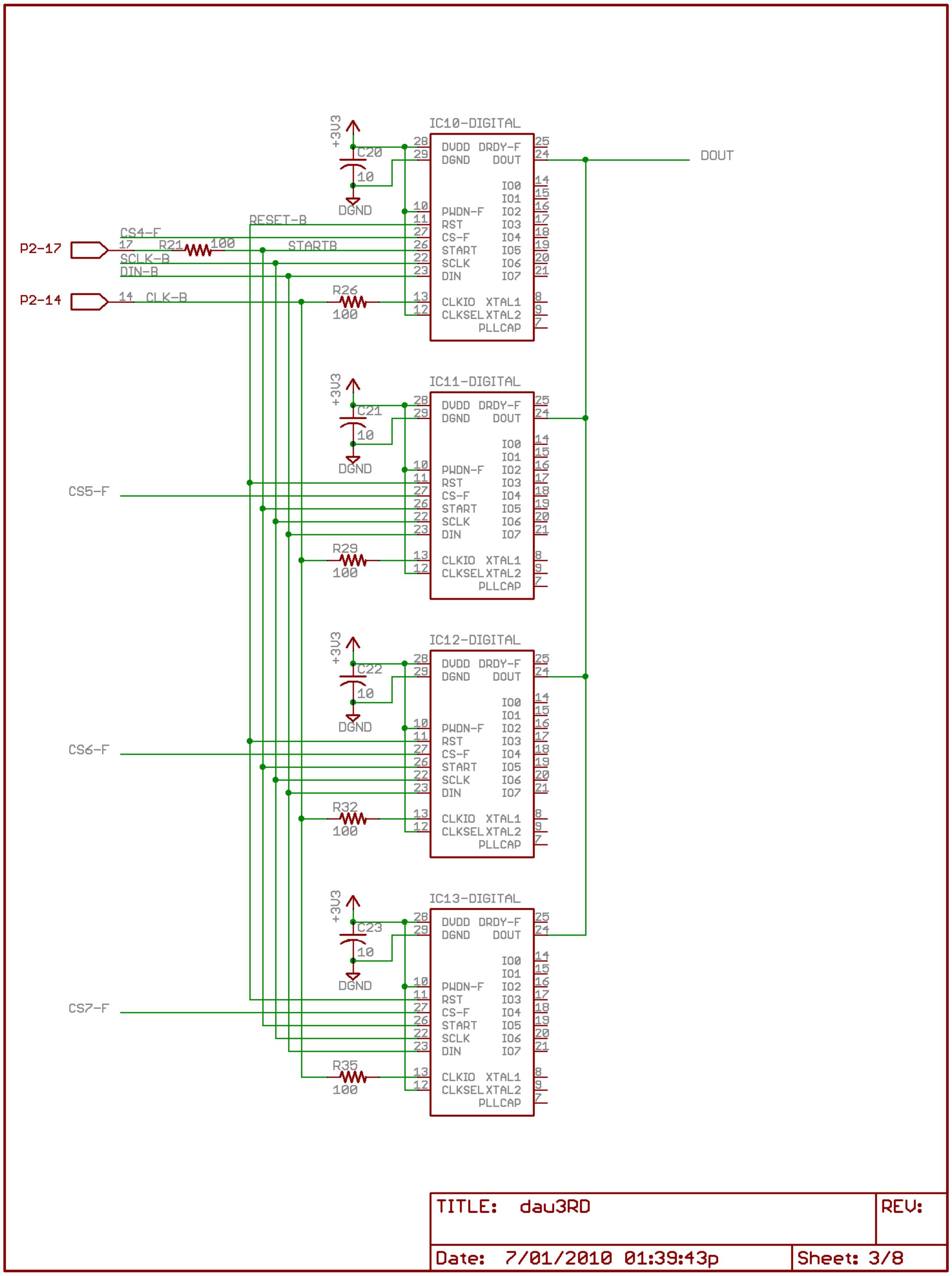
```
 odfPtr = fopen( "shotpc.dat", "w" );
 fprintf( odfPtr, "RF=%e Temp=%.1f Samp Ave=%.1f Isig=%e BW=%.6f deltaV=
%e Eni=%e\n", \
        rf, temp, avenum, isig, bw, deltav, eni );

 bw2 = 2.0 * bw * 1.57;
 j = 2;
 while ( j <= 8 ) {
  i = 1;
  while( i <= 9 ) {
    rpd = ( double ) i * pow( 10.0, ( double ) j);
    gain = 1.0 + rf / rpd;
    vn = sqrt( 0.0 * eni * eni * gain * gain * bw2 \
        + 0.0 * ( ibi * rf ) * ( ibi * rf ) * bw2 \
        + 0.0 * 4.0 * 1.38e-23 * temp * rf * gain * bw2 \
        + 0.0 * deltav * deltav *bw2*bw2/(2*390.6*2*390.6) \
          + 0.0 * 45e-9 * 45e-9 * bw2 \
          + 0.0 * 1.0 * 1.6E-19 * 1E-12 * bw2 * rf * rf \
          + 0.0 * 1.0 * 1.6E-19 * (70.0E-6 / rpd) * bw2 * rf * rf \
          + 1.0 * 1.0 * 1.6e-19 * 10.0e-9 * bw2 * rf * rf);

    vn = vn/sqrt((double)avenum);
    snr = isig * rf / vn;
    snrdb = 10.0 * log10( snr ) - 3.01;
        /* Subtract 3.01 dB since LIA output is multiplied by 2 */
        /* in order to calculate peak to peak Ipc*Rf           */
    fprintf (odfPtr, "%e %e\n", rpd, snrdb );
    i += 1;
  }
  j += 1;
 }

 fclose( odfPtr );
 return 0;
}
```

APPENDIX C

ADS1258 DAU SCHEMATIC DIAGRAM


Voltage Sources
Power From Batteries
TP1
TP2
TP3
Z2
Z3
VBAT+
V+
V-
C2 10
C92 47
C3 10
C93 47
DGND
C4 10
AGND
IC1
IN OUT
SHDN_F SENSE
GND BYP
LT1962-2.5
C5 0.012
C6 10
C10 47
+2.5V Analog Supply
VDD
C7 10
IC2
IN OUT
EN
GND NR
TPS72325
C8 0.012
C9 10
C40 47
-2.5V Analog Supply
VSS
IC3
IN OUT
GND TRIM
ADR440A
C59 10
C11 10
C42 47
+2.048V Reference Supply
+2.048V
IC5_PWR
VDD
SHDN_F
VSS
MAX4475
C14 0.1
IC4
ADR440A
C12 10
C37 10
MAX4475
IC5_AMP
-2.048V Reference Supply
-2.048V
C13 10
C43 47
+3.3V Digital Supply
+3V3
VBAT+
Z1
IC26 LT1129-3.3
IN OUT
GND
C36 10
C1 10
C45 47
R72 0
P2-1
P1-2
Power To MCU
Connect TP6 and TP7 with jumper
TP6
TP7
Unused (+5V)
P1-1
IC26_TAB
GND
TITLE: dau3RD
REV:
Date: 7/01/2010 01:39:43p
Sheet: 1/8


Digital Circuitry
IC25-MAIN
SN74AHC138PW
IC25-PWR
P2-4
P2-5
P2-6
P2-7
P2-10
P2-13
P2-16
CS0-F
CS1-F
CS2-F
CS3-F
CS4-F
CS5-F
CS6-F
CS7-F
IC6-DIGITAL
IC7-DIGITAL
IC8-DIGITAL
IC9-DIGITAL
DOUT
TESTLED
RESET-A
STARTA
DIN-A
SCLK-A
CLK-A
R73
R25
R19
R36
LED1
Q1
ZXMN2B01FCT
+3U3
DGND
TITLE: dau3RD
REV:
Date: 7/01/2010 01:39:43p
Sheet: 2/8

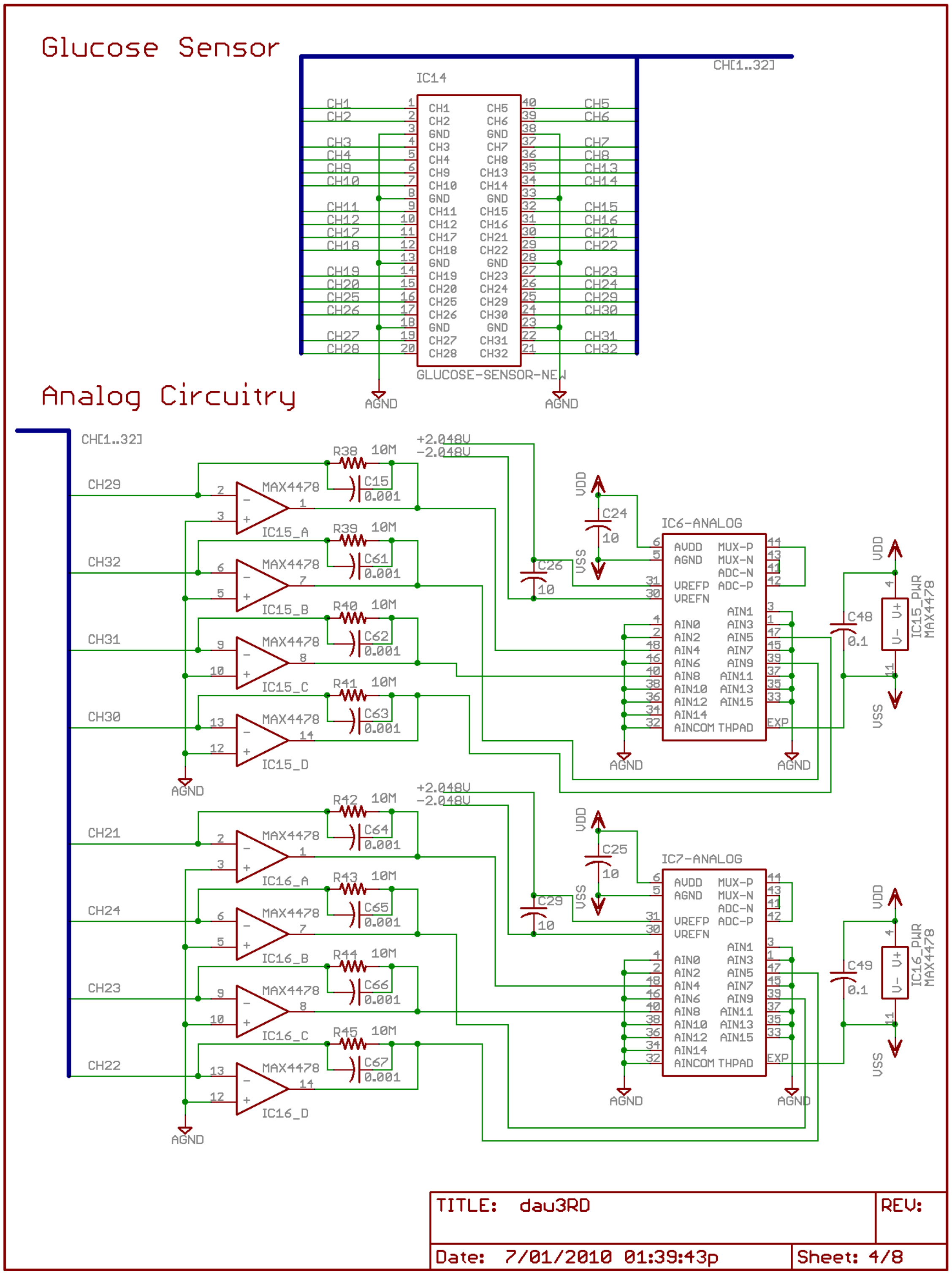


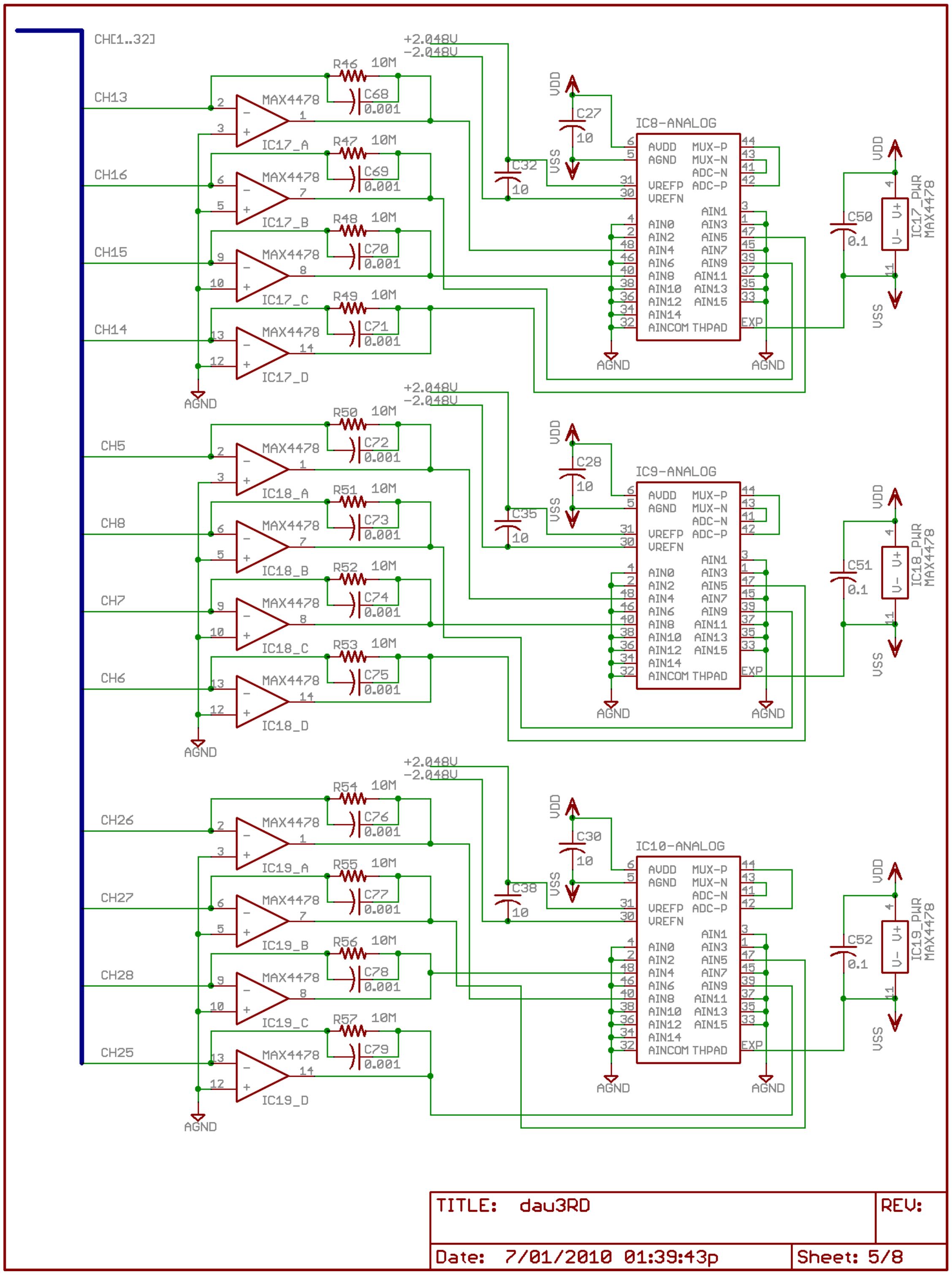
TITLE: dau3RD

REV:

