THAYER SCHOOL OF ENGINEERING

ON-BOARD BIOMETRIC RECOGNITION AND PHYSIOLOGICAL MEASUREMENTS FOR SHARED USE

Final Project Report

Bradley Ficko, Shadab Khan, Michael Nieskoski, and Roja Nunna

6/4/2012

Table of Contents

Introduction

Heart rate and SpO_2 are few of the widely first-response measured physiological parameters of a patient. This report summarizes our approach to develop a medical instrument for the measurement of these parameters. In addition to performing these physiological measurements, we wanted the device to be able to recognize the patient under test. Such a device can be very helpful in a shared household setting where a limited group of people can use it to record and log measurements of their heart rate and SpO2. These logs can then be accessed by a physician either on site or remotely for the patient feedback. This report first explains the techniques and circuits used to measure Heart Rate and SpO2. In the latter sections, we describe the methods used for biometric recognition. We used Bioimpedance as a potential parameter to classify and recognize a small group of patients under study.

Heart Rate and SpO_2 Measuremnt

Measuring the response of photodiode to the excitation of IR and R LEDs in an arrangement shown below has been typically used to measure the heart rate and SpO_2 .

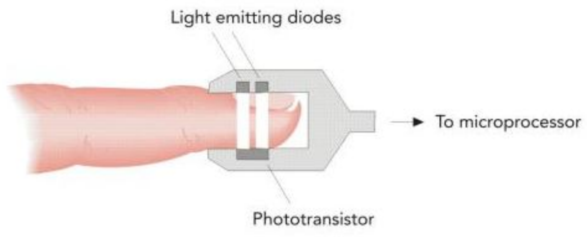


Figure 1 : General arrangement of finger-based oximetry, image reference : <u>www.medtek.ki.se</u>

Here is an image of the Nellcor Pulse Oximetry (PSOX) sensor use for our project.

Figure 2 : Nellcor Pulse Oximetry Sensor, Image courtesy www.nellcor.com

We begin with an overview of the block diagram of signal acquisition chain.

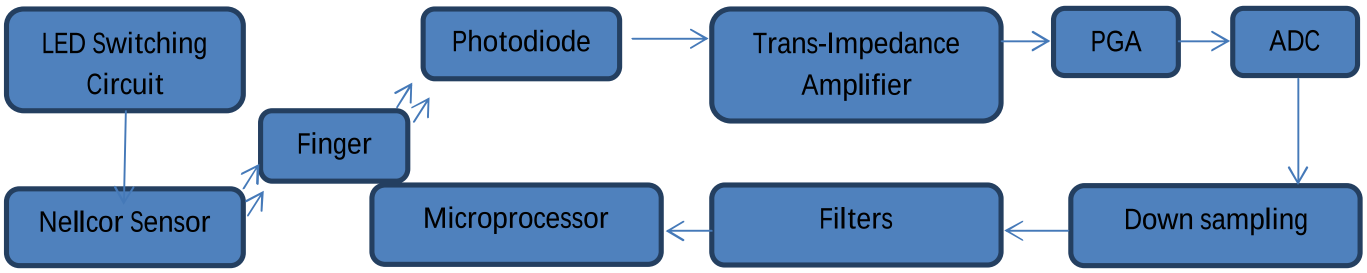


Figure i: Overview of signal flow for Heart Rate and SpO$_2$ measurements

LED Switching Circuit

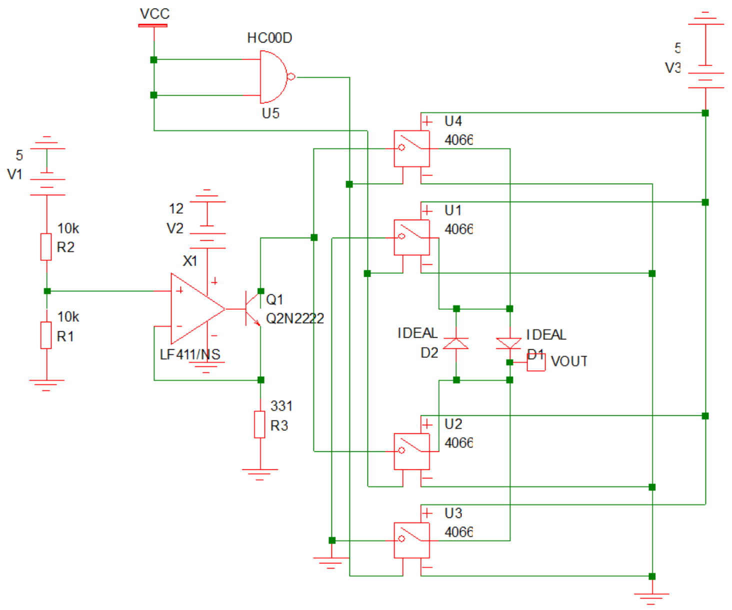


Figure 3: LED switching circuit to toggle between red and infrared LEDs

In developing this circuit, we began by building a current source which includes a LF6132 opamp, a voltage divider to ensure a mid-rail voltage at the output of the Opamp, an NPN BJT at the output of the opamp, and a 331Ω resistor connected to ground. By doing this, we will receive an output current of:

$$I_{out} = \frac{V_{out}}{R} = \frac{2.5V}{331\ \Omega} = 7.55mA$$

The current from this current source controls the intensity of light which will be emitted by the LED's on the opposite end of the circuit. In order to turn on the LED's, the Red light LED and the IR light LED needs an on-Voltage bias of 1.72V and 1.04V respectively. In order to create these voltage biases, we used a quad switch IC MAX4066 and a control line from the microprocessor (Cypress PSoC) to control the LEDs.

Control line's input (digital 1 or 0) is the input to the NAND gate. The idea is that using 4 switches and the inverted and non-inverted control logic from the PSoC we can control the four switches and thereby the direction of current flow. By passing the current in either direction through the combination of R and IR LEDs, we can generate signals to excite the photodiode.

Noise Analysis: R1 /R2 and R3: The thermal noise across R1 and R2:

$$\sigma_{R1,R2}^2 = 4KTR * Gain.^2$$

Where K is Boltzmann constant, T is the Temperature, R is the specific resistor value (R1 = R2), and the gain is $\frac{1}{2}^2$. Since R1 and R2 form a voltage divider, the gain from noise source to the output is half (For R1 = R2). Notice that we have to use the square of the gain instead of the gain since we are dealing with the power spectral density here. When we compute the actual numerical value for this, we get:

$$\sigma_{R1,R2}^2 = 4.04 * 10^{-17} \frac{V^2}{Hz}$$

Now we can calculate the thermal noise on R3 which is described as:

$$\sigma_{R3}^2 = 4KTR_3 = 5.48 * 10^{-18} \frac{V^2}{Hz}$$

Trans- Impedance Amplifier Circuit

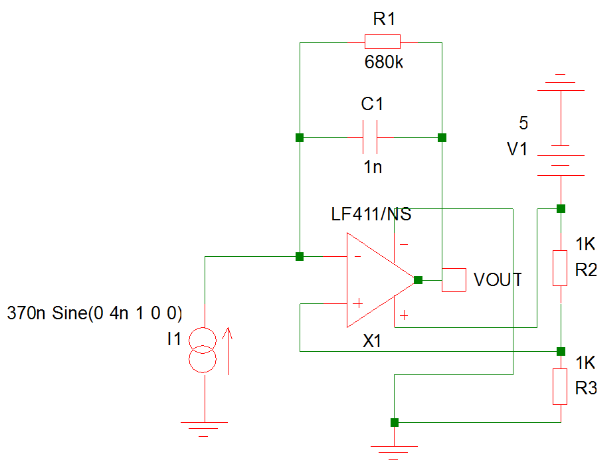


Figure 4: Transimpedance Amplifier circuit schematic

Expected Cutoff Frequency of Low Pass Filter: $f_c = \frac{1}{2*\pi*(6.81*10^5\Omega)*(100*10^{-12}F)} = 2.34kHz$

From the probe data sheet, the current which will flow out of the photodiode when IR is on is 0.32µA$_{DC}$ and 8nApp, and the current flowing while the Red light is on is 0.37µA$_{DC}$ and 7.5nA$_{pp}$. We can calculate the voltage at the output of our Trans-Impedance Amplifier to be:

$$V_{out} = I_{photo} * R$$

This comes out to be 0.218V_{DC} and 5.45mV_{pp} when IR is on, and 0.252V_{DC} and 5.1mV_{pp} when Red light is on. For this circuit, we used the Opamp LF411. In order to prevent our signal from being clipped we included a DC offset to place our signal mid-rail. This also keeps our signal within the input range of the ADC (0-5V).

Now we will analyze the noise of this circuit. To begin we can look at the LF411 datasheet to receive a value for the Equivalent Input Noise voltage (e_n) which is given to be 25 nV√Hz. Now in order to compute the noise calculation, we would need to integrate the function:

$$\sigma_1^2 = \int_{-\infty}^{\infty} e^2(\omega) * d\omega$$

When we go to the LF411 data sheet, we pull up the curve,

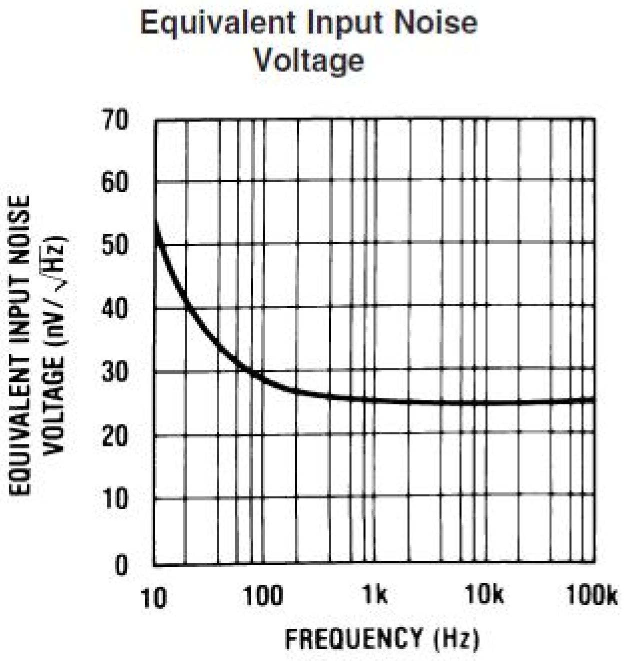


Figure 5: Input noise versus frequency curve of the LF411 op amp

(http://www.ti.com/lit/ds/symlink/lf411-n.pdf)

In order to accurately determine the noise from the input voltage, we would need to fit this curve and integrate from zero to the cutoff frequency defined by our Trans-Impedance Amplifier ($f_c = 2.34kHz$). Since the equation for this curve is unknown, we will perform a SIMetrix simulation for the noise of this system. However, before we do this, we need to analyze the thermal noise produced by the feedback resistor and also the output voltage due to current noise[i]. According to reference 1: "NOISE ANALYSIS OF FET TRANSIMPEDANCE AMPLIFIERS", the thermal noise of the feedback resistor is the same as the thermal noise of a regular resistor except it is multiplied by the effective noise bandwidth: $B = \frac{\pi}{2} * f_c$. The Thermal noise equation can be equated from the expression:

$$\sigma_2^2 = 2 * \pi * R * K * T * f_c = 4.048 * 10^{-11} V^2$$

Where R is the feedback resistor, K is Boltzmann constant, T is the temperature, and f_c is the cutoff frequency of the Trans-Impedance Amplifier. We can compute the thermal noise of the resistor by taking the square root of the above equation. This gives:

$$\sigma_2 = 6.36 \mu V_{rms}$$

Since our input is 1.8mV$_{rms}$, we are above the minimum SNR of 40dB required and slightly below the hopeful 60dB mark.