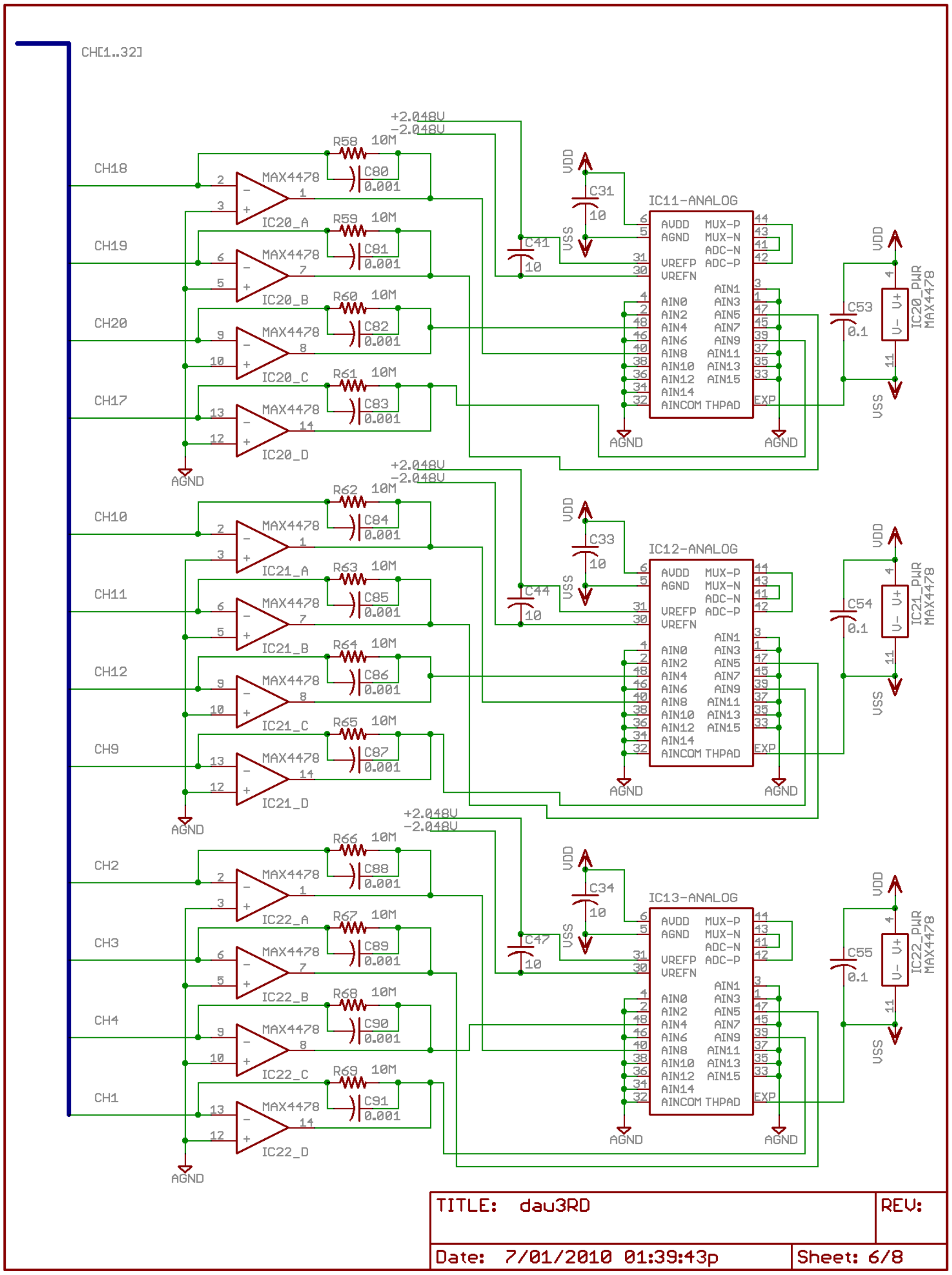
Date: 7/01/2010 01:39:43p

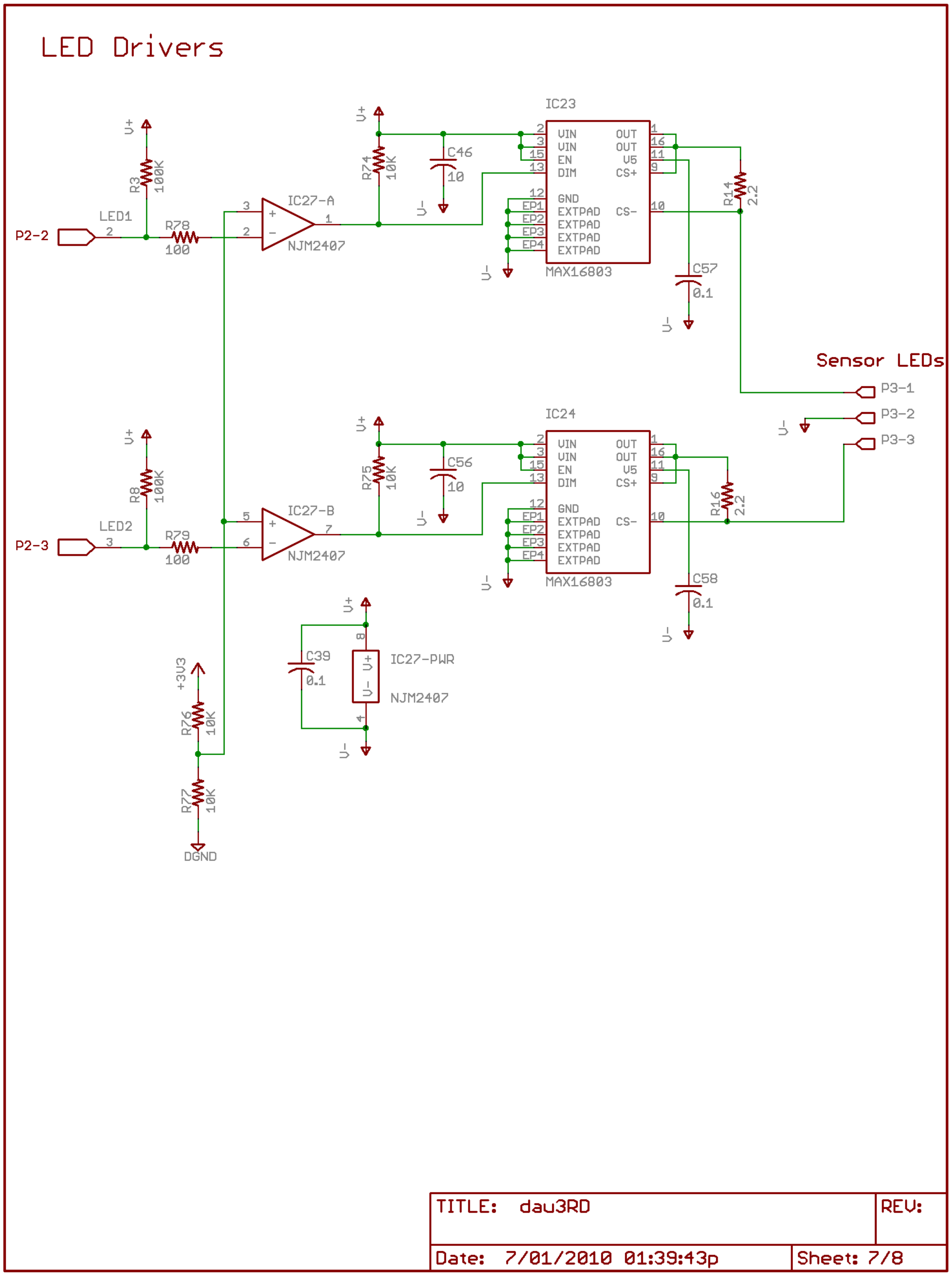
Sheet: 3/8

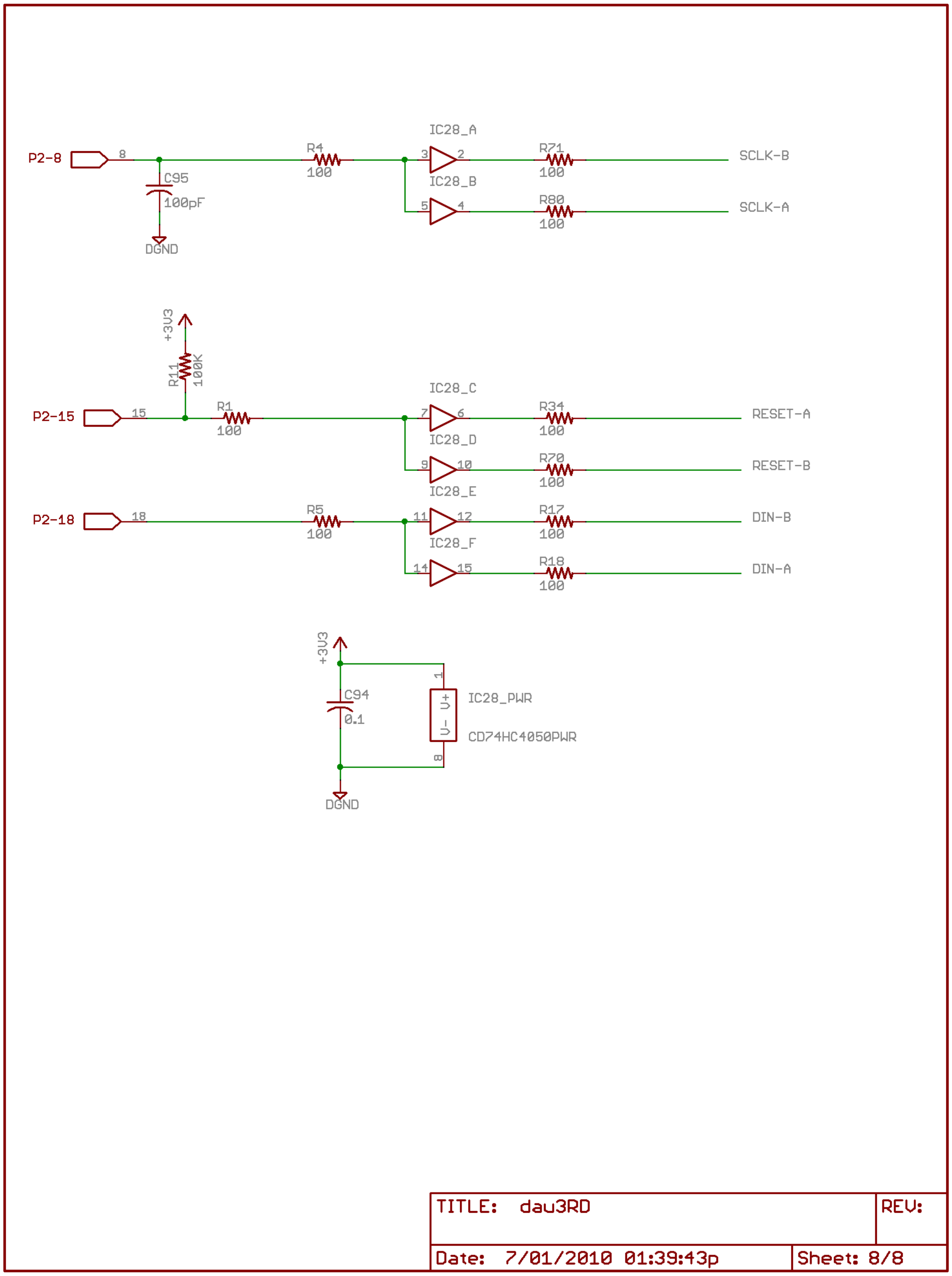

Glucose Sensor
IC14
GLUCOSE-SENSOR-NEW
CH[1..32]
AGND
Analog Circuitry
MAX4478
IC15_A
IC15_B
IC15_C
IC15_D
IC16_A
IC16_B
IC16_C
IC16_D
R38 10M
R39 10M
R40 10M
R41 10M
R42 10M
R43 10M
R44 10M
R45 10M
C15 0.001
C61 0.001
C62 0.001
C63 0.001
C64 0.001
C65 0.001
C66 0.001
C67 0.001
+2.048V
-2.048V
VDD
VSS
C24 10
C25 10
C26 10
C29 10
IC6-ANALOG
IC7-ANALOG
C48 0.1
C49 0.1
IC15_PWR MAX4478
IC16_PWR MAX4478
TITLE: dau3RD
REV:
Date: 7/01/2010 01:39:43p
Sheet: 4/8



TITLE: dau3RD		REV:
Date: 7/01/2010 01:39:43p	Sheet: 5/8	


CH[1..32]
CH18
CH19
CH20
CH17
CH10
CH11
CH12
CH9
CH2
CH3
CH4
CH1
MAX4478
IC20_A
IC20_B
IC20_C
IC20_D
IC21_A
IC21_B
IC21_C
IC21_D
IC22_A
IC22_B
IC22_C
IC22_D
R58 10M
R59 10M
R60 10M
R61 10M
R62 10M
R63 10M
R64 10M
R65 10M
R66 10M
R67 10M
R68 10M
R69 10M
C80 0.001
C81 0.001
C82 0.001
C83 0.001
C84 0.001
C85 0.001
C86 0.001
C87 0.001
C88 0.001
C89 0.001
C90 0.001
C91 0.001
+2.048V
-2.048V
VDD
VSS
AGND
C31 10
C41 10
C33 10
C44 10
C34 10
C47 10
C53 0.1
C54 0.1
C55 0.1
IC11-ANALOG
IC12-ANALOG
IC13-ANALOG
AVDD AGND VREFP VREFN
MUX-P MUX-N ADC-N ADC-P
AIN0 AIN1 AIN2 AIN3 AIN4 AIN5 AIN6 AIN7 AIN8 AIN9 AIN10 AIN11 AIN12 AIN13 AIN14 AIN15
AINCOM THPAD EXP
IC20_PWR MAX4478
IC21_PWR MAX4478
IC22_PWR MAX4478
U+ U-
TITLE: dau3RD
REV:
Date: 7/01/2010 01:39:43p
Sheet: 6/8


LED Drivers
P2-2
LED1
2
R3
100K
U+
R78
100
IC27-A
NJM2407
R74
10K
C46
10
IC23
VIN
VIN
EN
DIM
GND
EXTPAD
EXTPAD
EXTPAD
EXTPAD
OUT
OUT
V5
CS+
CS-
MAX16803
R14
2.2
C57
0.1
U-
Sensor LEDs
P3-1
P3-2
P3-3
P2-3
LED2
3
R8
100K
R79
100
IC27-B
NJM2407
R75
10K
C56
10
IC24
MAX16803
R16
2.2
C58
0.1
C39
0.1
IC27-PWR
NJM2407
+3V3
R76
10K
R77
10K
DGND
TITLE: dau3RD
REV:
Date: 7/01/2010 01:39:43p
Sheet: 7/8


IC28_A
P2-8
8
R4
100
3
2
R71
100
SCLK-B
C95
100pF
IC28_B
5
4
R80
100
SCLK-A
DGND
+3U3
R11
100K
IC28_C
P2-15
15
R1
100
7
6
R34
100
RESET-A
IC28_D
9
10
R70
100
RESET-B
IC28_E
P2-18
18
R5
100
11
12
R17
100
DIN-B
IC28_F
14
15
R18
100
DIN-A
+3U3
C94
0.1
1
U- U+
8
IC28_PWR
CD74HC4050PWR
DGND
TITLE: dau3RD
REV:
Date: 7/01/2010 01:39:43p
Sheet: 8/8

APPENDIX D

MCU C SOFTWARE FOR THE ADS1278 DAU

Filename: Dau3-equal-space.c:

```
#include <p33FJ128GP708.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dau_GP.h"
#include <delay.h>
#include <uart.h>
#include <clockSwitch.h>
#include <math.h>

/* March 16, 2012                                    */
/* This project is designed to filter data from the Rev C. DAU PC Board. */
/* The software utilizes a combined lock-in filter comprised of         */
/* shifting the spectrum of input data by half the sample rate and      */
/* simultaneously low pass filtering the samples.                       */
/* The software filters two independent sets of 32 channels.            */
/* Each set of channels measured data with its LED on and off.          */
/* The Rev. C. DAU contains four ADS1278 ADCs with eight channels each. */
/* Due to the 40 sample period latency of the software runs through a   */
/* loop recording 10 samples each for LED1 on, both LEDs off, LED2 on,  */
/* and both LEDs off.                                                   */

_FOSCSEL(FNOSC_FRCDIV16);                    // FRC
_FOSC(FCKSM_CSECME & OSCIOFNC_OFF  & POSCMD_EC);
                                                        // Clock Switching
and Fail Safe Clock Monitor is Enabled
                                                        // OSC2 Pin
Function: OSC2 is Clock Output
                                                        // Primary
Oscillator Mode: EC

_FWDT(FWDTEN_OFF);              // Watchdog Timer Enabled/disabled by user
software
                                                        // (LPRC can be
disabled by clearing SWDTEN bit in RCON register
_FPOR(FPWRT_PWR1);                          // Turn off the power-up timers.
```

```
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);          // Turn off Code
Protection & Write protection


/****************Global Variables****************/
long led1_in[32], led1_b1[32], led1_out[32];
long led2_in[32], led2_b1[32], led2_out[32];
long adc_temp[8]; // Temperature readings from adcs
unsigned long chtemp;
int raw_led[96];
unsigned int writeCtr, recordFlag, usb1Flag, dataSet;
unsigned int numsets, nrecord, nseconds;
unsigned int flagClock, ainSelect, ledFlag;
unsigned int transFlag;
long transLevel;
unsigned char regdata[8][8];
long unfilteredSamples[64];
int unfilteredCtr;
int startupFlag, startupDelay, phaseCtr;
/************************************************/

/**************** main Function ****************/
int main (void)
{
	unsigned int cycleCtr, numCycles;

	numCycles = 19;
	powerUp();                              // Initialize I/O pins

	// Configure Oscillator to operate the device at 40 MHz
	// Fin = 16 MHz
	// Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
	// Fosc= 16 MHz*40/(4*2) = 80MHz, Fcy = 40 MHz
	PLLFBD = 38;                            // M = 40
	CLKDIVbits.PLLPOST = 0;                         // N2=2
	CLKDIVbits.PLLPRE = 2;                  // N1=4
	clockSwitch(NOSC_PRIPLL);               // Clock is switched to
primary

	delay_ms(1000);                                 //  delay

	// Initialize SPI1 in 8-bit Master mode
	// SPI clock rate is Fcy/80 or 40MHz/8 = 5 MHz actually now 2 MHz
	initSpi1(0);
```

```
        // Initialize SPI2 in 8-bit framed Master mode
        // Use SCLK2 to provide 2 MHz ADC clock
        // SPI2 clock rate is Fcy/20 or 40MHz/20 = 2 MHz
        initSpi2();

        // Initialize Timer2
        // Timer2 no longer utilized
        // initTimer2();
        // Early software utilized timer2 and TESTLED2 for ADC clock

        // Read transient level selection from PC
        setupTransient();

        // Initialize variables
        flagClock = 1;                                    // State of ADC
clock
        clearData();
        dataSet = 0;
        startupFlag = 1;
        startupDelay = 0;
        transFlag = 0;

        // Enable timer2
        // Timer2 no longer utilized T2CONbits.TON = 1;

        //  Synchronize ADCS
        SYNC_ = LOW;
        delay_ms(2); // Delay
        SYNC_ = HIGH;
        // ADCs synchronizing

        //Main loop
        while(1)
   {
                transFlag = 0;                      // Reset flag after sending
samples to PC
                recordFlag = 1;
                for (cycleCtr=0; cycleCtr<numCycles; cycleCtr++)
                {

                        // One lock-in cycle
                        dataSet = 0;            // LED1 ON
                        for (phaseCtr=0; phaseCtr<10; phaseCtr++)
```

```
		{
			getData();
		}

		dataSet = 1;		// LED1 OFF
		for (phaseCtr=0; phaseCtr<10; phaseCtr++)
		{
			getData();
		}

		dataSet = 2;		// LED2 ON
		for (phaseCtr=0; phaseCtr<10; phaseCtr++)
		{
			getData();
		}

		dataSet = 3;		// LED2 OFF
		for (phaseCtr=0; phaseCtr<10; phaseCtr++)
		{
			getData();
		}

	} // Finished with numCycles lock in cycles

	// Send data to PC
	usb1Flag = 1;
	recordFlag = 0;
	writeCtr = 0;
	dataSet = 0;
	for (phaseCtr=0; phaseCtr<10; phaseCtr++)
	{
		if(phaseCtr<4) usbWritedata1On();
		getData();
	}

	writeCtr = 0;
	dataSet = 1;
	for (phaseCtr=0; phaseCtr<10; phaseCtr++)
	{
		if(phaseCtr<4) usbWritedata2On();
		getData();
	}

	dataSet = 2;
```

```
		for (phaseCtr=0; phaseCtr<10; phaseCtr++)
		{
			if(phaseCtr<1) usbWritetranslevel();
			getData();
		}

		dataSet = 3;
		for (phaseCtr=0; phaseCtr<10; phaseCtr++)
		{
			if(phaseCtr<1) usbWritetransient();
			getData();
		}

	} // While loop

	// Turn LEDs OFF
	LED1 = HIGH;
	LED2 = HIGH;
	TESTLED1 = LOW;
	TESTLED2 = LOW;

	return 0;
}
```