Lastly, we can analyze the output voltage due to current Noise. In order to compute this, we take the parallel impedance of the feedback resistor and capacitor and multiply this by the Equivalent Input Noise current (I_n = 0.01pA*√Hz). However, the Equivalent Input Noise current is a function of frequency and the manufacturer of the LF411 did not include the plot showing this. So for now we will describe how this would be solved if we were given the proper information.

$$\sigma_3^2 = \int_0^{f_c} I_n * Z_p(\omega) * d\omega$$

Here, $Z_p(\omega)$ is the parallel impedance of the two feedback components. Now in order to compute the total noise in this circuit, we can conduct a square-sum-root of all the noise.

$$\sigma_{Total} = \sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2}$$

This equation gives us the total noise of our Trans-Impedance amplifier. Due to the difficulty of accurately calculating noise within the system, we used SIMetrix to compute the total noise within the system. This plot is attached below.

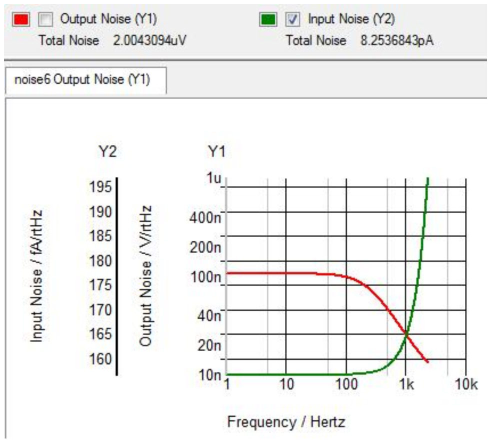

🟥 Output Noise (Y1) Total Noise 2.0043094uV
🟩 Input Noise (Y2) Total Noise 8.2536843pA

Figure 6: total noise within the system produced by SIMetrix

For Heart rate and SpO2, we know that the signal of our Heart beat is roughly 1 percent of the DC component. This means we need an absolute minimum of 40dB SNR. For this reason we chose to maintain a 60dB SNR. Had to choose a feedback resistor which gave us a low enough noise voltage so we were able to achieve the SNR of roughly 60dB.

Signal Amplification, Filtering and Processing

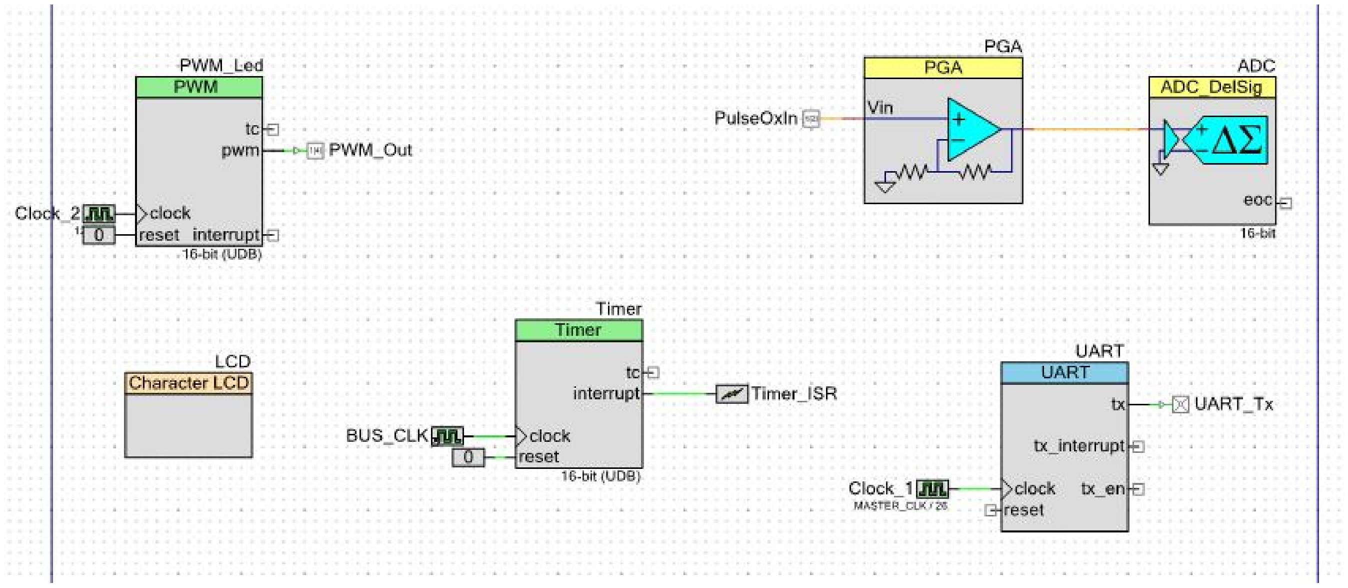


Figure 7: Top level hardware view of PSoC

In the Figure, the 'PWM_Led' block is set up to produce a square wave with a period of 1 ms. This square wave is the input to the LED switching circuit in Figure 4 such that every 500 µs the LED that is on switches from being red or infrared. The circuit shown in Figure 1 of the *Analog* section converts the photodiode current into voltage. This voltage is input to 'PulseOxIn' (Figure 9 above). The input voltage is buffered using a PGA with gain 1, and sent to a ∑Δ ADC. This Analog to Digital converter block has an ENOB of 16 bits and samples at 40 kHz. Such has a sampling frequencies and resolution allow us to meet our SNR demand of 80 dB without a problem.

The 'Timer' block is set up to have an interrupt every 500 µs. The program flow involves toggling between two phases: Red and Infrared. In the red phase, the red LED is on (PWM output is high) and one data point is collected via the ADC and stored in a buffer with a size of 256.This is followed by downsampling of the collected data to 2 kHz. Downsampling improves processing speed by reducing the amount of data. Digital filtering to smooth the data is the next step. When the timer interrupt occurs, the program in the infrared phase. Now the PWM output is low, making the infrared LED turn on. The ADC samples the data, the program downsamples and smooths the data. More about the filtering techniques can be found in the *Algorithms* section. Samples of collected data are shown in Figures 8 and 9.

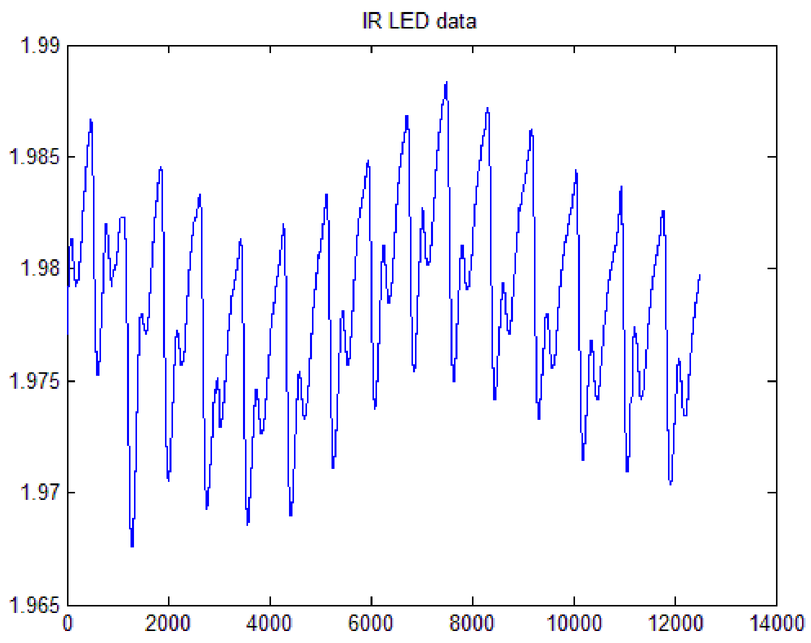


Figure 8: Infrared LED data after smoothing and downsampling

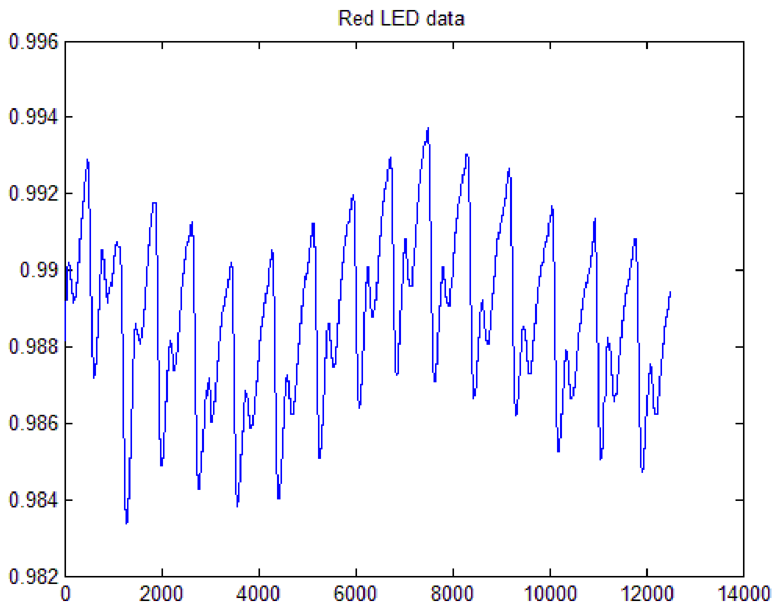


Figure 9: Red LED data after smoothing and downsampling

It can be seen that the data is smooth and that there is a DC offset between the red and the infrared data collected. Since there are no sudden jumps in the DC level within each figure, it is safe to assume that the data is not being interleaved. Note that the data displayed is collected via the UART and put together using MATLAB.

In each of the phases: red and infrared, algorithms perform heart rate and SpO2 calculations. These are displayed on the LCD screen of the PSoC. The refresh rate of the heart rate and SpO2 is every 1.5 seconds. While the group is satisfied with their heart rate results, the SpO2 requires better peak detection. As can be seen from Figures 9 and 10, false peaks are fairly common and need to be accounted for. See section on *Algorithms* for more details about heart rate and SpO2 calculations.

Bio-Impedance

We begin with an over view of our signal processing chain in Figure ii.

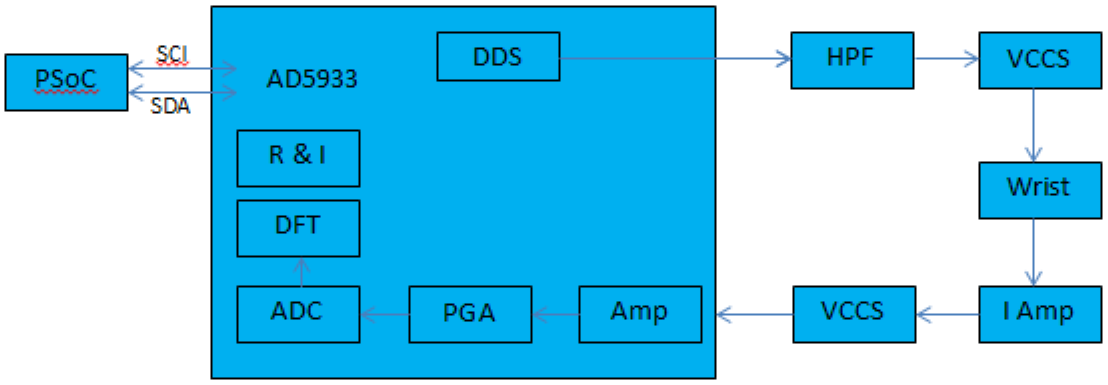


Figure ii: Overview of signal flow for Bio-Impedance measurements

Analog Front End Circuit

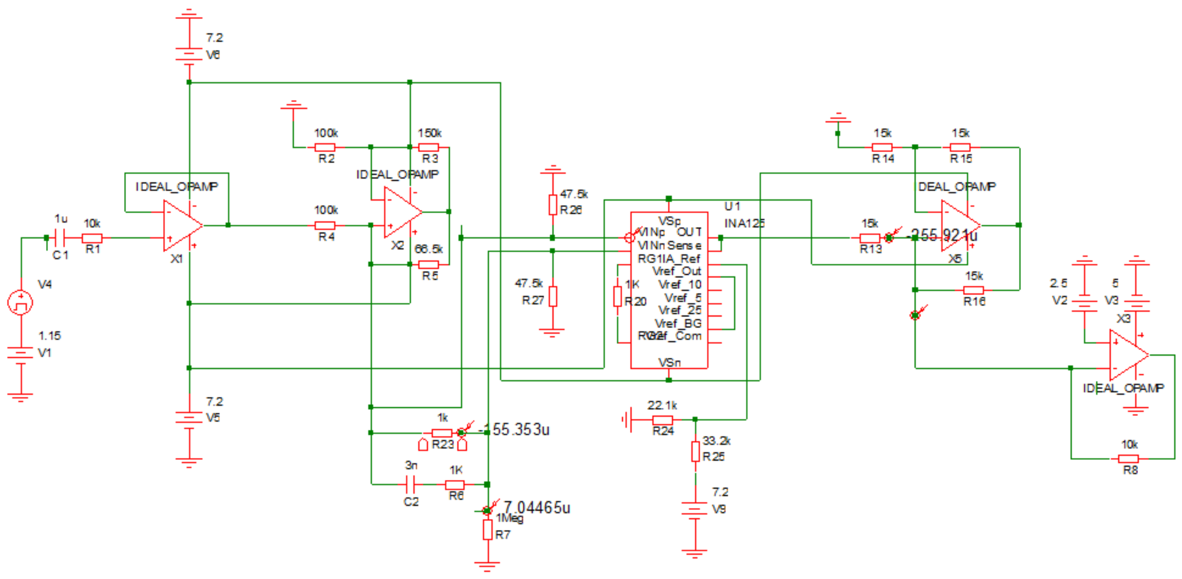


Figure 10: Analog front end of bioimpedance involving two howland circuits

High Pass Filter

A High pass filter was placed in this circuit to block the DC output of the AD5933. The cutoff frequency of this high pass filter can be calculated as:

$$f_c = \frac{1}{2 * \pi * (1 * 10^4 \Omega) * (1 * 10^{-6} F)} = 15.9\ Hz$$

Now we can compute the thermal noise of the high pass filter. The RMS value of the voltage for a given bandwidth can be computed as:

$v^2 = 4KTR * \Delta f = 16.39\ pV^2$

Where $\Delta f = 100kHz - 1kHz$.

Howland Current Source #1

After the high pass filter, we include a Howland Current Source (Figure 11) which controls the amount of current that will be passed through the load (in this case our team members' wrists). In order to stay below the limit of $10\mu A_{rms}$ we chose to use a resistor value of 100kΩ. Knowing that we have an input voltage that is $1.5V_{pp}$ we can compute the current leaving the Howland Current Source as:

$I_{out} = \frac{V_{in}}{R} = 5.3\mu A_{\mathrm{rms}}$

Now computed the thermal noise is slightly more difficult than before. In order to do this properly, we must think of each resistor as having a "noise voltage source" connected in series with it.

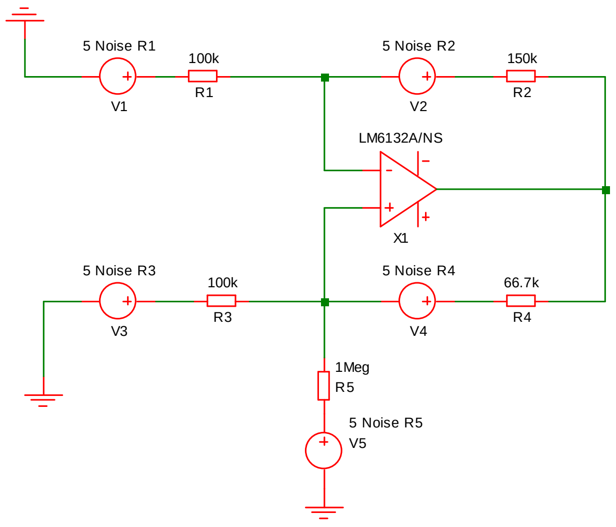


Figure 11: Howland circuit 1 used in the schematic of Figure 5

Now we would need to go through this circuit solving for the scale factor for each resistor.

Each resistor has its own Johnson's noise. However, the output power spectral density due to each of them will be affected differently, depending on the scaling factor associated with each of these resistors. We are showing the relations taken from this reference for estimating the contribution of each resistor's Johnson's noise. These relationships are as follows:

$$\alpha = (R_3 * R_5) + (R_4 * R_5) + (R_3 * R_4)$$

$$\beta = \alpha * R_1 - (R_1 + R_2) * (R_3 * R_5)$$

$$e_{number\ of\ resistor} = \sqrt{4KTR * \Delta f}$$

$$V_{out,1} = \frac{-\alpha * e_1 * R_2}{\beta}$$

$$V_{out,2} = \frac{\alpha * e_2 * R_1}{\beta}$$

$$V_{out,3} = \frac{e_3 * R_4 * R_5 * (R_1 + R_2)}{\beta}$$

$$V_{out,4} = \frac{e_4 * R_3 * R_5 * (R_1 + R_2)}{\beta}$$

$$V_{out,5} = \frac{e_5 * R_3 * R_4 * (R_1 + R_2)}{\beta}$$

Where $V_{out,number\ of\ resistor}$ is the noise contribution of the specific resistor at the output of the Howland Current Source. By solving for these values using MATLAB, we received:

$$V_{out,1} = 43.4 \frac{\mu V_{rms}}{\sqrt{Hz}}$$

$$V_{out,2} = 35.4 \frac{\mu V_{rms}}{\sqrt{Hz}}$$

$$V_{out,3} = 27.8 \frac{\mu V_{rms}}{\sqrt{Hz}}$$

$$V_{out,4} = 34.11 \frac{\mu V}{\sqrt{Hz}}$$

$$V_{out,5} = 8.8 \frac{\mu V_{rms}}{\sqrt{Hz}}$$

These values are independent of the polarity. Now we can do a square-sum-root of these noise contributions. This is:

$$V_{out,total} = \sqrt{V_{out,1}^2 + V_{out,2}^2 + V_{out,3}^2 + V_{out,4}^2 + V_{out,5}^2}$$

$$V_{out,total} = 71.8 \frac{\mu V_{rms}}{\sqrt{Hz}}$$

To put this noise calculation in perspective, the voltage at the output of the Howland current source is going to be the product of output current from Howland current source and the total impedance through which the current flows. Since the total impedance will be a sum of the voltage across 1Meg resistor as well as the impedance of the wrist, the total voltage at the output will be in order of Volts, with the voltage across wrist in order of mV, since the wrist impedance is assumed to be around 5k. The approximate Voltage difference across the wrist is 25.6 mV_{rms}. This means we have an SNR which is larger than the minimum value of 40dB and much closer to the desire 60dB range. The exact calculation is:

$$SNR = 20 * log_{10}\left(\frac{25.6 mV_{rms}}{71.8 \mu V_{rms}}\right) = 51.04\ dB$$

<u>Howland Current Source #2</u>

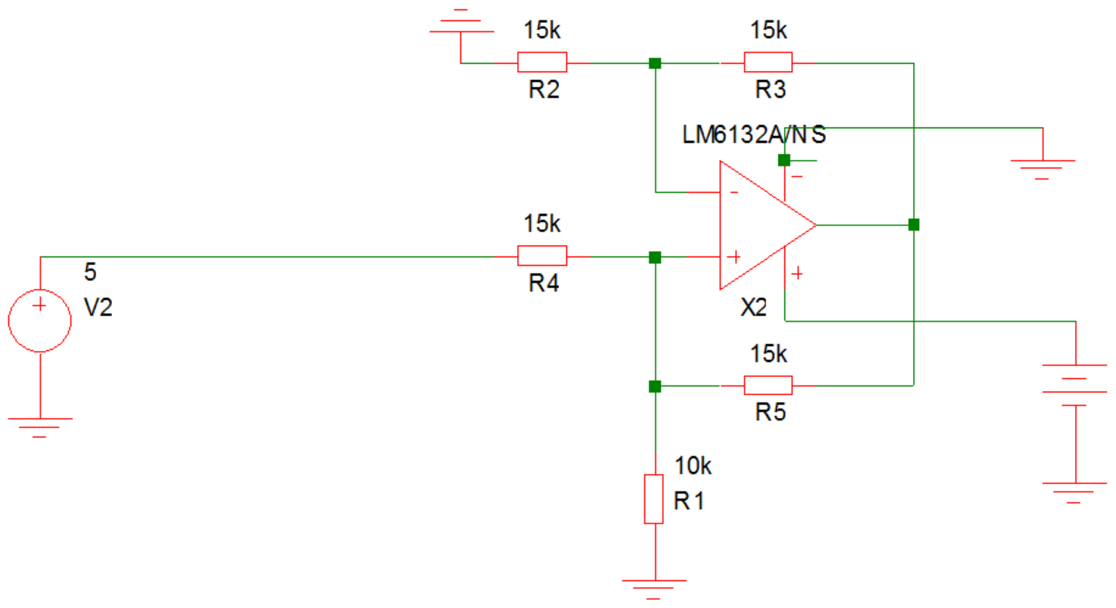


Figure 12: Howland circuit 2 used in schematic of Figure 5

Directly after the Instrumentation Amplifier we have another Howland Current Source. We needed such a current source because the AD9533 chip can only receive current inputs. For this design, the current was less of a concern except for the fact that it dictates which value must be chosen for the resistor (R_{FB})

which goes into the AD9533 chip. Depending upon the output current, we would choose the R_{FB} resistor to be a size which would have a voltage that stayed within the range of the ADC which is (0-5V). We tested to makes sure we stayed within this range by estimating that the maximum Load that we would add would be 10kΩ. Once we concluded that the voltage swing for a load of 10kΩ would stay within the range of the ADC we needed to make sure that our noise from the Howland would not affect the signal we are attempting to measure. The analysis for noise follows the same logic as above.