Filename: dauFunctions.c:

```
#include <p33FJ128GP708.h>
#include <stdio.h>
#include "dau_GP.h"
#include <delay.h>
#include <uart.h>
#include <math.h>
/*---------------------------MCU Powerup routine------------------------------*/
/*
Note: This routine initializes the MCU output lines.
*/
void powerUp(void){
//unsigned char data;
//Declare dsPICs I/O lines connected to DAU control pins as outputs
//	TRISAbits.TRISA15 = 0;		// RA15 is output
// The following ODCA statements do not have an effect as A15,A14 do not
exist
//	ODCAbits.ODCA15 = 0;		// Open drain output disabled
//	ODCAbits.ODCA14 = 0;		// Open drain output disabled
```

```
//      TRISA &= 0x79FF;                // RA15, RA10, RA9 are outputs; others
are inputs
        TRISA = 0x0004;                         // all outputs except RA2
        TRISB = 0x0003;                 // all outputs except RB0, RB1
        TRISD = 0x7000;                         // RD15, RD11, RD10, RD9,
RD8,RD7~RD0 are outputs; others are inputs
        TRISF = 0xFFFC;                         // RF0,RF1 are outputs
        TRISG = 0x0FFC;                         //
RG0,RG1,RG12,RG13,RG14,RG15 outputs
        AD1PCFGL=0xFFFF;                        // all analog channel pins are
digital
        AD1PCFGH=0xFFFF;                        //
//      TRISFbits.TRISF0 = 0;
//      TRISFbits.TRISF1 = 0;
//      TRISGbits.TRISG0 = 0;
//      TRISGbits.TRISG1 = 0;
//      TRISGbits.TRISG12 = 0;
//      TRISGbits.TRISG13 = 0;
//      TRISGbits.TRISG14 = 0;
//      TRISGbits.TRISG15 = 0;

        delay_us(10);

//  Initial configuration of outputs
        LED1 = HIGH;                                    // LEDs disabled - LED1
and LED2 active low
        LED2 = HIGH;
        TESTLED1 = LOW;
        TESTLED2 = LOW;

//Enable the digital buffers connecting dsPIC I/Os to DAU
//Enable the clock driver IC and the clock oscillator
        BUFFEN1_ = LOW;                                 //Enable digital buffers 1
& 2
        BUFFEN2_ = LOW;                                 //
        EOH = HIGH;                                             //Enable clock
oscillator output
        G1 = HIGH;                                      //Bank 1 of clock driver
enabled
        G2 = HIGH;                                      //Bank 2 enabled

//Set default signal levels for USB
        USB_WR = LOW;                                   // default WR line value
        USB_RD_ = HIGH;                                 // default RD line value
```

```
//Set SYNC_ HIGH
        SYNC_ = HIGH;

        return;
}
/*-----------------------------------------------------------------------*/
/*---------------------------ADC Initialization Routine---------------------*/
void initADCs (void)
{
/*---------------------------------------------
Initialize ADCs - for ADS1258
----------------------------------------------*/

//      Reset ADCs
//      RESET_ = LOW;
//      delay_ms(10);
//      RESET_ = HIGH;

        return;
}
/*-----------------------------------------------------------------------*/

void configADCs(void)
{
/*
// Routine to configure ADS1258 ADCs for next sample
// Added write to clear out register 6 after temp reading
// Removed write to register 6 since no temp measurements

        unsigned int adcctr;
        unsigned char data;

        ainSelect++;
        if(ainSelect>3) ainSelect = 0;

        for(adcctr=0; adcctr<8; adcctr++)
        {
                // Select IC
                if ((adcctr & 0x01) == 0x01) CS_A = HIGH;
                else CS_A = LOW;
                if ((adcctr & 0x02) == 0x02) CS_B = HIGH;
                else CS_B = LOW;
                if ((adcctr & 0x04) == 0x04) CS_C = HIGH;
```

```
		else CS_C = LOW;

		delay_us(1);

		// Write to ADC configuration registers
		CS_EN = HIGH;
		delay_us(1);

		while (SPI1STATbits.SPITBF);		// wait if SPITBF is set
		SPI1BUF = 0x74;			// Multiple register write,
starting with register 0x04
		while (SPI1STATbits.SPIRBF==0);			// wait if SPITBF is
set
		data = SPI1BUF;

		if(ainSelect == 0)
		{

			//Select first AIN
			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
			SPI1BUF = 0x10;					// Use
AIN4 only
			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
			data = SPI1BUF;

			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
			SPI1BUF = 0x00;					//
don't Use AIN8&9
			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
			data = SPI1BUF;

//			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
//			SPI1BUF = 0x00;					//
don't Use temp sensor
//			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
//			data = SPI1BUF;

		}
		else if(ainSelect == 1)
		{
			//Select second AIN
			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
			SPI1BUF = 0x20;					// Use
AIN5 only
			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
			data = SPI1BUF;

			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
			SPI1BUF = 0x00;					//
don't Use AIN8&9
			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
			data = SPI1BUF;

//			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
//			SPI1BUF = 0x00;					//
don't Use temp sensor
//			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
//			data = SPI1BUF;

		}
		else if(ainSelect == 2)
		{
			//Select third AIN
			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
			SPI1BUF = 0x00;					//
Don't use AIN 4&5
			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
			data = SPI1BUF;

			while (SPI1STATbits.SPITBF);		// wait if SPITBF is
set
			SPI1BUF = 0x01;					// Use
AIN8 only
			while (SPI1STATbits.SPIRBF==0);			// wait if
SPITBF is set
                        data = SPI1BUF;

//                      while (SPI1STATbits.SPITBF);            // wait if SPITBF is
set
//                      SPI1BUF = 0x00;                                         //
don't Use temp sensor
//                      while (SPI1STATbits.SPIRBF==0);                 // wait if
SPITBF is set
//                      data = SPI1BUF;

                }
                else
                {
                        //Select fourth AIN
                        while (SPI1STATbits.SPITBF);            // wait if SPITBF is
set
                        SPI1BUF = 0x00;                                         //
Don't use AIN 4&5
                        while (SPI1STATbits.SPIRBF==0);                 // wait if
SPITBF is set
                        data = SPI1BUF;

                        while (SPI1STATbits.SPITBF);            // wait if SPITBF is
set
                        SPI1BUF = 0x02;                                         // Use
AIN9 only
                        while (SPI1STATbits.SPIRBF==0);                 // wait if
SPITBF is set
                        data = SPI1BUF;

//                      while (SPI1STATbits.SPITBF);            // wait if SPITBF is
set
//                      SPI1BUF = 0x00;                                         //
don't Use temp sensor
//                      while (SPI1STATbits.SPIRBF==0);                 // wait if
SPITBF is set
//                      data = SPI1BUF;

                }

                CS_EN = LOW;
//              delay_us(1);                            // Delay, must be > 2 ADC clock
cycles
```

```
        }
*/
}

/*------------------------------------------------------------------------*/

void configTemp(void)
{
/*
// Routine to configure ADS1258 ADCs for recording temperature

        unsigned int adcctr;
        unsigned char data;

        for(adcctr=0; adcctr<8; adcctr++)
        {
                // Select IC
                if ((adcctr & 0x01) == 0x01) CS_A = HIGH;
                else CS_A = LOW;
                if ((adcctr & 0x02) == 0x02) CS_B = HIGH;
                else CS_B = LOW;
                if ((adcctr & 0x04) == 0x04) CS_C = HIGH;
                else CS_C = LOW;

                // Write to ADC configuration registers
                CS_EN = HIGH;
                delay_us(1);

                while (SPI1STATbits.SPITBF);            // wait if SPITBF is set
                SPI1BUF = 0x74;                         // Multiple register write,
starting with register 0x04
                while (SPI1STATbits.SPIRBF==0);                 // wait if SPITBF is
set
                data = SPI1BUF;

                // Select temperature measurement only
                while (SPI1STATbits.SPITBF);            // wait if SPITBF is set
                SPI1BUF = 0x00;                                         // Don't use
any AIN's
                while (SPI1STATbits.SPIRBF==0);                 // wait if SPITBF is
set
                data = SPI1BUF;

                while (SPI1STATbits.SPITBF);            // wait if SPITBF is set
                SPI1BUF = 0x00;                                    // Don't use
any AIN's
                while (SPI1STATbits.SPIRBF==0);                // wait if SPITBF is
set
                data = SPI1BUF;

                while (SPI1STATbits.SPITBF);              // wait if SPITBF is set
                SPI1BUF = 0x08;                                    // Select
temp reading
                while (SPI1STATbits.SPIRBF==0);                // wait if SPITBF is
set
                data = SPI1BUF;

                CS_EN = LOW;
                delay_us(1);                          // Delay, must be > 2 ADC clock
cycles
        }
*/
}


/*----------------------------SPI1 Initialization Routine----------------------*/
void initSpi1 (unsigned char bitMode)
{
/* Configure and Enable SPI1 */
//      if (bitMode) SPI1CON1 = 0x073A;         //SS' disabled,16-bit Master
Mode, CKP = 0, CKE = 1, SCK = Fcy/8
//      else SPI1CON1 = 0x033A;                 //SS' disabled,8-bit Master Mode,
CKP = 0, CKE = 1, SCK = Fcy/8
        SPI1CON1 = 0x032E;                              //SS' disabled,8-bit Master
Mode, CKP = 0, CKE = 1, SCK = Fcy/20, SMP=1
   SPI1STATbits.SPIROV = 0;             //clear SPIROV
   SPI1STATbits.SPIEN = 1;                      // Enable the spi module */
        return;
}
/*----------------------------SPI1 Initialization Routine----------------------*/
void initSpi2 (void)
{
/* Configure and Enable SPI2 */
/* Use SPI2 clock to provide 2 MHz ADC clock */
        SPI2CON1 = 0x082E;                              // DISSDO=1,8-
bit,SMP=0,CKE=0,SSEN=0,CKP=0,Master,SCK=Fcy/20
        SPI2CON2 = 0xA002;                              //
FRMEN=1,SPIFSD=0,FRMPOL=1,FRMDLY=1
```

```
   SPI2STATbits.SPIROV = 0;              //clear SPIROV
   SPI2STATbits.SPIEN = 1;                        // Enable the spi module */
         return;
}
/*-----------------------------------------------------------------------*/
void getData (void)
{

// To test scale factor apply voltage divider between 3.3V and LED2
// Connect middle node of divider to negative input of adc
// Connect positive ADC input to 2.5V
// LED2 low - result should be 2*(2.5 - 1.7) = 1.6V
// LED2 high - result should be 2*(2.5 - 3.3) = -1.6V

         unsigned int i;
         long temp, temp_b1;
         long onInput1, offInput1;
         long onInput2, offInput2;
         long difference, error;

         // Wait until data is available
         while (DREADY_ == HIGH);

         // Modulate LEDs
         switch (dataSet)
         {
                  case 0: // Turn on LED1
                           LED1 = LOW;
                           LED2 = HIGH;
                  break;

                  case 1: // Turn LEDs off
                           LED1 = HIGH;
                           LED2 = HIGH;
                  break;

                  case 2: // TUrn on LED2
                           LED1 = HIGH;
                           LED2 = LOW;
                  break;

                  case 3: // Turn LEDs off
                           LED1 = HIGH;
                           LED2 = HIGH;
			break;

			default:
			break;
		}

		//	Need one ADC clock cycle delay after assertion of Data Ready
		//delay_us(2);

		// Timing measurement
		// TESTLED1 = HIGH;

		// Read 32 channels of data - 3 bytes each
		for(i=0;i<96;i++)
		{
			while (SPI1STATbits.SPITBF);		// wait if SPITBF is set
			SPI1BUF = 0x00;						// Write to
SPI data register to start SCK
			while (SPI1STATbits.SPIRBF == 0);	// Wait for reception to
complete
			raw_led[i] = SPI1BUF;				// Load received
data to data array
		}

		if(startupFlag == 1)
		{
			startupDelay++;
			if(startupDelay>40) startupFlag = 0;
		}

		// Timing measurement
		// TESTLED1 = LOW;

		// If recordFlag is set, convert and filter data
		if ((recordFlag == 1)&&(startupFlag==0)&&(phaseCtr==5))
		{

			// Timing measurement
			// TESTLED1 = HIGH;

			for (i=0; i<32;i++)
			{
				// First convert raw data into 32 bit integer called temp
				temp = raw_led[i*3];
```

```
                    temp = temp << 8;
                    temp += raw_led[i*3 + 1];
                    temp = temp << 8;
                    temp += raw_led[i*3 + 2];
                    temp = temp << 8;

                    // Filter sample
                    switch (dataSet)
                    {
                            case 0:

//                                  // Check voltage scaling
//                                  if(unfilteredCtr<64)
//                                  {
//                                          if(i==24)
//                                          {
//                                                  unfilteredSamples[unfilteredCtr]
= temp;
//                                                  unfilteredCtr++;
//                                          }
//                                  }

                                    // Implement a single pole low pass filter and
invert every other sample
                                    // Y(n) = Y(n-1) + (X(n) - X(n-1))/2^7 - Y(n-
1)/2^6

                                    // New input is temp
                                    // Previous input is led1_in[i]
                                    // Store value of led1_b1[i] for second filter
                                    temp_b1 = led1_b1[i];
                                    // Combined lock-in filter
                                    led1_b1[i] -= (led1_b1[i]) >> 6;
                                    led1_b1[i] += (led1_in[i] - temp) >> 7;
                                    // Store input value
                                    led1_in[i] = temp;
                                    // To use only a one pole filter, place ledn_b1
output into output array
                                    // One pole filter: Place output in array for usb
write subroutine
                                    // One pole filter: led1_out[i] = led1_b1[i];

                                    // Implement another single pole low pass
filter
```

```
                                        // Y(n) = Y(n-1) + (X(n) + X(n-1))/2^7 - Y(n-
1)/2^6

                                        // New input is led1_b1[i]
                                        // Previous input is led1_b1[i] before it was
updated above
                                        led1_out[i] -= (led1_out[i]) >> 6;
                                        led1_out[i] += (led1_b1[i] + temp_b1) >> 7;

                                        // Transient monitoring - store new led on
input
                                        onInput1 = temp;

                                        break;

                                case 1:

//                                      // Check voltage scaling
//                                      if(unfilteredCtr<64)
//                                      {
//                                              if(i==24)
//                                              {
//                                                      unfilteredSamples[unfilteredCtr]
= temp;
//                                                      unfilteredCtr++;
//                                              }
//                                      }

                                        // Implement a single pole low pass filter and
invert every other sample
                                        // Y(n) = Y(n-1) + (X(n-1) - X(n))/128 - Y(n-
1)/64

                                        // New input is temp
                                        // Previous input is led1_in[i]
                                        // Store value of led1_b1[i] for second filter
                                        temp_b1 = led1_b1[i];
                                        // Combined lock-in filter
                                        led1_b1[i] -= (led1_b1[i]) >> 6;
                                        led1_b1[i] += (temp - led1_in[i]) >> 7;
                                        // Store input value
                                        led1_in[i] = temp;
                                        // To use only a one pole filter, place ledn_b1
output into output array
```

```
                                        // One pole filter: Place output in array for usb
write subroutine
                                        // One pole filter: led1_out[i] = led1_b1[i];

                                        // Implement another single pole low pass
filter
                                        // Y(n) = Y(n-1) + (X(n) + X(n-1))/128 - Y(n-
1)/64

                                        // New input is led1_b1[i]
                                        // Previous input is led1_b1[i] before it was
updated above
                                        led1_out[i] -= (led1_out[i]) >> 6;
                                        led1_out[i] += (led1_b1[i] + temp_b1) >> 7;

                                        // Transient monitoring - store new led off
input
                                        offInput1 = temp;
                                        // Calculate difference between on and off
data
                                        difference = onInput1 - offInput1;
                                        // Find absolute value of difference
                                        if((difference & 0x80000000) == 0x80000000)
                                        {
                                             difference= ~difference;
                                             difference++;
                                        }
                                        // Difference has not been filtered and thus
represents peak to peak difference
                                        // between on and off values
                                        // LIA output has been filtered and represents
the amplitude of on and off sample difference
                                        // Thus divide difference by two, calculate
error, and compare with half the limit in volts
                                        difference = difference >> 1;
                                        // Calculate error: ledn_out is a positive
number
                                        error = difference - led1_out[i];
                                        // Find absolute value of error
                                        if((error & 0x80000000) == 0x80000000)
                                        {
                                             error= ~error;
                                             error++;
                                        }
                                        // Set flag if difference is greater than limit
divided by two
                                        if(error > (transLevel >> 1)) transFlag = 1;
                                        break;

                                case 2:
                                        // Implement a single pole low pass filter and
invert every other sample
                                        // Y(n) = Y(n-1) + (X(n) - X(n-1))/128 - Y(n-
1)/64

                                        // New input is temp
                                        // Previous input is led2_in[i]
                                        // Store value of led2_b1[i] for second filter
and transient monitoring
                                        temp_b1 = led2_b1[i];
                                        // Combined lock-in filter
                                        led2_b1[i] -= (led2_b1[i]) >> 6;
                                        led2_b1[i] += (led2_in[i] - temp) >> 7;
                                        // Store input value
                                        led2_in[i] = temp;
                                        // One pole filter: Place output in array for usb
write subroutine
                                        // One pole filter: led2_out[i] = led2_b1[i];

                                        // Implement another single pole low pass
filter
                                        // Y(n) = Y(n-1) + (X(n) + X(n-1))/128 - Y(n-
1)/64

                                        // New input is led2_b1[i]
                                        // Previous input is led2_b1[i] before it was
updated above
                                        led2_out[i] -= (led2_out[i]) >> 6;
                                        led2_out[i] += (led2_b1[i] + temp_b1) >> 7;

                                        // Transient monitoring - store new led on
input
                                        onInput2 = temp;
          //                                    // Transient monitoring
          //                                    difference = led1_out[i] - old_out;
          //                                    if((difference & 0x80000000) ==
0x80000000) difference = (~difference)++;
```

```
                //                              if(difference > transientMax)
transientFlag = 1;
                                        break;

                                case 3:
                                        // Implement a single pole low pass filter and
invert every other sample
                                        // Y(n) = Y(n-1) + (X(n-1) - X(n))/128 - Y(n-
1)/64

                                        // New input is temp
                                        // Previous input is led2_in[i]
                                        // Store value of led2_b1[i] for second filter
and transient monitoringr
                                        temp_b1 = led2_b1[i];
                                        // Combined lock-in filter
                                        led2_b1[i] -= (led2_b1[i]) >> 6;
                                        led2_b1[i] += (temp - led2_in[i]) >> 7;
                                        // Store input value
                                        led2_in[i] = temp;
                                        // One pole filter: Place output in array for usb
write subroutine
                                        // One pole filter: led2_out[i] = led2_b1[i];

                                        // Implement another single pole low pass
filter
                                        // Y(n) = Y(n-1) + (X(n) + X(n-1))/128 - Y(n-
1)/64

                                        // New input is led2_b1[i]
                                        // Previous input is led2_b1[i] before it was
updated above
                                        led2_out[i] -= (led2_out[i]) >> 6;
                                        led2_out[i] += (led2_b1[i] + temp_b1) >> 7;

                                        // Transient monitoring - store new led off
input
                                        offInput2 = temp;
                                        // Calculate difference between on and off
data
                                        difference = onInput2 - offInput2;
                                        // Find absolute value of difference
                                        if((difference & 0x80000000) == 0x80000000)
                                        {
```

```
                                                  difference = ~difference;
                                                  difference++;
                                        }
                                        // Difference has not been filtered and thus
represents peak to peak difference
                                        // between on and off values
                                        // LIA output has been filtered and represents
the amplitude of on and off sample difference
                                        // Thus divide difference by two, calculate
error, and compare with half the limit in volts
                                        difference = difference >> 1;
                                        // Calculate error: ledn_out is a positive
number
                                        error = difference - led2_out[i];
                                        // Find absolute value of error
                                        if((error & 0x80000000) == 0x80000000)
                                        {
                                                  error= ~error;
                                                  error++;
                                        }

                                        // Set flag if difference is greater than limit
divided by two
                                        if(error > (transLevel >> 1)) transFlag = 1;

                              break;

                              default:
                              break;
                    } // Switch statement
          } // For loop

          // Timing measurement
          // TESTLED1 = LOW;

     } // If statement

     return;
}

/*-----------------------------------------------------------------------------*/

void getTemp (void)
{
```

```
// Configure ADS1258 for a temperature reading
/*
        unsigned int newFlag, adc;
        int data, status, byte1, byte2, byte3;
        long temp;

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

                // Select IC
                if ((adc & 0x01) == 0x01) CS_A = HIGH;
                else CS_A = LOW;
                if ((adc & 0x02) == 0x02) CS_B = HIGH;
                else CS_B = LOW;
                if ((adc & 0x04) == 0x04) CS_C = HIGH;
                else CS_C = LOW;

                // Read status byte and temperature sample

                newFlag = 0;
                while(newFlag==0)
                {

                        // Set CS_EN
                        CS_EN = HIGH;
                        delay_us(1);

                        // Read ADC data and test status byte bit 7 (NEW data)

                        // Channel data read command, multiple bytes
                        while (SPI1STATbits.SPITBF);// wait if SPITBF is set
                        SPI1BUF = 0x30;
                        while (SPI1STATbits.SPIRBF==0);         // wait if SPITBF is
set
                        data = SPI1BUF;

                        // Read status byte
                        while (SPI1STATbits.SPITBF);// wait if SPITBF is set
                        SPI1BUF = 0x00;
                        while (SPI1STATbits.SPIRBF==0);         // wait if SPITBF is
set
                        status = SPI1BUF;
```

```
				// Update newFlag
				if ((status & 0x80) == 0x80) newFlag = 1;

				// Read three data bytes

				while (SPI1STATbits.SPITBF);// wait if SPITBF is set
				SPI1BUF = 0x00;			// Write to SPI data register to
start SCK
				while (SPI1STATbits.SPIRBF == 0);// wait for reception to
complete
				byte1 = SPI1BUF;	// load received data to data array

				while (SPI1STATbits.SPITBF);// wait if SPITBF is set
				SPI1BUF = 0x00;			// Write to SPI data register to
start SCK
				while (SPI1STATbits.SPIRBF == 0);// wait for reception to
complete
				byte2 = SPI1BUF;	// load received data to data array

				while (SPI1STATbits.SPITBF);// wait if SPITBF is set
				SPI1BUF = 0x00;			// Write to SPI data register to
start SCK
				while (SPI1STATbits.SPIRBF == 0);// wait for reception to
complete
				byte3 = SPI1BUF;	// load received data to data array

				// Reset CS_EN
				CS_EN = LOW;
			}

			// Convert raw data to integer
			temp = byte1;
			temp = temp << 8;
			temp |= byte2;
			temp = temp << 8;
			temp |= byte3;
			temp = temp << 8;		// Form Q0.31 number

			// store in array
			adc_temp[adc] = temp;

		}
*/
		return;
```

```
}


/*---------------------------------------------------------------------------*/
void usbWritedata1On (void)
// Sends data in led1_on array via usb, eight channels at a time
{
	long templong;
	unsigned int i;

	// Timing measurement
	// TESTLED1 = HIGH;

	if (usb1Flag == 1)
	{
		// Send preamble - four bytes of 0xFF
		for(i=0; i<4; i++)
		{
			while(USB_TXE_ == HIGH);	// wait if TXE is high,
proceed if TXE is low
			USB_WR = HIGH;			// Set WR low to write
data
			PORTDbits.LSB = 255;
			delay_us(1);
			USB_WR = LOW;			// Set WR high
		}
		usb1Flag = 0;

//		while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if
TXE is low
//		USB_WR = HIGH;			// Set WR low to write data
//		if (transientFlag == 1) PORTDbits.LSB = 255;
//		else PORTDbits.LSB = 0;
//		delay_us(1);
//		USB_WR = LOW;			// Set WR high
	}

	for(i=0; i<8; i++)
	{
		// Use next statement for testing voltage scale factor
		//templong = unfilteredSamples[writeCtr];
		templong = led1_out[writeCtr];
		while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if
TXE is low
```

```
                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                writeCtr++;
        }

        // Timing measurement
        // TESTLED1 = LOW;

        return;
}
/*-----------------------------------------------------------------------*/
void usbWritedata1Off (void)
// Sends data in led1_off array via usb, eight channels at a time
```

```
{
        long templong;
        unsigned int i;

        for(i=0; i<8; i++)
        {
                templong = 0;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                writeCtr++;
        }
        return;
}
/*---------------------------------------------------------------------------*/
void usbWritedata2On (void)
// Sends data in led2_on array via USB, eight channels at a time
{
        long templong;
        unsigned int i;

        for(i=0; i<8; i++)
        {
                // Use next statement for testing voltage scale factor
                //templong = unfilteredSamples[writeCtr];
                templong = led2_out[writeCtr];
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low
```

```
                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                writeCtr++;
        }

        return;
}
/*------------------------------------------------------------------------*/
void usbWritedata2Off (void)
// Sends data in led2_off array via USB, eight channels at a time
{
        long templong;
        unsigned int i;

        for(i=0; i<8; i++)
        {
                templong = 0;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                templong = templong>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low

                USB_WR = HIGH;                  // Set WR high
                PORTDbits.LSB = templong;
                delay_us(1);
                USB_WR = LOW;                   // Set WR low

                writeCtr++;
        }

        return;
}

/*--------------------------------------------------------------------------*/
void usbWritetransient (void)
// Sends status of transient detection to PC
{
        long templong;

        templong = 0x00000000;
        if (transFlag) templong = 0x00000001;

        while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if TXE is
low

        USB_WR = HIGH;                  // Set WR high
        PORTDbits.LSB = templong;
        delay_us(1);
        USB_WR = LOW;                   // Set WR low

        templong = templong>>8;
        while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if TXE is
low

        USB_WR = HIGH;                  // Set WR high
        PORTDbits.LSB = templong;
        delay_us(1);
        USB_WR = LOW;                   // Set WR low
```

```
	templong = templong>>8;
	while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if TXE is
low

	USB_WR = HIGH;			// Set WR high
	PORTDbits.LSB = templong;
	delay_us(1);
	USB_WR = LOW;			// Set WR low

	templong = templong>>8;
	while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if TXE is
low

	USB_WR = HIGH;			// Set WR high
	PORTDbits.LSB = templong;
	delay_us(1);
	USB_WR = LOW;			// Set WR low

	return;
}

/*------------------------------------------------------------------------*/
void usbWritetranslevel (void)
// Sends status of transient detection to PC
{
	long templong;

	templong = transLevel;

	while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if TXE is
low

	USB_WR = HIGH;			// Set WR high
	PORTDbits.LSB = templong;
	delay_us(1);
	USB_WR = LOW;			// Set WR low

	templong = templong>>8;
	while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if TXE is
low

	USB_WR = HIGH;			// Set WR high
	PORTDbits.LSB = templong;
	delay_us(1);
	USB_WR = LOW;			// Set WR low

	templong = templong>>8;
	while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if TXE is
low

	USB_WR = HIGH;			// Set WR high
	PORTDbits.LSB = templong;
	delay_us(1);
	USB_WR = LOW;			// Set WR low

	templong = templong>>8;
	while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if TXE is
low

	USB_WR = HIGH;			// Set WR high
	PORTDbits.LSB = templong;
	delay_us(1);
	USB_WR = LOW;			// Set WR low

	return;
}

/*------------------------------------------------------------------------*/
void usbWriteTemp (void)
// Sends data in adc_temp array via usb
{
	unsigned int i;
//	unsigned int ctr;

	for(i=0; i<8; i++)
	{
/*
		while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if
TXE is low
		PORTDbits.LSB = adc_temp[i];
		// Pulse WR pin high for 200 ns
		USB_WR = HIGH;			// take WR high
		for(ctr=0; ctr<24; ctr++);
		//delay_us(1);
		USB_WR = LOW;			// take WR low to write data

		adc_temp[i] = adc_temp[i]>>8;
		while(USB_TXE_ == HIGH);	// wait if TXE is high, proceed if
```

```
TXE is low
                PORTDbits.LSB = adc_temp[i];
                // Pulse WR pin high for 200 ns
                USB_WR = HIGH;                  // take WR high
                for(ctr=0; ctr<24; ctr++);
                USB_WR = LOW;                   // take WR low to write data

                adc_temp[i] = adc_temp[i]>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low
                PORTDbits.LSB = adc_temp[i];
                // Pulse WR pin high for 200 ns
                USB_WR = HIGH;                  // take WR high
                for(ctr=0; ctr<24; ctr++);
                USB_WR = LOW;                   // take WR low to write data

                adc_temp[i] = adc_temp[i]>>8;
                while(USB_TXE_ == HIGH);        // wait if TXE is high, proceed if
TXE is low
                PORTDbits.LSB = adc_temp[i];
                // Pulse WR pin high for 200 ns
                USB_WR = HIGH;                  // take WR high
                for(ctr=0; ctr<24; ctr++);
                USB_WR = LOW;                   // take WR low to write data
*/
        }

        return;
}

/*---------------------------------------------------------------------------*/
void clearData (void)
{
        unsigned int n;

        // Clear the raw data
        for (n = 0; n<96; n++) raw_led[n] = 0;

        // Clear the channel data arrays
        for(n=0;n<32;n++)
        {
                led1_in[n]=0;
                led1_b1[n]=0;
                led1_out[n]=0;
```

```
            led2_in[n]=0;
            led2_b1[n]=0;
            led2_out[n]=0;
        }
        // Clear the temp variable
        chtemp = 0;
        return;
}

/*-------------------------------------------------------------------------*/
void setupTransient (void)
{
        unsigned int i, itest;
        char inChar;
        char inString[10];

//      Set PortD<7:0> and PortD<14:12> as inputs
        TRISD = 0x70FF;     //RD15, RD11, RD10, RD9, and RD8 are outputs;
others are inputs

        TESTLED1 = HIGH;
        delay_ms(1000);
        TESTLED1 = LOW;
   delay_ms(1000);
        TESTLED1 = HIGH;
        delay_ms(1000);
        TESTLED1 = LOW;
   delay_ms(1000);
        TESTLED1 = HIGH;
        delay_ms(1000);
        TESTLED1 = LOW;
   delay_ms(1000);

        // Read data string from usb port
        // String format: "HLIMITx" where x is integer from 0 to 9
        // Integer          Transient limit
        //   0                          2.5V
        //   1                          1.25V
        //   2                          0.625V
        //   3                          0.313V
        //   4                          0.156V
        //   5                          0.078V
        //   6                          0.039V
        //   7                          0.020V
        //  8                           0.010V
        //  9                           0.005V

        while(USB_RXF_ == HIGH);            // Wait if RXF is high, proceed if
RXF is low
        USB_RD_ = LOW;                          // Set RD_ low
        delay_us(2);
        inChar = PORTDbits.LSB;
        USB_RD_ = HIGH;                         // Set RD_ high
        while (inChar != 72) {              // Wait for H to be read
                while(USB_RXF_ == HIGH);    // Wait if RXF is high, proceed if
RXF is low
                USB_RD_ = LOW;                  // Set RD_ low
                delay_us(2);
                inChar = PORTDbits.LSB;
                USB_RD_ = HIGH;                 // Set RD_ high
        }
        inString[0] = inChar;               // Record character
        for(i=1;i<7;i++)                    // Read the remainder of the
string
        {
                while(USB_RXF_ == HIGH);    // Wait if RXF is high, proceed if
RXF is low
                USB_RD_ = LOW;                  // Set RD_ low
                delay_us(2);
                inChar = PORTDbits.LSB;
                USB_RD_ = HIGH;                 // Set RD_ high
                inString[i] = inChar;
        }

//      Set PortD pin directions to original values
        TRISD = 0x7000;     //RD15, RD11, RD10, RD9, RD8,RD7~RD0 are
outputs; others are inputs
        delay_ms(1000);
        itest = inString[6] - 48;

        transLevel = 0x40000000 >> itest;
//(inString[6] - 48);

        return;

}

/*------------------------------------------------------------------------*/
```

```
void clockSwitchC (void)
{
//	TRISBbits.TRISB14 = 0;		// PORTB 0 is an output
//	LATBbits.LATB14 = HIGH;			// PORTB 0 is high
	TRISBbits.TRISB15 = 0;		// PORTB 0 is an output
	LATBbits.LATB15 = HIGH;			// PORTB 0 is high
//	// Configure Oscillator to operate the device at 40 MHz
//	// Fin = 16 MHz
//	// Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
//	// Fosc= 16 MHz*40/(4*2) = 80MHz, Fcy = 40 MHz
//	PLLFBD = 38;				// M = 40
//	CLKDIVbits.PLLPOST = 0;			// N2=2
//	CLKDIVbits.PLLPRE = 2;		// N1=4

//	while (!OSCCONbits.LOCK);	// wait for OSCCONbits.LOCK to be
set, i.e. PLL to lock

	OSCCONbits.MSB = 0x78;		// Unlock sequence to allow write to
	OSCCONbits.MSB = 0x9A;			// OSCCONH system control
register
	OSCCONbits.MSB = 0x02;		// Primary EC is the desired clock

	OSCCONbits.LSB = 0x46;		// Unlock sequence to allow write to
	OSCCONbits.LSB = 0x57;		// OSCCONL system control register
	OSCCONbits.OSWEN = 1;		// Request a clock switch

	delay_us(100);				// Delay
//	if (!OSCCONbits.LOCK) LATBbits.LATB14 = LOW;	// PORTB14 low
indicates PLL failure
	if (OSCCONbits.OSWEN)						//
clock switch failure
	{
		LATBbits.LATB15 = LOW;		// PORTB15 low indicates clock
switch failure
		OSCCONbits.LSB = 0x46;				// Unlock sequence
to allow write to
		OSCCONbits.LSB = 0x57;				// OSCCONL
system control register
		OSCCONbits.OSWEN = 0;		// abort clock switch
	}
	return;
}
/*------------------------------------------------------------------------*/
```