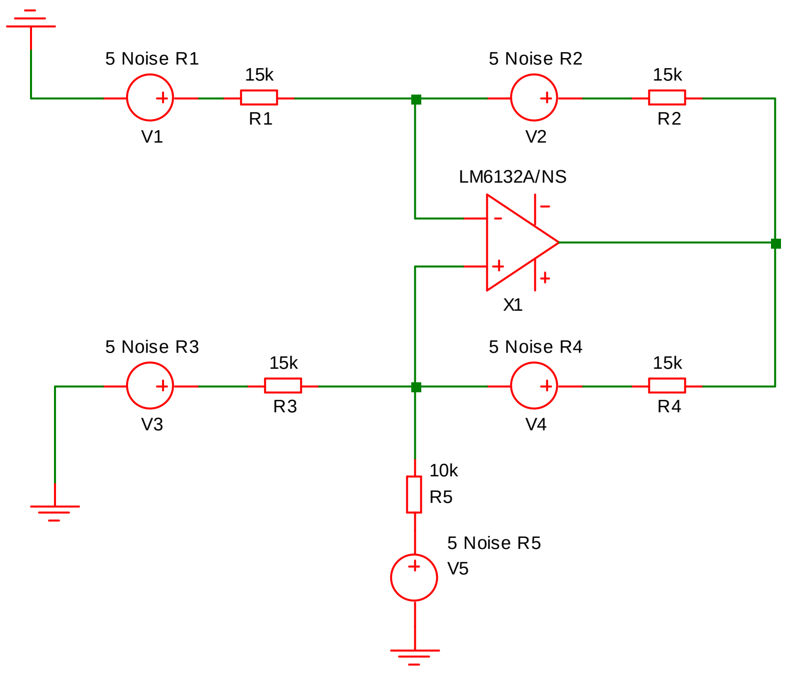


Figure 13: Noise analysis schematic

To compute the noise, we can do a similar calculation to which has been shown above. The only change has been the resistor values. When we make these changes, we were able to compute the contributions of noise from each resistor on the output voltage of the Howland Current source to be:

$$V_{out,1} = 11.5\frac{\mu V_{rms}}{\sqrt{Hz}}$$

$$V_{out,2} = 11.5\frac{\mu V_{rms}}{\sqrt{Hz}}$$

$$V_{out,3} = 6.6\frac{\mu V_{rms}}{\sqrt{Hz}}$$

$$V_{out,4} = 6.6\frac{\mu V}{\sqrt{Hz}}$$

$$V_{out,5} = 8.09\frac{\mu V_{rms}}{\sqrt{Hz}}$$

$$V_{out,total} = \sqrt{V_{out,1}^2 + V_{out,2}^2 + V_{out,3}^2 + V_{out,4}^2 + V_{out,5}^2}$$

$$V_{out,total} = 20.5\frac{\mu V_{rms}}{\sqrt{Hz}}$$

Now we must compare this noise voltage to the voltage of our signal. If we were to simulate a load resistance of 10kΩ, the voltage at the output of the Instrumentation Amplifier is 1.15V_{pp} or 0.4066 V_{rms}. This means that the current which leaves the Howland Current source is 27.1μA_{rms}. When the signal passes through R_{FB}, it gets converted back to a Voltage w AC component of 0.271V_{rms}. We can calculate the SNR for this to be:

$$SNR = 20 * log_{10}\left(\frac{0.271V_{rms}}{20.5\mu V_{rms}}\right) = 82.45\ dB$$

As you can see this SNR is much higher than the desired SNR of the system. This shows that the noise generated in the second Howland Current Source will not affect the signal we are analyzing.

I2C and Xbee

The group used the AD5933 chip to make bioimpedance measurements. On the PSoC, a master_I2C block was used to communicate with the chip. A frequency sweep was generated at the Vout pin of the chip and acted as the input to the analog front end shown in Figure 5 in the *Analog* section. Its output went back into the AD5933 chip via the Vin pin. The Rfb used was 10 kΩ.

For I2C communication, the PSoC is the master and the chip is treated as the slave. From the datasheet, its address is 0x0D. Note that the communication is asynchronous i.e. the master and the slave do not share a clock. They however share bidirectional digital I/O pins: SDL and SDA. These are open drain lines that need pull up resistors in order to be able to go high. The steps for writing to the chip are:

1. Send a master signal with the slave address (0x0D) and the write command (0).
2. Write the byte of the register whose contents need to be altered
3. Write the desired byte value
4. Send a master stop signal

The steps for reading from the chip are:

1. Send a master signal with the slave address (0x0D) and the write command (0).
2. Write a command byte (0xB0)
3. Write the byte of the register whose contents need to be read
4. Send a master stop signal
5. Send a master signal with the slave address (0x0D) and the read command (1)
6. Send a master read byte command with input 0
7. Send a master stop signal

In order to generate a frequency sweep from 1 Kz to 100 kHz in 100 increments with amplitude of 1.5 V and 1.15 V Dc bias, the AD5933 chip was configured using I2C and the state diagram shown in Figure 13. All the registers associated can be found in the datasheet of the chip.

The real and imaginary data were conveyed to matlab using xbee. The bioimpedance between different members of the group was different in magnitude and consistent for different trials. Figure 12 shows the raw data of two group members.

Using a KNN (K = 5) classification system, the users could be identified on MATLAB. Implementing the same functionality on the PSoC proved to be a challenge because of the way PSoC treats signed integers. More about classification systems can be found in the *Algorithms* section.

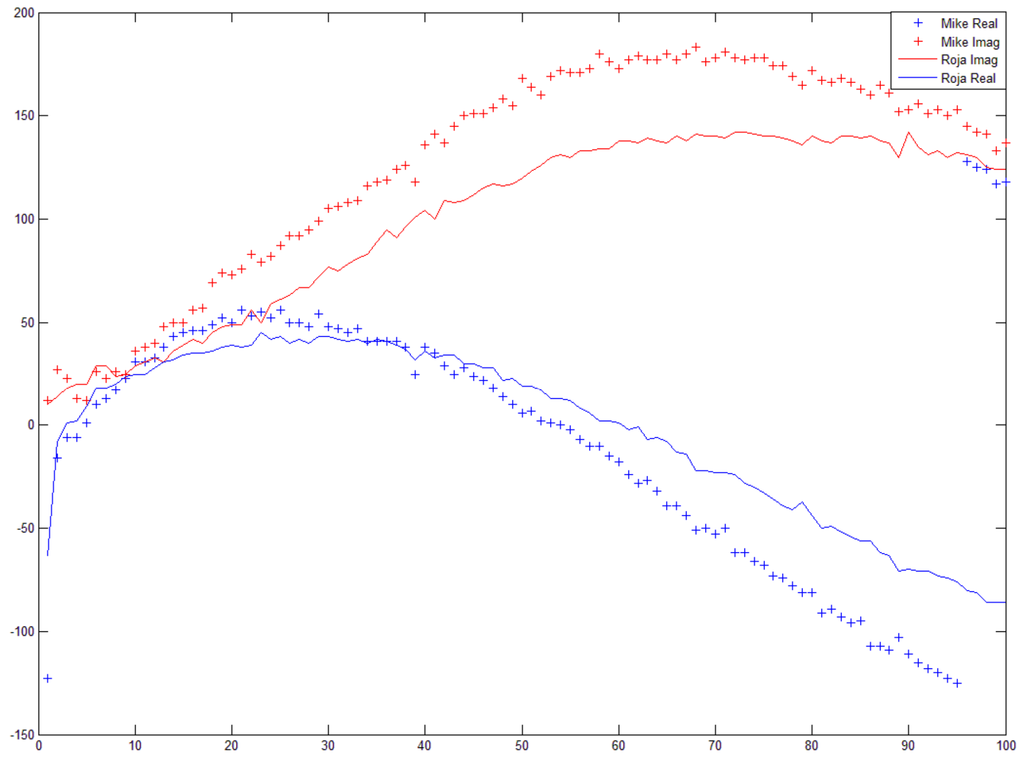


Figure 13: Raw bioimpedance data collected using MATLAB and transmitted using XBee

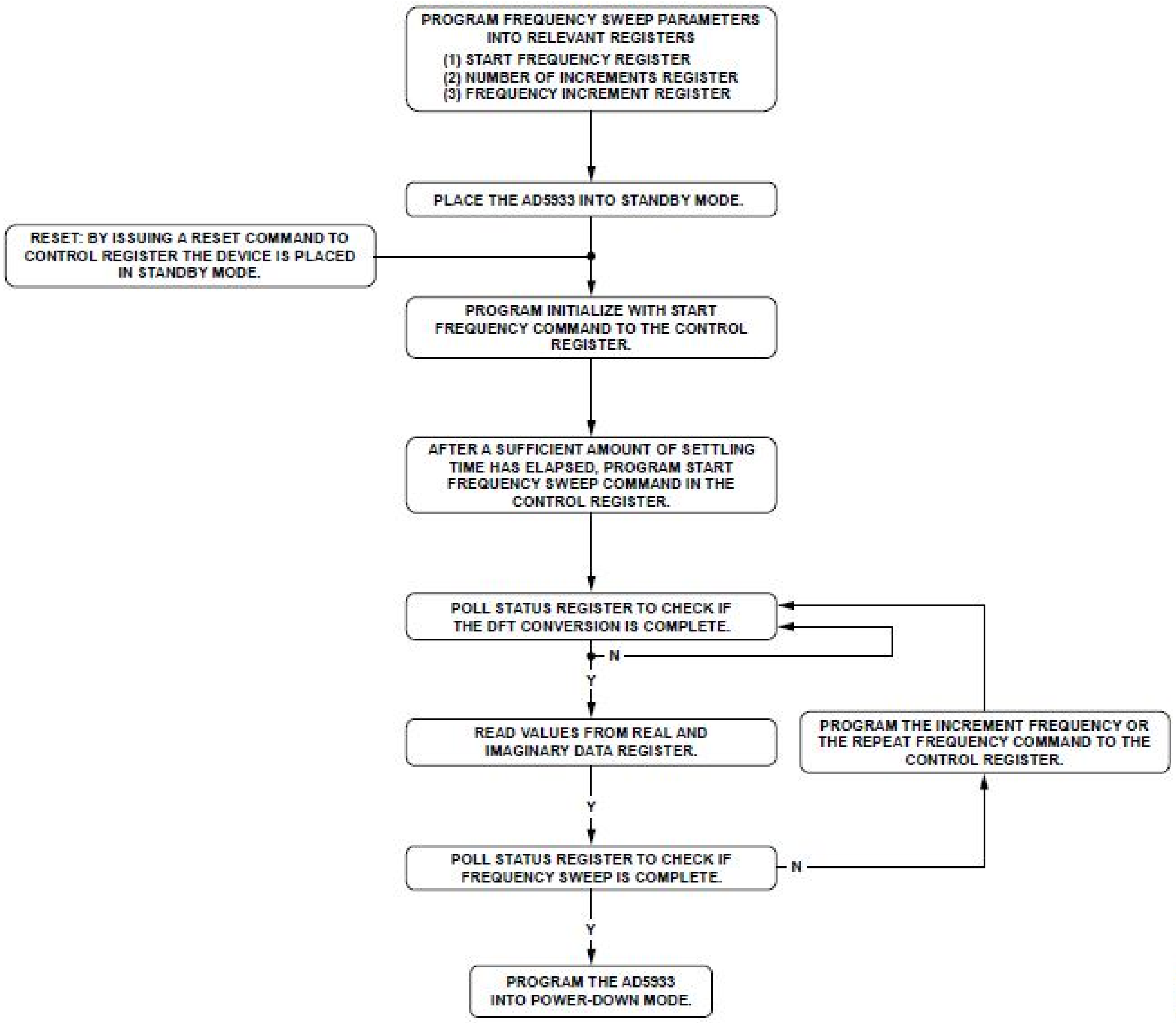
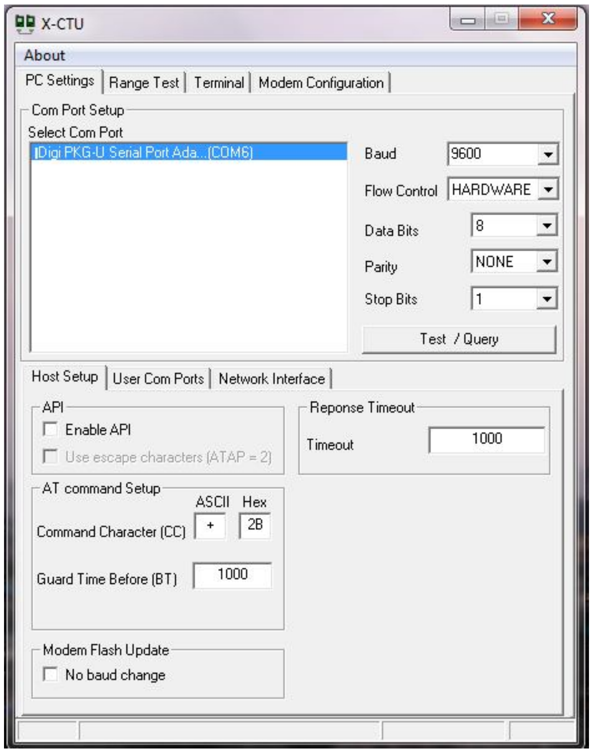

PROGRAM FREQUENCY SWEEP PARAMETERS INTO RELEVANT REGISTERS
(1) START FREQUENCY REGISTER
(2) NUMBER OF INCREMENTS REGISTER
(3) FREQUENCY INCREMENT REGISTER
PLACE THE AD5933 INTO STANDBY MODE.
RESET: BY ISSUING A RESET COMMAND TO CONTROL REGISTER THE DEVICE IS PLACED IN STANDBY MODE.
PROGRAM INITIALIZE WITH START FREQUENCY COMMAND TO THE CONTROL REGISTER.
AFTER A SUFFICIENT AMOUNT OF SETTLING TIME HAS ELAPSED, PROGRAM START FREQUENCY SWEEP COMMAND IN THE CONTROL REGISTER.
POLL STATUS REGISTER TO CHECK IF THE DFT CONVERSION IS COMPLETE.
N
Y
READ VALUES FROM REAL AND IMAGINARY DATA REGISTER.
PROGRAM THE INCREMENT FREQUENCY OR THE REPEAT FREQUENCY COMMAND TO THE CONTROL REGISTER.
Y
POLL STATUS REGISTER TO CHECK IF FREQUENCY SWEEP IS COMPLETE.
N
Y
PROGRAM THE AD5933 INTO POWER-DOWN MODE.

Figure 14: State diagram of steps to complete a frequency sweep on the AD5933 chip. (http://www.analog.com/static/imported-files/data_sheets/AD5933.pdf)

To wirelessly communicate, the group used Digi's XBee IC module. To get the XBee module configured and running, the user has to download the X-CTU software, USB bus and virtual COM port drivers from digi.com. Once all the necessary software is available and compatible, configuration can begin. These are the steps followed to configure a home station XBee module to receive data transmitted wirelessly via serial by transmitter module. Note that XBee only runs on 3.3 V.

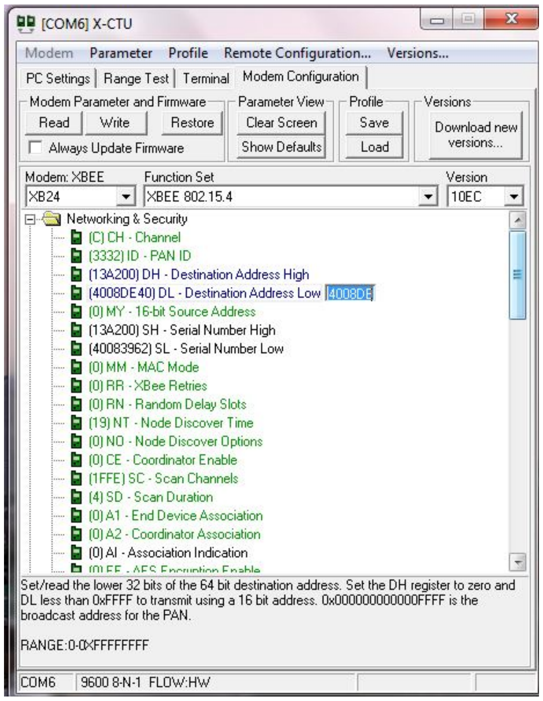
1. Open the X-CTU software. A screen shot can be seen in Figure 14.



Digi PKG-U Serial Port Ada...(COM6)
Baud 9600
Flow Control HARDWARE
Data Bits 8
Parity NONE
Stop Bits 1
Enable API
Use escape characters (ATAP = 2)
Timeout 1000
Command Character (CC) + 2B
Guard Time Before (BT) 1000
No baud change

Figure 15: Welcome Screen of X-CTU software

2. One the correct Com Port is chosen, press the Test/Query button to obtain the Modem type, Model firmware version and the Serial Number. Note that the host and the receiver should be configured to have the same Modem type and Modem firmware version. Note down the Serial Number for both the host and the receiver.
3. With the transmitter module in place, click on the Modem Configuration tab. Check the Always Update Firmware box and click on 'Read'. Figure 2 shows the data that it reads. The source address is automatically filled. Enter the destination address to be Serial Number of the host module. Make sure that the Modem type and firmware version are the same as that of the host and click on 'Write'. When this operation is successful, data can be wirelessly transmitted from the transmitter to the host one byte at a time.



```
[COM6] X-CTU
Modem  Parameter  Profile  Remote Configuration...  Versions...
PC Settings | Range Test | Terminal | Modem Configuration
Modem Parameter and Firmware: Read  Write  Restore  [ ] Always Update Firmware
Parameter View: Clear Screen  Show Defaults
Profile: Save  Load
Versions: Download new versions...
Modem: XBEE  XB24     Function Set  XBEE 802.15.4     Version  10EC
Networking & Security
  (C) CH - Channel
  (3332) ID - PAN ID
  (13A200) DH - Destination Address High
  (4008DE40) DL - Destination Address Low  4008DE
  (0) MY - 16-bit Source Address
  (13A200) SH - Serial Number High
  (40083962) SL - Serial Number Low
  (0) MM - MAC Mode
  (0) RR - XBee Retries
  (0) RN - Random Delay Slots
  (19) NT - Node Discover Time
  (0) NO - Node Discover Options
  (0) CE - Coordinator Enable
  (1FFE) SC - Scan Channels
  (4) SD - Scan Duration
  (0) A1 - End Device Association
  (0) A2 - Coordinator Association
  (0) AI - Association Indication
  (0) EE - AES Encryption Enable
Set/read the lower 32 bits of the 64 bit destination address. Set the DH register to zero and
DL less than 0xFFFF to transmit using a 16 bit address. 0x000000000000FFFF is the
broadcast address for the PAN.

RANGE:0-0XFFFFFFFF
COM6   9600 8-N-1  FLOW:HW
```

Figure 16: Configuration tab of the X-CTU software

All of the configuration chances can also be done from the terminal window. We used a baud rate of 9600, which is default. Make sure that the PSoC's UART has the same baud rate setting as the xbee modules. Pins 1 and 10 are power and ground respectively. We used pin 3 to transmit the serial data of the transmitter. Pin 5 is a reset button that has to connect to power via a pull up resistor.

The resolution that we set was an 80% success rate on subject identification. This was decided because consistency in data changed with placement of probes, time of day, and subject's perspiration level. Although our subject identification did not work on the PSoC, it worked when the data was wirelessly transmitted to a local computer and we run a MATLAB script (see Section *Code*). Having a higher order classifier could have helped with reduction in error. When 25 trials were performed using MATLAB, 19 were correct, which gives us a success rate of 76%.

Algorithms

Heart Rate

The heart rate algorithm was implemented with a peak detection algorithm. The algorithm would look at each data point and determine if it was bigger than the current maximum. If it was, that point would become the new maximum. If the next point was smaller than the maximum by more than a threshold, then the maximum would be considered a peak. The reason for thresholding was to help avoid false peaks being detected due to noise. In general, the pulse ox data looked fairly Gaussian and detecting peaks could be accomplished with a simple algorithm. After finding a maximum, the algorithm would then look for a minimum in the same fashion. In this way, it would find a maximum and then a minimum, and then a new maximum, ect… The algorithm was also designed to make sure that peaks could not be too close. This helped eliminate problems with heartbeats that had 2 distinct peaks in a single beat.

To find the heart rate, the time difference between maximums and the difference between minimums in a short time window were computed. The window length that ended up working the best was 4 seconds. The number that was reported on the LCD screen was a median value of all heartbeats for maximums and minimums for Red and IR channels.

The detection of a sample heartrate is shown in the plot below. The double peaks are ignored by the algorithm.

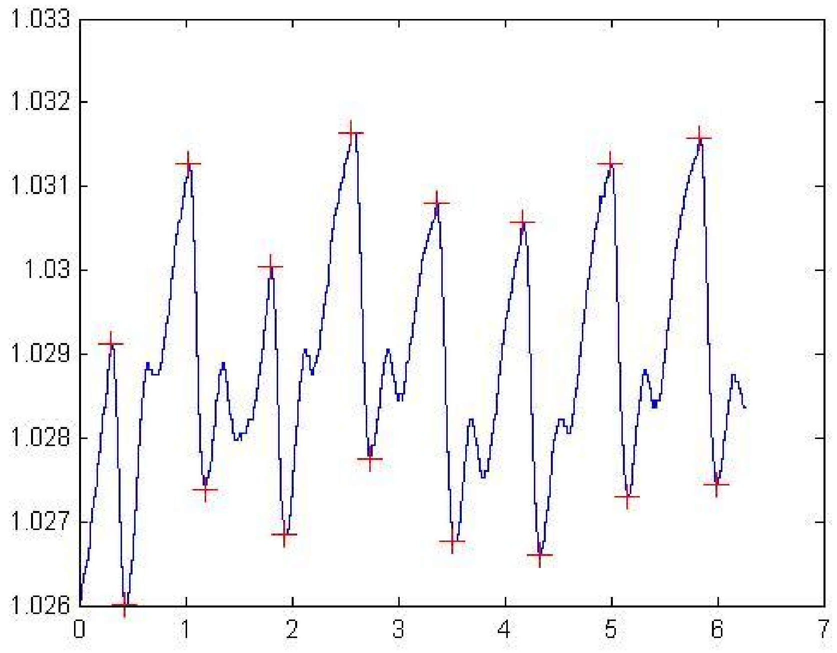


Figure 17: Peak detection example

SpO2

Using the maximum and minimum values found in the heart rate algorithm, the log of their rations was taken. This was then multiplied but the inverse matrix of HbO and HbR at 660 and 920 nm. These values were taken from a Matlab script released through MGH at Harvard. Finally, the SpO2 value was found as the ratio of HbO/(HbO+HbR). Once again, the values were taken over a 4s window and the median value displayed. If the value was greater than 100, it was not shown on the screen.