Filename: initTimer2.c:

```
#include "p33FJ128GP708.h"
#include "dau_GP.h"
#include <math.h>

/*------------------------------------------------------------------
 Function Name: initTimer2
 Description:   Initializes Timer2 for desired sample rate
 Inputs:        None
 Returns:        None
 Ex. call:                initTimer2();
 Note:
------------------------------------------------------------------*/
void initTimer2(void)
{

        /* ensure Timer 2 is off and in 16-bit counter mode with 1:1 prescaler and
source = Fcy*/
        T2CON = 0x0000;

        /* reset Timer 2 interrupt flag */
        IFS0bits.T2IF = 0;

        /* set Timer2 interrupt priority level to 4 */
        IPC1bits.T2IP = 4;

        /* enable Timer 2 interrupt */
        IEC0bits.T2IE = 1;

        /* set Timer 2 period register */
//      PR2 = 0x5014;                  // 512.5 microsec period and Fref = 489 Hz
//      PR2 = 0x4010;                  // 410 microsec period and Fref = 610 Hz
//      PR2 = 0x3340;                  // 328 microsec period and Fref = 762 Hz

        // Desired interrupt rate = 1 MHz
        // Fcy = 40E6
        // Clock divisor = 40E6/1E6 = 40 = 0x0028
        //PR2 = 0x0028; // Result is 0.97 Mhz interrupt and 485 kHz clock
        PR2 = 0x0026; // Result is 1.024 MHz interrupt and 512 kHz clock

        return;
}
```

Filename: isrTimer2.c:

```
#include <p33FJ128GP708.h>
#include "dau_GP.h"
#include <delay.h>
//unsigned char flag;
/*------------------------------------------------------------------
 Function Name: T2Interrupt
 Description:   Timer2 Interrupt Handler
 Inputs:        None
 Returns:       None
------------------------------------------------------------------*/
void __attribute__((__interrupt__, no_auto_psv)) _T2Interrupt ( void )
{

        if (flagClock == 1)
        {
                flagClock = 2;
                TESTLED2 = HIGH;
        }
        else
        {
                flagClock = 1;
                TESTLED2 = LOW;
        }

        /* reset Timer 2 interrupt flag */
        IFS0bits.T2IF = 0;

}
```

Filename: dau_GP.h:

```
//dau_GP.h
/*
Note:   This is the header file that contains definitions of pins
                and declarations of prototypes of functions used for data
                collection using the DAU daughter board.

                Pins/Signals ending with '_' are active low signals.

                Include this file before calling the functions and/or using
                the DAU pins
*/
```

```
#ifndef __DAU_GP_H__
#define __DAU_GP_H__

//Control signals definitions using LAT registers

// Delta Sigma, LED control pin definitions
//c #define RESET_                        LATGbits.LATG1
//c #define START_A                       LATFbits.LATF1
//c #define START_B                       LATFbits.LATF0
//c #define CS_EN                 LATGbits.LATG0
//c #define CS_C                  LATGbits.LATG12
//c #define CS_B                  LATGbits.LATG13
//c #define CS_A                  LATGbits.LATG14

#define SYNC_                   LATGbits.LATG0
#define DREADY_                 PORTAbits.RA2
#define LED1                    LATGbits.LATG15
#define LED2                    LATAbits.LATA3
#define TESTLED1        LATFbits.LATF1
#define TESTLED2        LATFbits.LATF0

//Digital buffer enable pins
#define BUFFEN1_ LATAbits.LATA6
#define BUFFEN2_ LATAbits.LATA7

//1:10 Clock driver bank output enable pins
#define G1              LATDbits.LATD8
#define     G2                  LATDbits.LATD9

//Clock oscillator enable pin
#define EOH             LATDbits.LATD15

//USB-FIFO control pins
#define USB_RD_         LATDbits.LATD10
#define USB_WR          LATDbits.LATD11
#define USB_TXE_ LATDbits.LATD12
#define USB_RXF_ LATDbits.LATD13

//Signal level definitions
#define HIGH     1
#define LOW             0

//Global variable prototypes
```

```
extern unsigned int flagClock, ainSelect, ledFlag, dataSet;     // Flags
extern unsigned int transFlag;                                  // Flag to
indicate presence of transient
extern long transLevel;                                              //
Changes above this value are transients
extern unsigned long chtemp;                                    // Temp
variable used in integration
extern long led1_in[32],led1_b1[32],led1_out[32];       // Channel data
extern long led2_in[32],led2_b1[32],led2_out[32];       // Channel data
extern long adc_temp[8];                                        //
Temperature readings from adcs
extern int raw_led[96];                                              // Raw
channel data from the ADCs
extern unsigned long y_filter[32];
extern unsigned int sampleCtr, writeCtr, recordFlag, usb1Flag;
extern unsigned char regdata[8][8];

extern long unfilteredSamples[64];
extern int unfilteredCtr;
extern int startupFlag, startupDelay, phaseCtr;

//Function prototypes
extern void clockSwitchC (void);
extern void powerUp (void);
extern void initADCs (void);
extern void configADCs (void);
extern void configTemp (void);

extern void initTimer1 (void);
//extern void _ISR _T1Interrupt (void);

extern void initTimer2 (void);
//extern void _ISR _T2Interrupt (void);

//extern void initTimer3 (void);
//extern void _ISR _T3Interrupt (void);

extern void initSpi1 (unsigned char bitMode);
extern void initSpi2 (void);
extern void accumOff1 (void);
extern void accumOn1 (void);
extern void accumOff2 (void);
extern void accumOn2 (void);
extern void clearData (void);
extern void getData (void);
extern void getData1 (void);
extern void getData2 (void);
extern void getData3 (void);
extern void getData4 (void);
extern void getTemp (void);
extern void usbWritedata1Off (void);
extern void usbWritedata1On (void);
extern void usbWritedata2Off (void);
extern void usbWritedata2On (void);
extern void usbWritetransient (void);
extern void usbWritetranslevel (void);
extern void usbWriteTemp (void);
extern void setupTransient (void);

#endif
```