Digital Filtering

Two different digital filters were implemented. The first was a 6th order Butterworth filter, with a cutoff frequency at 10 Hz. This ensured that enough filtering was in place to downsample the data to 50 Hz.

The magnitude and phase are shown in the plot below.

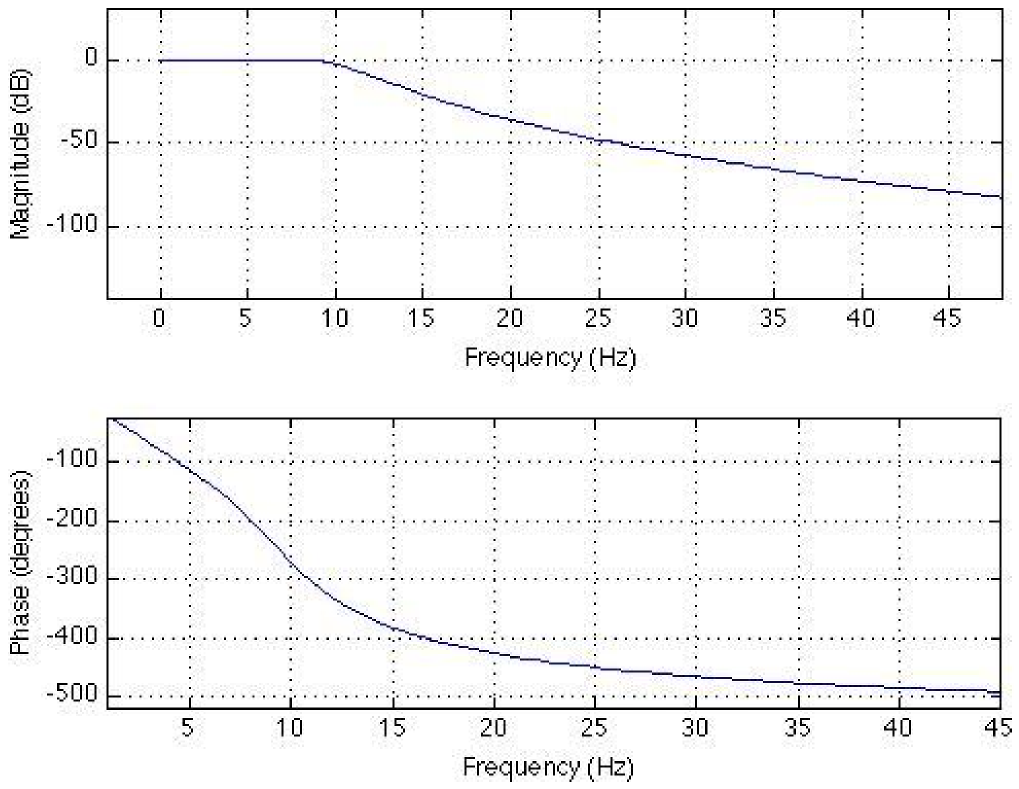


Figure 18: Magnitude and pahse of the digital filters

The filter was found using the state space output from the Butter() function in Matlab. Using the A,B,C,D matrix values, it was possible to pass every value from the ADC through the filter as they were read. In order to keep the channels separated, two state vectors were maintained. One for red light and one for IR light. Once data was filtered, every 20th point taken and stored in a buffer, such that the data was downsampled from 1 kHz to 50 Hz.

This buffer was then further filtered using a Hamming Window convolution filter. The Hamming Window used was an 11 point window. This meant that high frequencies should be smoothed out and low frequencies unchanged. It also meant that data processing was delayed by 5 time points because of the Hamming Window. However, this smoothing was necessary to help ensure that the heart rate algorithm worked properly and didn’t find false peaks.

Classifier

A k=5 classier was implemented to classify data taken bioimpedence. The classifier found the distance between the real and imaginary data from a test subject and the training data. This was accomplished by looping through each person, and using their training data to find the 2-norm distance between the real and imaginary parts at each frequency. The frequency range used for the classifier was 11-90 kHz because the beginning and ending of the sweep seemed to produce less stable data.

Each training set would produce a distance that was ranked in a list of the 5 shortest distances. The person that was displayed on the LCD screen was the person with the highest number of distances in the list. This algorithm worked well in Matlab during testing and was able to classify 3 group members. Unfortunately, the implementation on the Psoc was less successful and eventually failed completely during the demo. The likely cause was a conversion error between the data stored during training and the data taken during evaluation.

Code

Heart rate and SpO_2

```
/* =========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * =========================================
*/
#include <device.h>
#include <math.h>

/* hamming window*/
float lham[101] = {1.47928994082840E-03,1.49607442535124E-
03,1.54636163822744E-03,1.62995311880183E-03,1.74651896968990E-
03,1.89559915873205E-03,2.07660533453041E-03,2.28882314840294E-
03,2.53141507359140E-03,2.80342371059677E-03,3.10377556559771E-
03,3.43128528704025E-03,3.78466034367882E-03,4.16250612560657E-
03,4.56333144814354E-03,4.98555443686136E-03,5.42750877051889E-
03,5.88745025727091E-03,6.36356371819650E-03,6.85397015098092E-
03,7.35673414547937E-03,7.86987152189657E-03,8.39135716143799E-
03,8.91913299852848E-03,9.45111614305687E-03,9.98520710059172E-
03,1.05192980581266E-02,1.10512812026550E-02,1.15790570397454E-
02,1.21005426792869E-02,1.26136800557041E-02,1.31164440502025E-
02,1.36068504829869E-02,1.40829639439125E-02,1.45429054306645E-
02,1.49848597643221E-02,1.54070827530399E-02,1.58079080755769E-
02,1.61857538575046E-02,1.65391289141432E-02,1.68666386355857E-
02,1.71669904905867E-02,1.74389991275920E-02,1.76815910527805E-
02,1.78938088666530E-02,1.80748150424514E-02,1.82238952314935E-
02,1.83404610823816E-02,1.84240525629560E-02,1.84743397758322E-
02,1.84911242603550E-02,1.84743397758322E-02,1.84240525629560E-
02,1.83404610823816E-02,1.82238952314935E-02,1.80748150424514E-
02,1.78938088666530E-02,1.76815910527805E-02,1.74389991275920E-
02,1.71669904905867E-02,1.68666386355857E-02,1.65391289141432E-
02,1.61857538575046E-02,1.58079080755769E-02,1.54070827530399E-
02,1.49848597643221E-02,1.45429054306645E-02,1.40829639439125E-
02,1.36068504829869E-02,1.31164440502025E-02,1.26136800557041E-
02,1.21005426792869E-02,1.15790570397454E-02,1.10512812026550E-
02,1.05192980581266E-02,9.98520710059172E-03,9.45111614305687E-
03,8.91913299852848E-03,8.39135716143799E-03,7.86987152189657E-
03,7.35673414547937E-03,6.85397015098092E-03,6.36356371819650E-
03,5.88745025727091E-03,5.42750877051889E-03,4.98555443686136E-
03,4.56333144814354E-03,4.16250612560657E-03,3.78466034367882E-
03,3.43128528704025E-03,3.10377556559771E-03,2.80342371059677E-
03,2.53141507359140E-03,2.28882314840294E-03,2.07660533453041E-
03,1.89559915873205E-03,1.74651896968990E-03,1.62995311880183E-
03,1.54636163822744E-03,1.49607442535124E-03,1.47928994082840E-03};
float hham[11] = {0.0145985401459854,     0.0306299603261905
      ,0.0726007632458986     ,0.124479528724904      ,0.166450331644612,
      0.182481751824818 ,0.166450331644612      ,0.124479528724904,
      0.0726007632458986,     0.0306299603261905      ,0.0145985401459854};
/*  */

uint16 filter(float* A, float* B, float* C, float* D, float* x, uint8 fOrd,
uint8 isLow, uint16 intDat);
uint16 hammingFilter( uint8 ptr, uint16* datBuf, float* ham,  uint8 cnt,
uint8 isLow);
void sendUart(uint8 isRed, uint16 dat);
uint16 medianVal(float* dat, uint8 count);
void CountMax(uint16* dataRed, uint16* dataIR, uint8 count);
void MaxMin(uint16* dat, uint8 count, uint8* maxPos, uint8* cntmx, uint8*
minPos, uint8* cntmn);
float findSpO2(uint16* mx, uint16* mn);

//uint16 slave_address = "0x0D";


uint8 hr(uint8 isIR);
void sendUart(uint8 isRed, uint16 dat);
uint16 filter(float* A, float* B, float* C, float* D, float* x, uint8 fOrd,
uint8 isLow, uint16 intDat);


uint8 timer_250u_flag;
char LUTHex[] = "0123456789ABCDEF";
char COut[] = "IR";
char hrStr[] = "HR=";
char spo2Str[] = "SpO2=";
//uint16 slave_address = "0x0D";

      uint16 Fs = 50;
      uint8 offset = 255;
      uint16 redData[256];
      uint16 irData[256];
      uint16 maxtabRed[256];
      uint16 mintabRed[256];
      uint16 maxtabIR[256*2];
      uint16 mintabIR[256*2];
      uint8 redPtr;
      uint8 irPtr;


void main()
{
      uint8 phase = 2;
      uint16 output = 0;
      uint16 dig_filter_output;
      uint8 ptrData = 0;

      uint8 ds1Count =0;

      uint16 fdatpIR =0;
      uint16 fdatpRed = 0;
      uint16 dataHRed[256];
      uint16 dataHIR[256];
      uint16 dataSIR[256];
      uint16 dataSRed[256];
      uint16 dataHSIR[256];
      uint16 dataHSRed[256];

      float Al[36] = {
       0.502560881689557  , 0.237982264299942 ,  0.030176479469455 ,
0.004779484811734  , 0.000683782448592 ,  0.000108300500424,
  -0.237982264299942 ,  0.962307312261688 ,  0.248823274689888 ,
0.039409735101476 ,  0.005638198723822 ,  0.000893002949311,
                   0  ,                 0  , 0.601184806929141 ,
0.253602759501623 ,  0.036281968181162 ,  0.005746499224246,
                   0  ,                 0 , -0.253602759501623 ,
0.959833268871578 , 0.280385783030588  , 0.044408745320257,
                   0  ,                 0 ,                  0  ,
0 ,  0.806568130483884 ,  0.286132282254834,
                   0   ,                0 ,                  0 ,
0 , -0.286132282254834 ,  0.954681098616286 };

      float Bl[6] = {0.336557745777237  , 0.053305510201815 ,
0.006759212241911 ,  0.001070554047970,   0.000153160036511  ,
0.000024258166663};
      float Cl[6] = {0.000038290009128 ,  0.000315724220539 ,
0.002031694284774,   0.015700862479970 ,  0.101163038549388 ,
0.691084129944373};
      float Dl = 8.576557073259405e-006;

      float xlIR[6] = {0,0,0,0,0,0};
      float xlRed[6] = {0,0,0,0,0,0};

    CyGlobalIntEnable;
      Clock_1_Start();
      Clock_2_Start();
      PWM_Led_Start();
      Opamp_Start();
      Timer_Start();
      Timer_ISR_Start();
      ADC_Start();
      UART_Start();
      LCD_Start();
      ADC_IRQ_Disable();
      PGA_Start();

      ADC_StartConvert();
      LCD_ClearDisplay();

      while(1) {


            if(timer_250u_flag == 1) {
                  //Control_Reg_PWM_Write(0x00);
                  //PWM_Led_Start();
                  timer_250u_flag = 0;
                  Timer_ReadStatusRegister();


                  if(phase == 1)
                  {
                        uint16 c = 0;
                        while(c < 1)
                        {
                              if(ADC_IsEndConversion(ADC_WAIT_FOR_RESULT))
                              {
                                    output = (uint16) ADC_GetResult16();
                                    c = c + 1;
                              }
                        }
                        dig_filter_output = filter(&Al, &Bl, &Cl, &Dl,
&xlRed, 6, 1, output);

                        if (ds1Count == 19)
                        {
                              dataHRed[ptrData] = dig_filter_output;
                              dataSRed[ptrData] = hammingFilter(ptrData-5,
&dataHRed, &hham, 5, 1);
                              sendUart(1, dataSRed[ptrData]);
                        }
                  }

                  if(phase == 2) {
                        uint16 c = 0;
                        while(c < 1)
                        {
                              if(ADC_IsEndConversion(ADC_WAIT_FOR_RESULT))
                              {
                                    output = (uint16) ADC_GetResult16();
                                    c = c + 1;
                              }
                        }

                        dig_filter_output = filter(&Al, &Bl, &Cl, &Dl, &xlIR,
6, 1, output); //Filter_Read16(1);

                        if (ds1Count == 19)
                        {
                              dataHIR[ptrData] = dig_filter_output;
                              dataSIR[ptrData] = hammingFilter(ptrData-5,
&dataHIR, &hham, 5, 1);
                              sendUart(0, dataSIR[ptrData]);

                              if (ptrData == 255)
                              {
                                    CountMax(dataSRed, dataSIR, ptrData);
                              }

                              ptrData = ptrData + 1;
                              ds1Count = 0;

                        }
                        ds1Count = ds1Count+1;
            }

                  if(phase == 2) {
                        phase = 1;
                  }
                  else {
                        phase++;
                  }

            } //timer flag

      } //while loop

} //main loop

uint16 filter(float* A, float* B, float* C, float* D, float* x, uint8 fOrd,
uint8 isLow, uint16 intDat)
{
      float dat = 0;
      float tmpa[8] = {0,0,0,0,0,0,0,0};
      float tmpb[8] = {0,0,0,0,0,0,0,0};
      float tmpc = 0;
      uint8 r = 0;
      uint8 c = 0;
      float fdat = 0;

      dat = 1.0*intDat/65536.0;

      for (r = 0; r < fOrd; r++)
      {
        for (c = 0; c<fOrd; c++)
        {
                  tmpa[r] = tmpa[r] + A[r + c*fOrd]*x[c];
        }

        tmpb[r] = B[r]*dat;

    }
    for (c = 0; c<fOrd; c++)
      {
            tmpc = tmpc + C[c]*x[c];
    }

      fdat = tmpc + (*D)*dat;

    for( r = 0; r < fOrd; r++)
      {
        x[r] = tmpa[r] + tmpb[r];

      }
      if (isLow)
      {
            return fdat*65535;
      }
      else
      {
            return fdat*65535 + 32768;
      }
}
```

```
void MaxMin(uint16* dat, uint8 count, uint8* maxPos, uint8* cntmx, uint8*
minPos, uint8* cntmn)
{
      int32 cur =0;
      uint8 mxpos = 0;
      uint8 mnpos = 0;
      int32 mx = -1000;
      int32 mn = 1000;
      uint8 lookformax = 1;
      uint16 i = 0;
      uint8 delta = 65;


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

      cur = dat[i];
      if (cur > mx)
      {
            mx = cur;
            mxpos = i;
      }

      if ( cur < mn)
      {
            mn = cur;
            mnpos = i;
      }

      if (lookformax)
      {
      if( (cur < (mx-delta)) && ((*cntmn == 0) || (*cntmn > 0 && i >
(minPos[*cntmn-1] + 13))))
            {
                  maxPos[*cntmx] = mxpos;

                  mn = cur;
                  mnpos = i;
                  lookformax = 0;
                  *cntmx = *cntmx+1;
            }
    }
  else
  {
    if (cur > mn+delta)
      {
      minPos[*cntmn] = mnpos;
      mx = cur;
        mxpos = i;
      lookformax = 1;
      *cntmn = *cntmn + 1;
       }
  }
 }
}

void sendUart(uint8 isRed, uint16 dat)
```

```
{
      UART_PutChar(COut[isRed]);
      UART_PutChar(LUTHex[dat >> 12 & 0xf]);
      UART_PutChar(LUTHex[dat >> 8 & 0xf]);
      UART_PutChar(LUTHex[dat >> 4 & 0xf]);
      UART_PutChar(LUTHex[dat & 0xf]);
}

uint16 hammingFilter( uint8 ptr, uint16* datBuf, float* ham,  uint8 cnt,
uint8 isLow)
{
      uint8 ii = 0;
      float tmp = 0;
      uint8 datPtr = ptr - cnt;

      while (ii < 2*cnt+1)
      {
            tmp = tmp + datBuf[datPtr]*ham[ii];
            datPtr++;
            ii++;
      }

      if (isLow)
      {
            return tmp;
      }
      else
      {
            return ((datBuf[ptr] - tmp) + 32768);
      }
}

void CountMax(uint16* dataRed, uint16* dataIR, uint8 count)
{

            uint8 cntMaxRed = 0;
            uint8 cntMaxIR =0;
            uint8 cntMnRed =0;
            uint8 cntMnIR = 0;

            uint8 maxPosRed[128] = {0};
            uint8 maxPosIR[128] = {0};
            uint8 mnPosRed[128] = {0};
            uint8 mnPosIR[128] = {0};

            uint8 ii = 0;
            uint8 beats = 0;
            float hr[128] = {0};
            uint8 hrCount = 0;

            float spO2[128] = {0};
            uint8 lenMnMx = 0;
            uint16 mx[2] = {0};
            uint16 mn[2] = {0};

            int16 medianSpO2 = 0;
            MaxMin(dataRed, count, &maxPosRed, &cntMaxRed, &mnPosRed,
&cntMnRed);
            MaxMin(dataIR, count, &maxPosIR, &cntMaxIR, &mnPosIR, &cntMnIR);

            beats = cntMaxRed + cntMnRed + cntMaxIR + cntMnIR - 4;


            for( ii = 1; ii<cntMaxRed; ii++)
            {
                  hr[hrCount] = 60.0*Fs/(maxPosRed[ii] - maxPosRed[ii-1]);
                  hrCount = hrCount + 1;
            }

            for( ii = 1; ii<cntMnRed; ii++)
            {
                  hr[hrCount] = 60.0*Fs/(mnPosRed[ii] - mnPosRed[ii-1]);
                  hrCount = hrCount + 1;
            }

            for( ii = 1; ii<cntMaxIR; ii++)
            {
                  hr[hrCount] = 60.0*Fs/(maxPosIR[ii] - maxPosIR[ii-1]);
                  hrCount = hrCount + 1;
            }

            for( ii = 1; ii<cntMnIR; ii++)
            {
                  hr[hrCount] = 60.0*Fs/(mnPosIR[ii] - mnPosIR[ii-1]);
                  hrCount = hrCount + 1;
            }


            LCD_ClearDisplay();
            LCD_Position(0,0);
            LCD_PrintString(&hrStr);
            LCD_Position(0, 4);
            LCD_PrintDecUint16(medianVal(&hr, beats));


            lenMnMx = cntMnRed;

            if (cntMnIR < lenMnMx)
            {
                  lenMnMx = cntMnIR;
            }

            for (ii = 0; ii<lenMnMx; ii++)
            {
                  mx[0] = dataRed[maxPosRed[ii]];
                  mn[0] = dataRed[mnPosRed[ii]];

                  mx[1] = dataIR[maxPosIR[ii]];
                  mn[1] = dataIR[mnPosIR[ii]];

                  spO2[ii] = findSpO2(&mx, &mn);
            }
            medianSpO2=  medianVal(&spO2, lenMnMx);

            if (medianSpO2 > 90 && medianSpO2 < 101)
            {
                  LCD_Position(1,0);
                  LCD_PrintString(&spo2Str);
                  LCD_Position(1, 6);
                  LCD_PrintDecUint16(medianSpO2);
            }
}

uint16 medianVal(float* dat, uint8 count)
{
      uint8 ii = 0;
      uint8 jj = 0;
      uint8 ind = 0;

      for (ii = 0; ii<count/2; ii++)
      {
            float tmp = 0;
            ind = 0;
            for(jj = 0; jj< count; jj++)
            {
                  if(dat[jj] > tmp)
                  {
                        tmp = dat[jj];
                        ind = jj;
                  }
            }

            if ( ii < count/2 -1)
            {
                  dat[ind] = 0;
            }
      }

      return dat[ind];
}

float findSpO2(uint16* mx, uint16* mn)
{
      float U[2] = {0,0};
      float iE[4] = { -0.000058206494365309, 0.000386921088344483,
0.000091229451765499, -0.000045840169338513};
      float C1[2] = {0,0};
      float C2[2] = {0,0};
      float tmp1 = 0;
      float tmp2 = 0;

      U[0] = log10((1.0*mx[0])/mn[0]);
      U[1] = log10((1.0*mx[1])/mn[1]);

      C1[0] = iE[0]*U[0] + iE[1]*U[1];
      C1[1] = iE[2]*U[0] + iE[3]*U[1];

      C2[0] = iE[0]*U[1] + iE[1]*U[0];
      C2[1] = iE[2]*U[1] + iE[3]*U[0];
      tmp1 = (C1[0]/(C1[0]+C1[1]))*100;
      tmp2 = (C2[0]/(C2[0]+C2[1]))*100;

      if (tmp2 > tmp1 && tmp2 < 100)
      {
            return tmp2;
      }
      return tmp1;
}
```