APPENDIX E

PC SOFTWARE FOR THE ADS1278 DAU

Visual Basic 2006 Form Software:

```
Dim ChannelCount As Integer
Dim ChannelValueLo As Long
Dim ChannelValueHi As Long
Dim ByteCount As Integer
Dim PreambleCount As Integer
Dim Startup As Integer
Dim ReadCommand As Integer
Dim dataPoints As Long
Dim minVoltage As Single
Dim maxVoltage As Single
Dim voltageDiff As Single
Dim recordFlag As Byte
Dim recordedBytes As Long
Dim fileLocation As String
Dim OutputString As String
Dim LIAOutput(66) As Double
Dim LIAOutputOld(64) As Double
Dim LIAFiltered(64) As Double
Dim NewPinout(66) As Integer
Dim OldPinout(66) As Integer
Dim FilterFactor As Double
Dim TempDouble As Double
Dim OnDataFlag As Byte
Dim PinoutFlag As Byte
Dim part1 As Integer
Dim part2 As Integer
Dim part3 As Integer
Dim part4 As Integer
Dim DataRecordNumber As Integer
Dim DataRecordCount As Integer
Dim TransientFlag As Integer


Private Sub Command1_Click()

  minVoltage = CSng(Text6.Text)
  maxVoltage = CSng(Text5.Text)
```

```
    voltageDiff = maxVoltage - minVoltage

    If Combo2.ListIndex = 0 Then
        FilterFactor = 1024
    Else
        If Combo2.ListIndex = 1 Then
            FilterFactor = 512
        Else
            If Combo2.ListIndex = 2 Then
                FilterFactor = 256
            Else
                If Combo2.ListIndex = 3 Then
                    FilterFactor = 128
                Else
                    If Combo2.ListIndex = 4 Then
                        FilterFactor = 64
                    Else
                        If Combo2.ListIndex = 5 Then
                            FilterFactor = 32
                        Else
                            If Combo2.ListIndex = 6 Then
                                FilterFactor = 16
                            Else
                                If Combo2.ListIndex = 7 Then
                                    FilterFactor = 8
                                Else
                                    If Combo2.ListIndex = 8 Then
                                        FilterFactor = 4
                                    Else
                                        FilterFactor = 2
                                    End If
                                End If
                            End If
                        End If
                    End If
                End If
            End If
        End If
    End If

    PinoutFlag = Combo3.ListIndex
    DataRecordNumber = CInt(Text2.Text)

End Sub
```

```
Private Sub Command2_Click()

   If recordFlag = 0 Then
      'Open Text3.Text For Output As #1
      recordFlag = 1
      recordedBytes = 0
      Command2.BackColor = &H8080FF
      Command2.Caption = "Stop Recording"
      'Print #1, "0  0x00"
   Else
      recordFlag = 0
      Command2.BackColor = &H80FF80
      Command2.Caption = "Record Data"
      'Close #1
   End If

End Sub

Private Sub Command3_Click()

   Dim PortSel As Integer

   On Error GoTo EH

   'If the port is already open, close it
   If MSComm1.PortOpen = True Then
      MSComm1.PortOpen = False
      Command3.Caption = "Connect!"
   Else
      PortSel = Combo1.ListIndex

      ' Fire Rx Event Every Byte
      MSComm1.RThreshold = 1

      ' When Inputting Data, Input 1 Byte at a time
      MSComm1.InputLen = 1

      ' 921600 Baud, No Parity, 8 Data Bits, 1 Stop Bit
      MSComm1.Settings = "921600,N,8,1"

      ' Disable DTR
      MSComm1.DTREnable = True
```

```
        ' Open COM Port
        MSComm1.CommPort = PortSel
        MSComm1.PortOpen = True

        Command3.Caption = "Disconnect!"
    End If
    Exit Sub

EH:
    MsgBox "ERROR: " & Err.Description & ". [Error #" & Err.Number & "]",
vbCritical
End Sub

Private Sub Command5_Click()
    Dim tString As String
    Dim String2 As String

    On Error GoTo EH2

    String2 = CStr(Combo4.ListIndex)
    tString = "HLIMIT" + String2

    For i = 1 To 7
        Do While MSComm1.OutBufferCount > 500
        Loop
        MSComm1.Output = Mid(tString, i, 1)
    Next

Exit Sub
EH2:
    MsgBox "ERROR: " & Err.Description & ". [Error #" & Err.Number & "]",
vbCritical

End Sub

Private Sub Form_Load()

    Dim Counter, i As Integer
    Dim TestFile As String
    Dim lhigh As Long
    Dim llow As Long
    Dim lhighorig As Long
    Dim dtemp1 As Double
    Dim dtemp2 As Double
    Dim dtemp3 As Double
    Dim llarge As Long
    Dim templong As Long
    Dim temp As Long
    Dim larged As Double

    For Port = 0 To 16
        Combo1.AddItem "COM" & Port, Port
    Next

    Combo2.AddItem "BW = 0.00054 Hz, TC= 295 s", 0
    Combo2.AddItem "BW = 0.00108 Hz, TC= 147 s", 1
    Combo2.AddItem "BW = 0.0022 Hz, TC= 72.3 s", 2
    Combo2.AddItem "BW = 0.0043 Hz, TC= 37.0 s", 3
    Combo2.AddItem "BW = 0.0087 Hz, TC= 18.3 s", 4
    Combo2.AddItem "BW = 0.0174 Hz, TC= 9.1 s", 5
    Combo2.AddItem "BW = 0.0340 Hz, TC= 4.7 s", 6
    Combo2.AddItem "BW = 0.0620 Hz, TC= 2.6 s", 7
    Combo2.AddItem "BW = 0.1015 Hz, TC= 1.7 s", 8
    Combo2.AddItem "BW = 0.1475 Hz, TC= 1.1 s", 9

    Combo3.AddItem "New", 0
    Combo3.AddItem "Old+Adapter", 1

    Combo4.AddItem "2.5 Volts", 0
    Combo4.AddItem "1.25 Volts", 1
    Combo4.AddItem "0.625 Volts", 2
    Combo4.AddItem "0.313 Volts", 3
    Combo4.AddItem "0.156 Volts", 4
    Combo4.AddItem "0.078 Volts", 5
    Combo4.AddItem "0.039 Volts", 6
    Combo4.AddItem "0.020 Volts", 7
    Combo4.AddItem "0.010 Volts", 8
    Combo4.AddItem "0.005 Volts", 9

    PreambleCount = 0
    Startup = 1
    OnDataFlag = 1
    ChannelCount = 0
    ChannelValue = 0
    Combo2.ListIndex = 0
    FilterFactor = 1024
    Combo3.ListIndex = 1
    Combo4.ListIndex = 0
```

```
PinoutFlag = 1
TransientFlag = 0

'LIAFlag is set to 0 for lock in amp
LIAFlag = 0

'Init Recording Info
recordedBytes = 0

'Text2.Text = recordedBytes
'Take care of scaling
minVoltage = 0
maxVoltage = 0.05
Text5.Text = maxVoltage
Text6.Text = minVoltage
voltageDiff = maxVoltage - minVoltage

'Init recordFlag to off until button is pressed
recordFlag = 0
fileLocation = "C:\data.raw"
Text3.Text = fileLocation

For Counter = 1 To 64
   LIAOutput(Counter) = 0
   LIAOutputOld(Counter) = 0
   LIAFiltered(Counter) = 0
Next

DataRecordNumber = 1 ' Record one data set every DataRecordNumber sets
Text2.Text = CStr(DataRecordNumber)
DataRecordCount = 1

' Populate pinout arrays

NewPinout(1) = 27
NewPinout(2) = 28
NewPinout(3) = 25
NewPinout(4) = 26
NewPinout(5) = 19
NewPinout(6) = 20
NewPinout(7) = 17
NewPinout(8) = 18
NewPinout(9) = 11
NewPinout(10) = 12
NewPinout(11) = 9
NewPinout(12) = 10
NewPinout(13) = 3
NewPinout(14) = 4
NewPinout(15) = 1
NewPinout(16) = 2
NewPinout(17) = 22
NewPinout(18) = 21
NewPinout(19) = 24
NewPinout(20) = 23
NewPinout(21) = 30
NewPinout(22) = 29
NewPinout(23) = 32
NewPinout(24) = 31
NewPinout(25) = 6
NewPinout(26) = 5
NewPinout(27) = 8
NewPinout(28) = 7
NewPinout(29) = 14
NewPinout(30) = 13
NewPinout(31) = 16
NewPinout(32) = 15

NewPinout(33) = 27 + 32
NewPinout(34) = 28 + 32
NewPinout(35) = 25 + 32
NewPinout(36) = 26 + 32
NewPinout(37) = 19 + 32
NewPinout(38) = 20 + 32
NewPinout(39) = 17 + 32
NewPinout(40) = 18 + 32
NewPinout(41) = 11 + 32
NewPinout(42) = 12 + 32
NewPinout(43) = 9 + 32
NewPinout(44) = 10 + 32
NewPinout(45) = 3 + 32
NewPinout(46) = 4 + 32
NewPinout(47) = 1 + 32
NewPinout(48) = 2 + 32
NewPinout(49) = 22 + 32
NewPinout(50) = 21 + 32
NewPinout(51) = 24 + 32
NewPinout(52) = 23 + 32
NewPinout(53) = 30 + 32
NewPinout(54) = 29 + 32
NewPinout(55) = 32 + 32
NewPinout(56) = 31 + 32
NewPinout(57) = 6 + 32
NewPinout(58) = 5 + 32
NewPinout(59) = 8 + 32
NewPinout(60) = 7 + 32
NewPinout(61) = 14 + 32
NewPinout(62) = 13 + 32
NewPinout(63) = 16 + 32
NewPinout(64) = 15 + 32
NewPinout(65) = 65
NewPinout(66) = 66

OldPinout(1) = 29
OldPinout(2) = 31
OldPinout(3) = 25
OldPinout(4) = 27
OldPinout(5) = 21
OldPinout(6) = 23
OldPinout(7) = 17
OldPinout(8) = 19
OldPinout(9) = 13
OldPinout(10) = 15
OldPinout(11) = 9
OldPinout(12) = 11
OldPinout(13) = 5
OldPinout(14) = 7
OldPinout(15) = 1
OldPinout(16) = 3
OldPinout(17) = 20
OldPinout(18) = 18
OldPinout(19) = 24
OldPinout(20) = 22
OldPinout(21) = 28
OldPinout(22) = 26
OldPinout(23) = 32
OldPinout(24) = 30
OldPinout(25) = 4
OldPinout(26) = 2
OldPinout(27) = 8
OldPinout(28) = 6
OldPinout(29) = 12
OldPinout(30) = 10
```

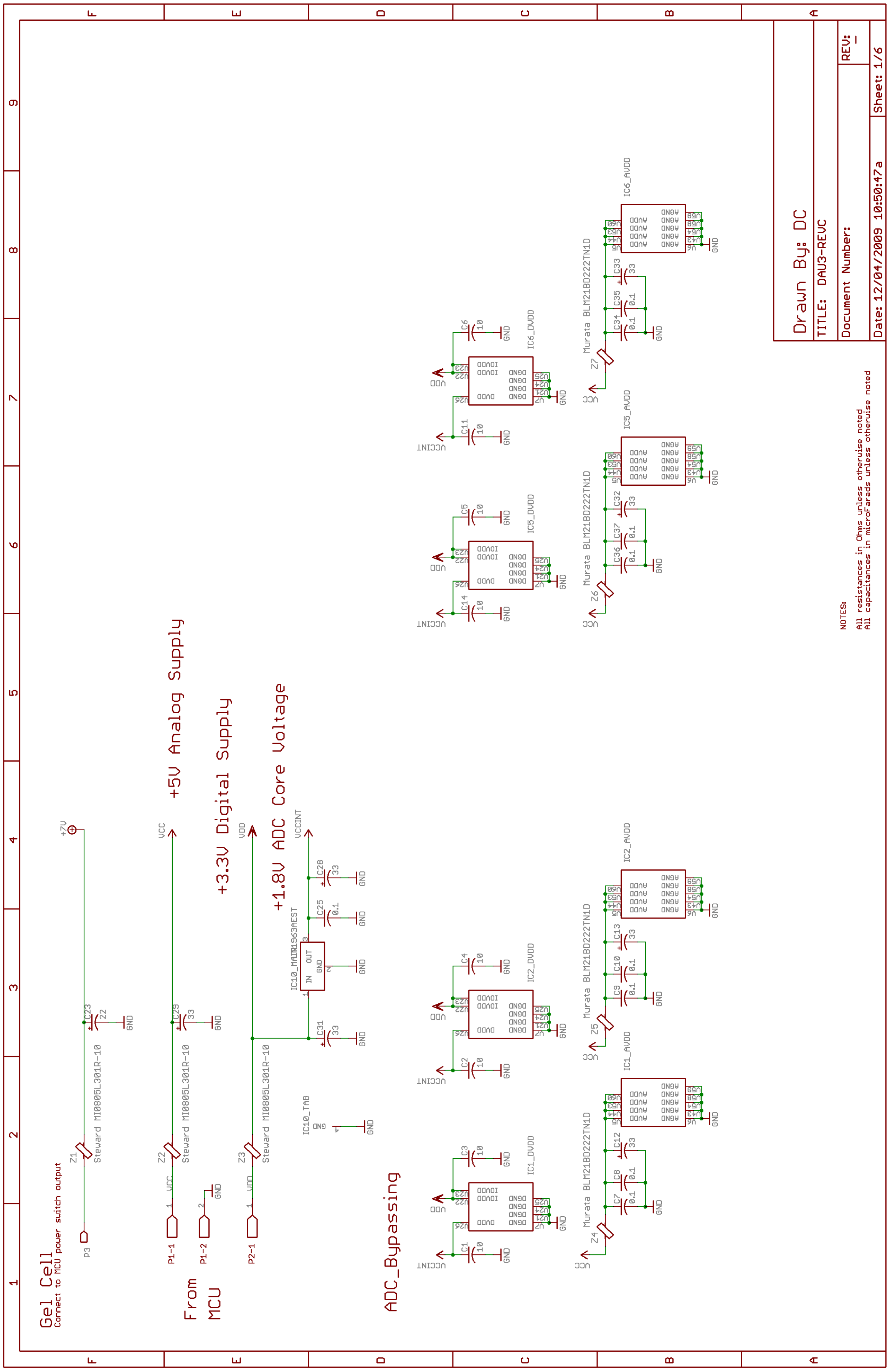
```
    OldPinout(31) = 16
    OldPinout(32) = 14

    OldPinout(33) = 29 + 32
    OldPinout(34) = 31 + 32
    OldPinout(35) = 25 + 32
    OldPinout(36) = 27 + 32
    OldPinout(37) = 21 + 32
    OldPinout(38) = 23 + 32
    OldPinout(39) = 17 + 32
    OldPinout(40) = 19 + 32
    OldPinout(41) = 13 + 32
    OldPinout(42) = 15 + 32
    OldPinout(43) = 9 + 32
    OldPinout(44) = 11 + 32
    OldPinout(45) = 5 + 32
    OldPinout(46) = 7 + 32
    OldPinout(47) = 1 + 32
    OldPinout(48) = 3 + 32
    OldPinout(49) = 20 + 32
    OldPinout(50) = 18 + 32
    OldPinout(51) = 24 + 32
    OldPinout(52) = 22 + 32
    OldPinout(53) = 28 + 32
    OldPinout(54) = 26 + 32
    OldPinout(55) = 32 + 32
    OldPinout(56) = 30 + 32
    OldPinout(57) = 4 + 32
    OldPinout(58) = 2 + 32
    OldPinout(59) = 8 + 32
    OldPinout(60) = 6 + 32
    OldPinout(61) = 12 + 32
    OldPinout(62) = 10 + 32
    OldPinout(63) = 16 + 32
    OldPinout(64) = 14 + 32
    OldPinout(65) = 65
    OldPinout(66) = 66

End Sub


Private Sub MSComm1_OnComm()
Dim cData As String             ' Holds our incoming data
Dim bData As Byte              ' Holds our converted data
Dim NewOnDataFlag As Byte       ' Flag to store whether data is LED1 or 2 on or
off
Dim NewChannelCount As Integer   ' Which channel number gets updated
Dim exponent As Integer         ' Used for the exponent
Dim temp As Double             ' Temp number for 2 ^ exponent
Dim temp2 As Long
Dim i As Integer

Dim tempChannelValue As Long
Dim hex_string As String        ' Used to print hex value to file

Dim valuedbl As Double
Dim tempdbl As Double
Dim largedbl As Double
Dim ChannelValueHiOrig As Long

  ' If comEvReceive Event then get data and display

  If Startup = 1 Then
    ' Read byte from USB
    If MSComm1.CommEvent = comEvReceive Then
      cData = MSComm1.Input            ' Get data
      ' If byte has value of 0xFF, increase preamble count, if not, set preamble
count to zero
      bData = Asc(cData)
      If bData = 255 Then
        PreambleCount = PreambleCount + 1
        ' If we have 4 bytes of 255, preamble has been sent
        If PreambleCount = 4 Then
          Startup = 0
          PreambleCount = 0
        End If
      Else
        PreambleCount = 0
      End If
    End If

  Else
    If MSComm1.CommEvent = comEvReceive Then
      cData = MSComm1.Input            ' Get data
      ByteCount = ByteCount + 1         ' Increment ByteCount
      bData = Asc(cData)
      If recordFlag = 1 Then
```

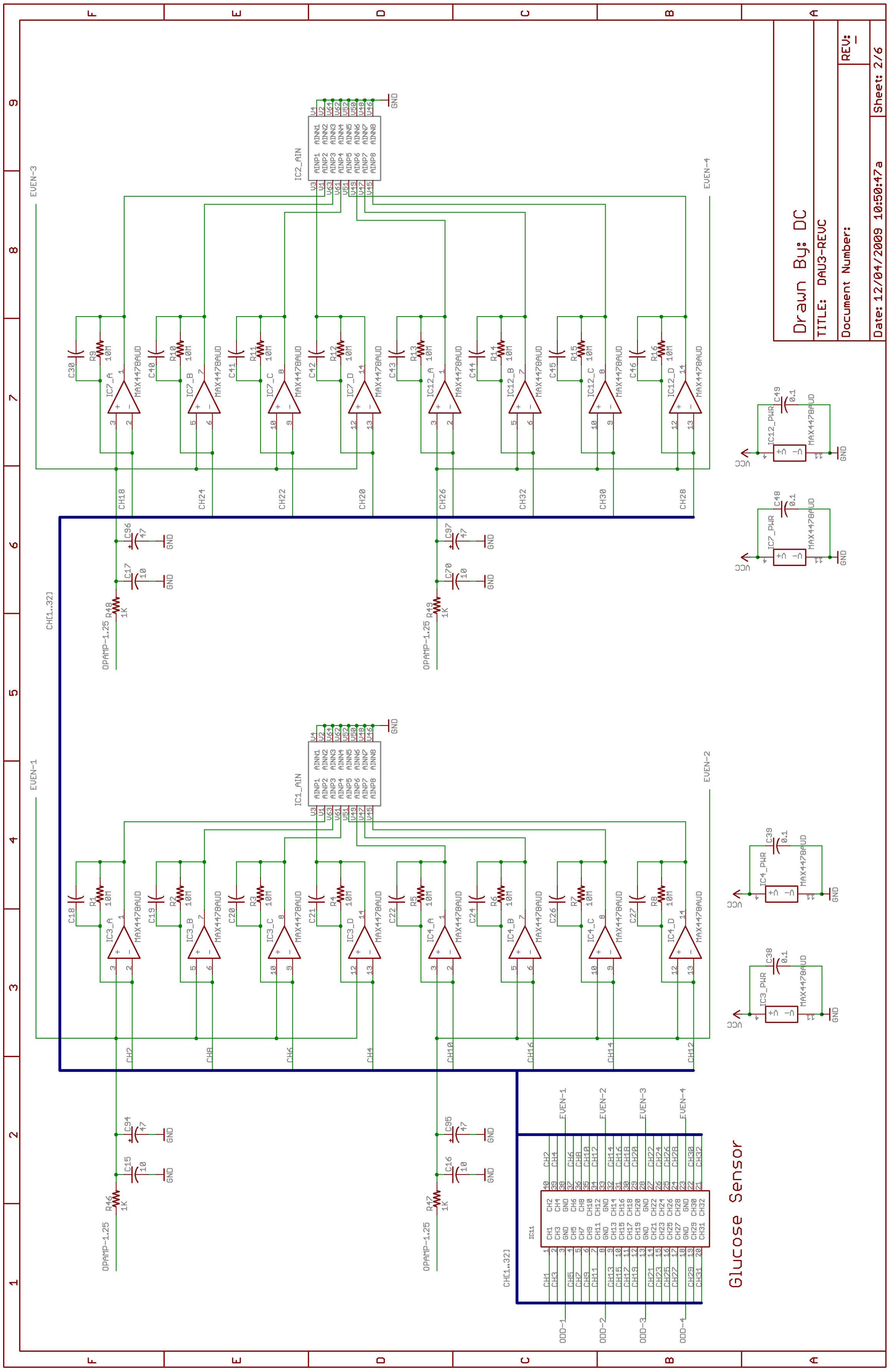
```
            recordedBytes = recordedBytes + 1   ' Increment recordedBytes
         End If

         ' Data bytes are read in least significant byte first
         If ByteCount = 1 Then
            exponent = 0
            part1 = bData
         End If
         If ByteCount = 2 Then
            exponent = 8
            part2 = bData
         End If
         If ByteCount = 3 Then
            exponent = 16
            part3 = bData
         End If
         If ByteCount = 4 Then
            exponent = 24
            part4 = bData
         End If

         If ByteCount = 4 Then

            ChannelValueLo = part2 * (2 ^ 8) + part1
            ChannelValueHi = part4 * (2 ^ 8) + part3
            ChannelCount = ChannelCount + 1

            'Make the update to the appropriate bar
            Call UpdateSize(ChannelCount, ChannelValueLo, ChannelValueHi,
PinoutFlag, 1)

            'Reset Variables
            ByteCount = 0
            If ChannelCount = 66 Then
               ChannelCount = 0
               Startup = 1
            End If
         End If
      End If
   End If

End Sub

Private Sub UpdateSize(Channel As Integer, IntensityLo As Long, IntensityHi As
Long, Pinout As Byte, Filtered As Byte)
   Dim Index As Integer
   Dim Tempheight As Long
   Dim VoltValue As Double
   Dim tempD As Double
   Dim SaveFileName As String
   Dim qTime As Double
   Dim iTime As Long
   Dim templong As Long
   Dim msActual As Integer
   Dim msString As String
   Dim high As Integer
   Dim low As Integer
   Dim sum As Long
   Dim value As Double
   Dim tempdbl As Double
   Dim large As Double
   Dim large2 As Double
   Dim dummy As Integer
   Dim IntensityHiOrig As Long
   Dim ActualChannel As Integer

   large = 2147483392#

   If (Channel < 66) Then
      ' Translate sample into real number
      ' First determine if number is negative
      ' If number is negative calculate twos complement
      IntensityHiOrig = IntensityHi

      If ((IntensityHi And &H8000) = 32768) Then
         'Number is negative so calculate twos complement
         IntensityLo = IntensityLo Xor 65535
         IntensityHi = IntensityHi Xor 65535
         IntensityLo = IntensityLo + 1
         If IntensityLo > 65535 Then
            'Carry into IntensityHi
            IntensityLo = 0
            IntensityHi = IntensityHi + 1
         End If
         ' Translate to real number
         tempdbl = CDbl(IntensityHi / large)
         value = -1 * (CDbl(tempdbl * 65536#) + CDbl(IntensityLo / large))
         Else
         'Translate to real number
         'Number is positive and high bit of IntensityHi is zero
         'so the next line will not cause overflow
         value = CDbl((IntensityHi * 65536# + IntensityLo) / large)
      End If
      ' Vref is defined as difference betwee Vref+ and Vref- or 2.5VDC
      ' Also multiply by 2 since LIA filter output is magnitude of input not peak to
peak
      value = value * CDbl(5#)
   Else
      ' Data indicates status of transient detection
      If (IntensityLo > 0) Then
         value = 1#
      Else
         value = 0#
      End If
   End If

   ActualChannel = 0
   ' Place data in correct location considering pinout and order that MCU reads
channel data
   If Pinout = 0 Then
         ActualChannel = NewPinout(Channel)
   Else
      If Pinout = 1 Then
         ActualChannel = OldPinout(Channel)
      End If
   End If

   LIAOutput(ActualChannel) = value

   ' If this is the last data for this spectrum - filter data and display values
   If Channel = 66 Then
      'Display LIA Data
      For voltCnt = 1 To 64
         ' Filter Samples - First order Butterworth
         LIAFiltered(voltCnt) = LIAFiltered(voltCnt) * (CDbl(FilterFactor) - 2#) /
CDbl(FilterFactor)
         LIAFiltered(voltCnt) = LIAFiltered(voltCnt) + (LIAOutputOld(voltCnt) +
LIAOutput(voltCnt)) / CDbl(FilterFactor)
         LIAOutputOld(voltCnt) = LIAOutput(voltCnt)
         VoltValue = LIAFiltered(voltCnt)

         Index = voltCnt - 1
         Tempheight = CLng(4815 - CLng(VoltValue * CDbl(4815 / voltageDiff)) +
CLng(minVoltage * CDbl(4815 / voltageDiff)))
         If Tempheight < 0 Then
           Tempheight = 0
         End If
         If Tempheight > 4815 Then
           Tempheight = 4815
         End If
         If (voltCnt < 33) Then
           Shape3(Index).Height = Tempheight
           Shape1(Index).Height = 4815 - Tempheight
           Shape1(Index).Top = 600 + Tempheight
         Else
           Shape10(Index - 32).Height = Tempheight
           Shape7(Index - 32).Height = 4815 - Tempheight
           Shape7(Index - 32).Top = 600 + Tempheight
         End If
      Next

      If LIAOutput(66) > 0 Then
         Command4.BackColor = &H8080FF
         Command4.Caption = "Transient"
         TransientFlag = 1
      Else
         Command4.BackColor = &H80FF80
         Command4.Caption = "Stable"
         TransientFlag = 0
      End If

      If recordFlag = 1 Then

         'Clear the output string
         OutputString = ""
         'Open up the file for writing
         SaveFileName = "" + Text3.Text
         Open SaveFileName For Append As #1
         'Populate the output string
         For voltCnt = 1 To 64
           OutputString = OutputString + " " + CStr(LIAFiltered(voltCnt))
         Next
         OutputString = OutputString + " " + CStr(LIAOutput(65))
         OutputString = OutputString + " " + CStr(LIAOutput(66))

         qTime = Timer
```

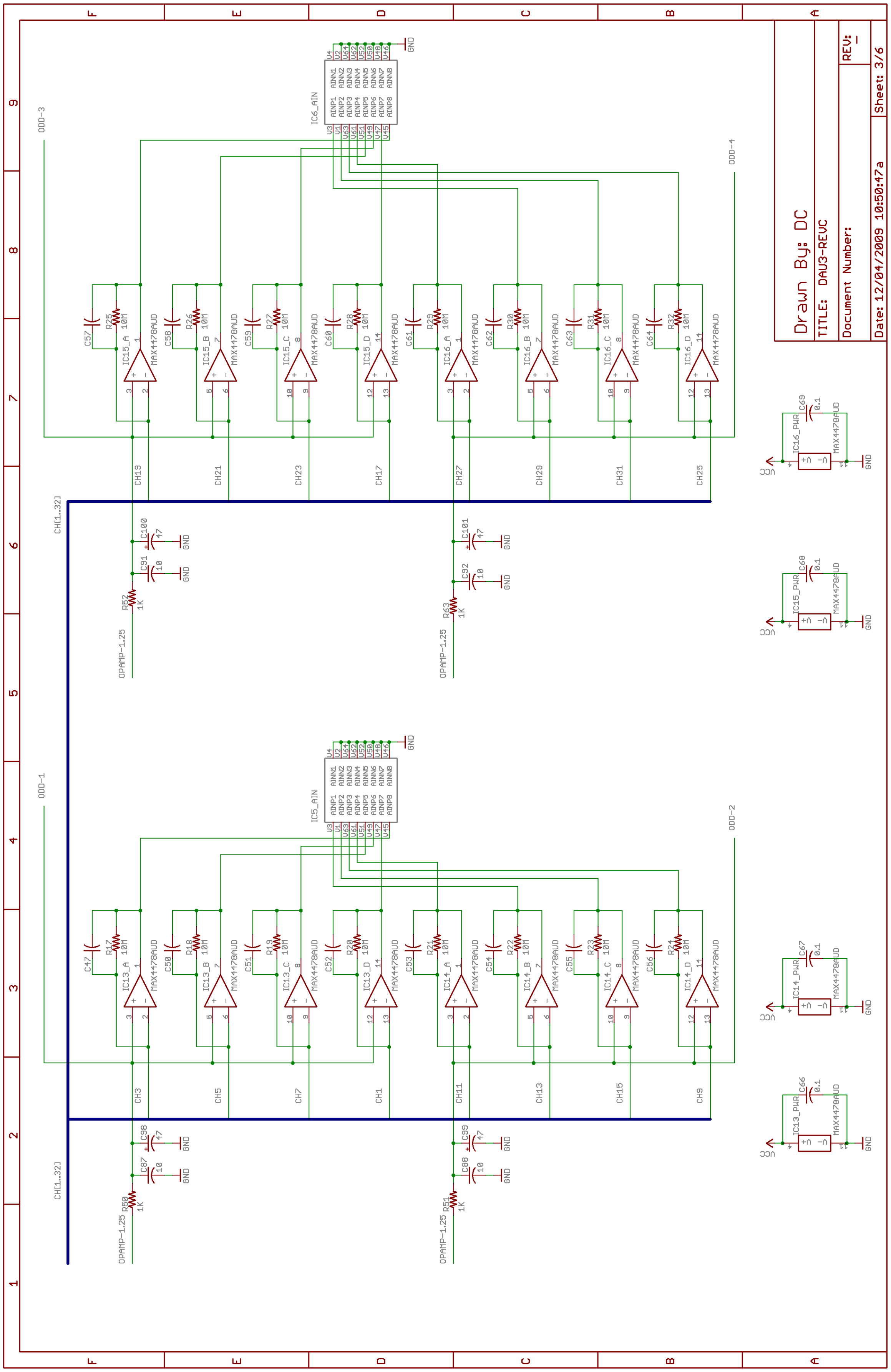
```
         iTime = Int(qTime)
         msActual = CInt((qTime - CDbl(iTime)) * 1000)
         msString = Format(msActual, "000")
         If DataRecordCount = DataRecordNumber Then
            Print #1, "" + CStr(Year(DateTime.Date)) + " " +
CStr(Month(DateTime.Date)) + " " + CStr(Day(DateTime.Date)) + " " +
CStr(Hour(DateTime.Time)) + " " + CStr(Minute(DateTime.Time)) + " " +
CStr(Second(DateTime.Time)) + "." + msString + " " + OutputString
            DataRecordCount = 0
         End If
         DataRecordCount = DataRecordCount + 1
         'Close the file
         Close #1


      End If

   End If

End Sub
```

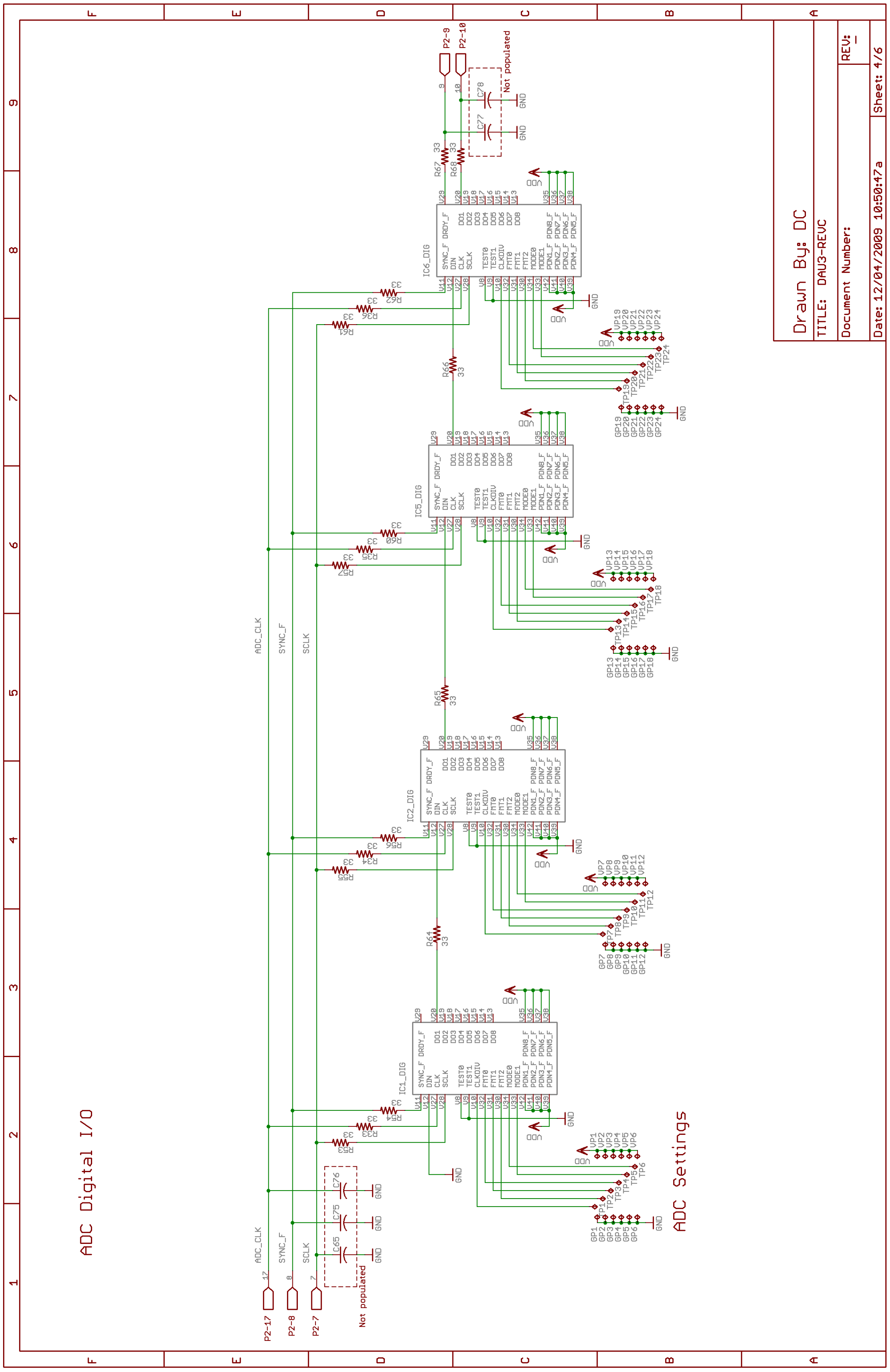
APPENDIX F

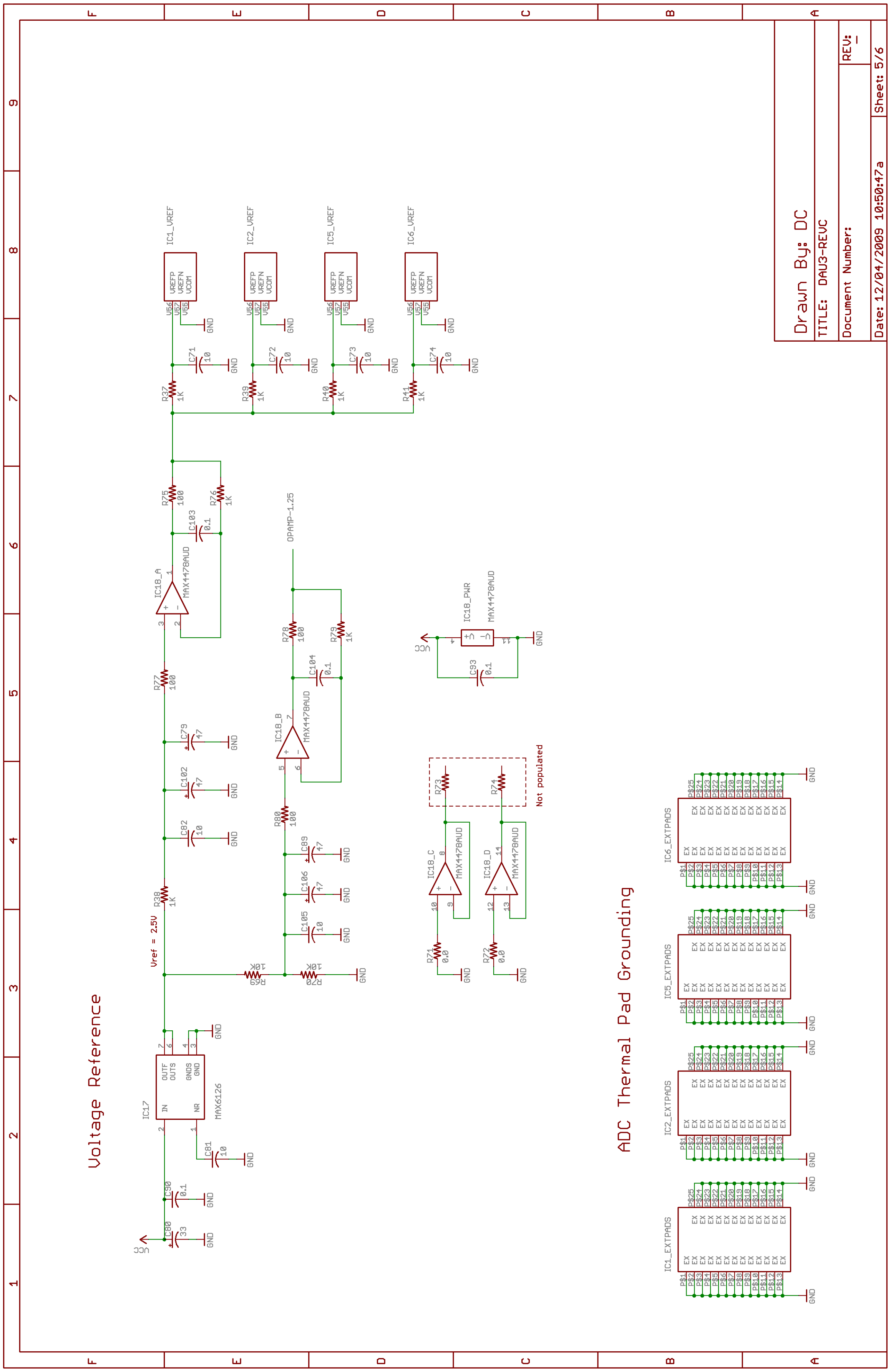
ADS1278 DAU SCHEMATIC DIAGRAM

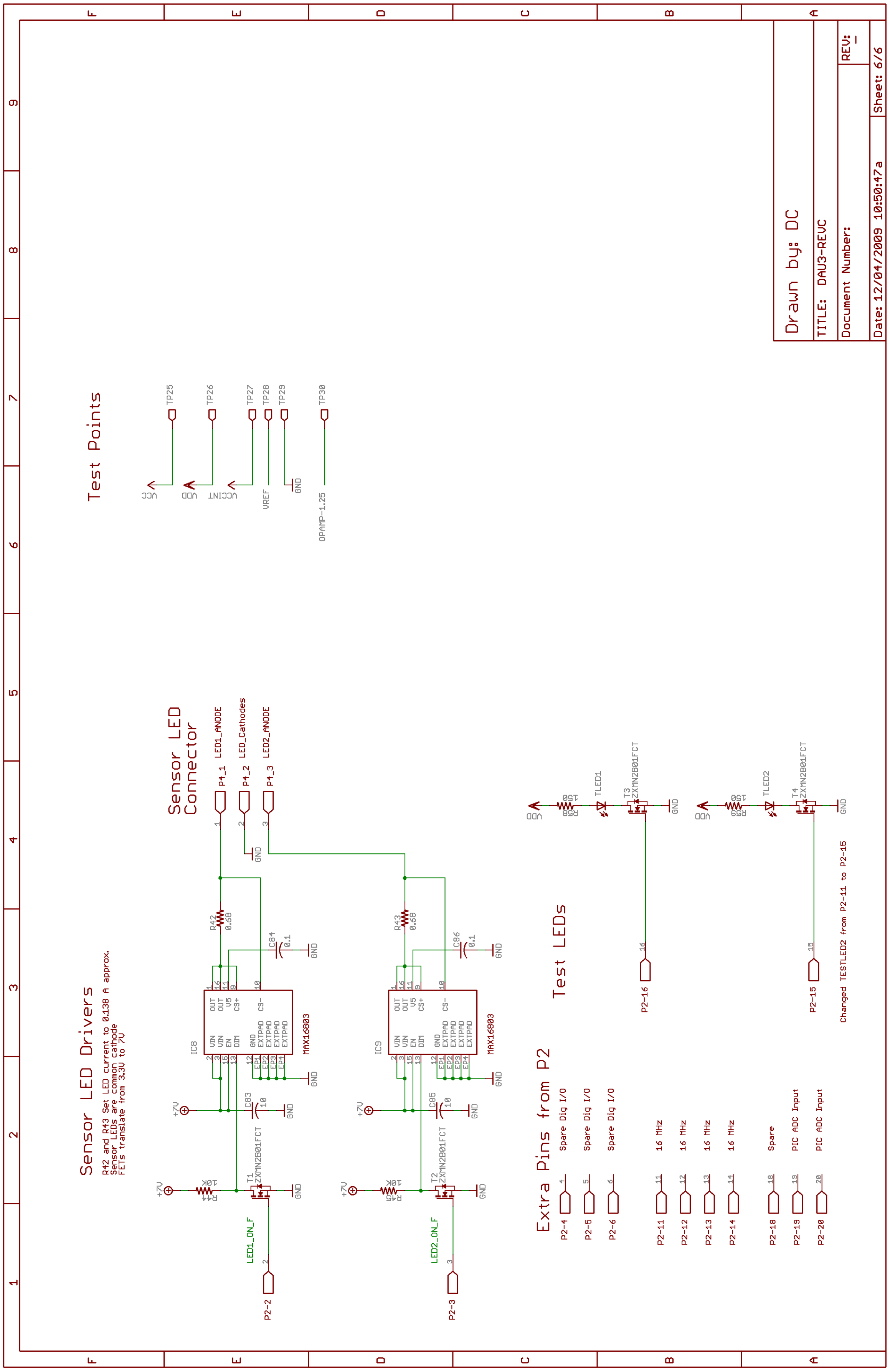

Gel Cell
Connect to MCU power switch output
From MCU
ADC_Bypassing
+5V Analog Supply
+3.3V Digital Supply
+1.8V ADC Core Voltage
+7V
VCC
VDD
VCCINT
Steward MI0805L301R-10
Murata BLM21BD222TN1D
IC10_MAX1963AEST
IC10_TAB
IC1_DVDD
IC2_DVDD
IC5_DVDD
IC6_DVDD
IC1_AVDD
IC2_AVDD
IC5_AVDD
IC6_AVDD
GND
NOTES:
All resistances in Ohms unless otherwise noted
All capacitances in microFarads unless otherwise noted
Drawn By: DC
TITLE: DAU3-REVC
Document Number:
REV: _
Date: 12/04/2009 10:50:47a
Sheet: 1/6


Drawn By: DC
TITLE: DAU3-REVC
Document Number:
REV: _
Date: 12/04/2009 10:50:47a
Sheet: 2/6
Glucose Sensor
IC1_AIN
IC2_AIN
MAX4478AUD
CH[1..32]
OPAMP-1.25
EVEN-1
EVEN-2
EVEN-3
EVEN-4
ODD-1
ODD-2
ODD-3
ODD-4
GND


Drawn By: DC
TITLE: DAU3-REVC
Document Number:
REV: _
Date: 12/04/2009 10:50:47a
Sheet: 3/6
IC5_AIN
IC6_AIN
MAX4478AUD
CH[1..32]
OPAMP-1.25
ODD-1
ODD-2
ODD-3
ODD-4
GND
VCC


ADC Digital I/O
ADC Settings
ADC_CLK
SYNC_F
SCLK
P2-17
P2-8
P2-7
P2-9
P2-10
Not populated
IC1_DIG
IC2_DIG
IC5_DIG
IC6_DIG
VDD
GND
Drawn By: DC
TITLE: DAU3-REVC
Document Number:
REV: _
Date: 12/04/2009 10:50:47a
Sheet: 4/6


Voltage Reference
Vref = 2.5V
IC17
MAX6126
IC18_A
IC18_B
IC18_C
IC18_D
IC18_PWR
MAX4478AUD
OPAMP-1.25
IC1_VREF
IC2_VREF
IC5_VREF
IC6_VREF
VREFP
VREFN
VCOM
Not populated
ADC Thermal Pad Grounding
IC1_EXTPADS
IC2_EXTPADS
IC5_EXTPADS
IC6_EXTPADS
GND
Drawn By: DC
TITLE: DAU3-REVC
Document Number:
REV: _
Date: 12/04/2009 10:50:47a
Sheet: 5/6


Sensor LED Drivers
R42 and R43 Set LED current to 0.138 A approx.
Sensor LEDs are common cathode
FETs translate from 3.3V to 7V
Sensor LED Connector
LED1_ANODE
LED_Cathodes
LED2_ANODE
P4_1
P4_2
P4_3
Test Points
TP25
TP26
TP27
TP28
TP29
TP30
VCC
VDD
VCCINT
VREF
GND
OPAMP-1.25
IC8
IC9
MAX16803
R42 0.68
R43 0.68
C83 10
C84 0.1
C85 10
C86 0.1
R44 10K
R45 10K
T1 ZXMN2B01FCT
T2 ZXMN2B01FCT
T3 ZXMN2B01FCT
T4 ZXMN2B01FCT
+7V
LED1_ON_F
LED2_ON_F
P2-2
P2-3
Test LEDs
TLED1
TLED2
R58 150
R59 150
P2-16
P2-15
Changed TESTLED2 from P2-11 to P2-15
Extra Pins from P2
P2-4 Spare Dig I/O
P2-5 Spare Dig I/O
P2-6 Spare Dig I/O
P2-11 16 MHz
P2-12 16 MHz
P2-13 16 MHz
P2-14 16 MHz
P2-18 Spare
P2-19 PIC ADC Input
P2-20 PIC ADC Input
Drawn by: DC
TITLE: DAU3-REVC
Document Number:
REV: _
Date: 12/04/2009 10:50:47a
Sheet: 6/6

APPENDIX G

CORRELATION COEFFICIENT SOFTWARE

Filename: pcorr.c.

```
#include <stdio.h>
#include <math.h>
#define MAX_DATA        1000

int main(int argc, char *argv[])
{

/* Feb. 22, 2012                                */
/* Calculates Pearson correlation coefficient          */
/* Input file consists of 1002 rows of data.           */
/* First row is mean of each channel                   */
/* Second row is standard deviation of each channel    */
/* Remainder of rows are samples for all channels      */
/* Each row is Ch1 to Ch 32 separated by a space.      */
/* Output data file is 32 rows and columns             */
/* of Pearson correlation coefficients                 */

 int incnt, chcnt, chcnt2;
 double mean[32], stddev[32];
 double indata[32][MAX_DATA];
 double outdata[32][32];

 FILE *idfPtr; /* idfPtr = input.dat file pointer */
 FILE *odfPtr; /* odfPtr = output.dat file pointer */

/* Read samples from input data file */
 if ( ( idfPtr = fopen( "corr-in.txt", "r" ) ) == NULL )
   printf( "No input file!\n" );
 else
 {
   printf( "Reading input data\n" );
   for(incnt=0; incnt<32; incnt++)
   {
     fscanf( idfPtr, "%lf", &mean[incnt]);
   }
   for(incnt=0; incnt<32; incnt++)
   {
     fscanf( idfPtr, "%lf", &stddev[incnt]);
   }
   for(incnt=0; incnt<MAX_DATA; incnt++)
   {
     for(chcnt=0; chcnt<32; chcnt++)
     {
       fscanf( idfPtr, "%lf", &indata[chcnt][incnt]);
     }
   }
   fclose( idfPtr );
   printf( "Got the input data okay\n" );
 }

/* Open output file */
 odfPtr = fopen( "output.dat", "w" );

/* Calculate correlations */
 for(chcnt=0; chcnt<32; chcnt++)
 {
   for(chcnt2=0; chcnt2<32; chcnt2++)
   {
     /* Compute correlation */
     outdata[chcnt][chcnt2] = 0.0;
     for(incnt=0; incnt<MAX_DATA; incnt++)
     {
       outdata[chcnt][chcnt2] += (indata[chcnt][incnt]-mean[chcnt]) * \
       (indata[chcnt2][incnt]-mean[chcnt2]);
     }
     outdata[chcnt][chcnt2] = (outdata[chcnt][chcnt2])/ \
     ((MAX_DATA-1)*stddev[chcnt]*stddev[chcnt2]);
   }
 }

/* Print result to output file */
 for(chcnt=0; chcnt<32; chcnt++)
 {
   for(chcnt2=0; chcnt2<32; chcnt2++)
   {
     fprintf( odfPtr, "%e ", outdata[chcnt][chcnt2]);
   }
   fprintf(odfPtr, "\n");
 }

 fclose( odfPtr );
 return 0;
}
```