Bio-Impedance

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
#include <device.h>
#include <math.h>

int8 readData(uint8 reg);
uint8 sleep_flag;
void writeData(uint8 reg, uint8 dat, uint8 upperFlag);
uint8 Classify(int16* dataReal, int16* dataImag, uint8 freqCount);
void TwoComplement(uint16* dataInBuf, int16* dataOutBuf, uint8 count);
void PrintName(uint8 namePos);
int16 phase(int16 dataReal, int16 dataImag);
int16 magnitude(int16 dataReal, int16 dataImag);

int16 TestDataC[2*3*81] =
{291,293,292,296,302,304,305,307,308,311,312,310,311,313,310,311,309,308,306,
305,302,304,300,297,293,292,292,287,281,275,275,269,267,269,264,261,256,253,2
48,245,240,235,233,233,223,223,216,215,208,205,202,198,194,192,187,184,179,17
8,169,166,165,160,161,154,152,149,146,143,139,138,135,131,130,126,123,120,117
,117,114,121,105,289,290,294,296,300,299,304,305,305,305,307,305,307,308,300,
305,302,300,299,299,300,294,294,294,291,287,287,283,282,272,270,273,268,266,2
63,260,258,254,254,251,247,241,237,238,233,229,228,222,221,215,212,211,209,20
2,200,196,192,189,185,180,178,176,174,169,169,166,160,159,156,159,152,148,144
,143,139,137,140,138,132,124,125,286,287,285,291,293,296,296,298,297,300,299,
301,300,303,301,301,303,302,300,300,298,300,295,295,295,293,288,286,284,283,2
81,279,270,269,269,266,261,258,255,253,248,246,241,237,234,228,229,222,218,21
9,216,210,208,206,204,197,195,190,187,184,183,179,176,172,171,168,166,163,162
,165,156,150,153,148,145,141,143,137,132,128,130,294,294,292,297,301,304,306,
306,306,308,307,308,311,311,307,310,308,306,304,304,301,297,297,299,294,291,2
89,286,283,285,273,271,270,268,267,264,260,253,255,250,246,243,239,236,232,22
8,223,223,221,211,212,207,209,207,199,195,193,191,184,186,178,181,169,170,170
,164,160,158,151,153,150,146,146,138,140,134,128,130,129,123,129,292,293,295,
294,298,298,302,303,300,304,304,305,306,301,303,304,303,300,300,298,299,294,2
```

```
95,293,294,290,285,286,283,274,277,277,269,268,267,263,262,261,254,252,249,24
4,241,240,233,231,228,224,219,220,218,210,207,206,202,198,196,192,189,185,182
,179,178,175,174,170,166,165,163,155,156,153,152,149,143,142,135,138,139,130,
131,285,286,288,290,295,297,300,301,302,304,301,305,305,309,307,305,304,302,3
02,301,299,301,298,297,295,295,294,290,288,291,289,281,284,280,276,273,271,26
5,267,266,260,257,255,253,249,247,241,240,238,233,229,227,225,221,219,213,210
,210,202,201,197,195,191,190,187,182,180,178,174,175,170,168,166,162,160,155,
154,151,149,146,145,290,289,291,292,296,298,299,300,302,304,303,305,304,307,3
03,307,308,307,306,303,303,300,304,303,297,297,294,292,291,292,284,282,280,27
7,275,271,269,266,262,261,254,256,251,248,244,244,237,234,229,228,225,221,218
,214,211,207,204,201,198,194,191,189,184,181,177,175,173,171,166,173,161,158,
156,153,150,149,144,145,138,141,138,287,288,291,293,298,298,300,302,303,304,3
07,305,305,309,306,308,308,311,308,308,304,305,304,302,299,298,297,294,293,29
5,289,286,281,277,278,274,269,266,264,261,255,255,248,247,243,241,239,233,231
,225,222,221,218,212,211,204,202,198,193,191,189,185,181,179,178,175,169,167,
164,168,160,158,152,149,145,144,139,136,134,133,126,285,287,287,292,296,298,3
00,300,301,302,304,307,307,307,306,309,307,306,304,305,304,298,300,300,297,29
3,294,292,290,279,288,285,280,277,272,272,268,269,262,259,252,253,249,248,242
,239,236,232,231,226,223,220,217,211,210,209,203,201,195,191,189,189,182,182,
177,173,171,168,163,170,160,156,155,153,150,144,138,140,138,139,127,288,289,2
90,293,295,298,299,301,301,303,306,306,305,307,307,308,306,305,304,305,304,30
4,299,299,297,295,294,290,287,281,286,277,277,277,273,268,267,261,259,257,253
,250,249,244,241,238,233,231,228,224,218,218,216,211,209,205,203,196,194,191,
186,186,184,179,172,174,169,167,164,169,158,154,149,150,145,142,138,136,136,1
36,129,343,337,339,344,342,346,340,339,343,337,338,335,331,326,319,317,304,29
6,289,287,268,262,252,248,235,220,214,207,192,181,169,173,151,145,130,121,115
,98,90,76,62,58,46,37,24,23,23,20,17,14,1,-13,-30,-40,-53,-58,-71,-81,-86,-
89,-98,-104,-107,-117,-120,-119,-123,-120,-122,-122,-125,-121,-125,-119,-
120,-119,-126,-111,-113,-122,-
110,343,344,348,354,361,358,365,362,361,357,355,356,342,344,336,326,311,303,3
02,295,278,262,258,249,231,220,214,197,185,170,162,149,142,135,124,113,103,88
,78,74,57,46,33,26,20,11,-2,-5,-15,-23,-31,-41,-44,-53,-50,-58,-67,-67,-73,-
74,-74,-75,-79,-81,-87,-88,-84,-91,-90,-76,-92,-89,-91,-93,-84,-89,-87,-86,-
85,-74,-
76,336,338,336,343,351,349,343,346,346,347,342,340,338,339,322,324,315,304,29
8,294,286,275,264,267,246,240,222,212,198,175,176,160,149,130,119,104,95,76,6
9,57,49,44,30,20,10,-1,-6,-16,-31,-37,-45,-52,-55,-67,-67,-73,-80,-85,-87,-
95,-99,-98,-101,-105,-113,-114,-110,-113,-120,-116,-120,-120,-117,-119,-121,-
124,-110,-109,-109,-116,-
113,351,351,353,357,362,366,361,360,355,351,349,344,332,324,318,317,296,291,2
89,278,269,260,251,244,230,220,212,203,186,167,167,164,143,137,123,112,96,92,
79,71,56,53,42,29,24,14,10,1,-1,-8,-21,-33,-27,-39,-38,-46,-49,-51,-59,-63,-
64,-67,-73,-76,-81,-76,-78,-82,-79,-77,-85,-84,-81,-85,-81,-88,-87,-75,-77,-
77,-
80,339,346,347,357,353,356,356,358,354,352,343,342,328,328,316,311,304,293,28
6,279,269,258,255,242,228,223,207,198,192,175,165,156,145,137,125,114,102,96,
80,78,64,54,41,35,24,19,6,-2,-9,-12,-19,-33,-41,-42,-48,-58,-63,-69,-77,-82,-
87,-87,-92,-94,-98,-97,-102,-102,-103,-94,-110,-109,-110,-106,-108,-107,-
106,-105,-106,-93,-103};
int16 TestDataS[2*3*81] =
{55,54,57,60,67,68,72,74,80,88,88,88,99,102,105,105,115,111,118,125,127,131,1
33,136,140,143,147,148,149,153,162,161,162,164,165,167,169,167,171,170,170,17
7,176,178,180,174,177,176,175,169,181,178,176,172,175,177,174,173,172,168,166
,164,166,162,161,161,156,154,152,147,149,146,140,142,144,136,135,130,131,129,
120,43,46,50,50,56,62,62,65,73,77,77,80,86,87,89,92,97,101,103,108,111,112,11
5,116,120,125,126,128,134,138,136,135,141,143,141,145,147,150,145,151,149,151
,154,154,152,151,154,153,151,157,154,156,153,151,152,154,151,152,155,149,148,
142,141,145,144,142,139,141,137,141,131,134,129,129,126,124,120,120,118,114,1
20,42,42,45,47,53,54,57,58,65,66,71,72,77,79,85,83,90,95,97,105,106,113,112,1
17,121,124,128,130,134,137,135,134,141,142,143,147,149,148,149,154,154,153,14
9,152,155,152,155,154,154,151,155,156,152,158,156,157,155,152,152,156,145,151
,145,145,142,139,138,140,132,132,131,131,127,127,126,122,118,120,120,114,123,
54,53,56,58,63,64,66,68,79,79,83,84,92,94,96,98,106,107,111,113,119,121,126,1
28,130,132,135,135,138,141,140,142,148,149,149,152,152,155,155,158,159,157,16
0,158,160,161,161,160,164,165,165,163,163,161,163,161,159,161,164,154,158,156
,154,153,149,150,147,147,147,140,144,142,139,137,136,130,130,126,131,131,120,
48,51,48,50,57,60,64,65,71,73,75,77,84,85,88,91,94,99,101,103,108,112,112,117
,120,121,124,129,129,131,131,131,138,141,142,147,149,150,153,155,154,156,155,
149,157,153,157,158,155,156,156,158,156,157,158,156,157,155,152,150,151,146,1
46,142,142,142,136,139,133,123,131,130,133,128,126,127,120,117,117,125,112,37
,40,42,44,52,55,55,56,60,66,68,69,74,76,77,83,88,92,93,96,97,102,106,109,111,
113,119,117,121,123,124,127,135,135,137,140,140,145,145,144,149,149,151,152,1
55,150,152,155,156,157,156,159,156,158,157,158,156,157,160,156,154,153,155,15
4,154,152,151,151,151,142,146,148,146,142,145,139,140,137,135,128,130,46,48,4
9,52,54,57,57,60,68,67,72,74,79,80,87,88,93,96,101,103,108,111,113,117,119,12
0,125,127,130,134,134,135,139,142,143,146,146,148,151,152,154,155,159,158,160
,159,161,162,165,161,162,166,165,165,166,166,167,163,165,165,162,165,163,163,
162,160,159,159,155,150,152,150,149,148,147,144,141,140,141,137,133,42,44,48,
49,55,56,60,59,65,70,76,76,81,84,87,90,96,100,103,107,109,112,115,120,121,125
,131,131,138,139,142,146,149,148,152,151,154,156,158,160,160,161,160,165,165,
164,166,167,170,169,169,172,171,169,170,169,169,167,168,167,166,167,164,164,1
60,164,161,162,158,158,157,154,151,151,150,148,147,148,142,134,135,42,43,45,4
7,54,52,57,61,67,69,73,76,82,83,87,91,97,98,101,104,109,112,116,119,122,123,1
27,129,131,137,136,141,144,145,147,149,151,153,155,155,157,157,159,160,162,16
3,162,165,164,162,163,165,163,162,163,163,162,161,163,160,160,158,160,157,155
,155,151,152,151,154,149,149,146,145,143,140,139,138,137,127,132,45,46,50,51,
56,57,61,61,69,69,75,76,82,82,89,91,96,99,104,104,111,113,117,117,124,126,129
,131,133,137,136,139,142,145,147,148,151,155,155,155,159,158,159,160,160,162,
163,162,162,168,165,166,165,164,165,164,164,163,165,161,160,160,161,158,159,1
57,154,154,151,150,150,146,147,145,144,138,135,136,138,127,127,150,156,159,15
9,180,182,190,195,217,226,233,250,265,261,279,289,299,306,307,305,317,324,326
,333,336,336,338,341,342,344,350,352,354,349,349,348,345,334,338,341,332,323,
318,321,309,303,282,278,272,265,271,275,262,247,252,244,238,228,226,212,200,1
89,185,177,175,160,148,138,131,135,116,101,88,80,73,59,49,44,35,27,23,146,155
,166,177,198,206,215,224,252,262,271,278,298,299,316,323,339,340,344,348,351,
350,365,368,368,371,377,373,380,369,377,381,371,371,371,367,365,362,354,362,3
45,342,331,332,321,311,307,298,297,286,272,270,262,254,246,233,228,217,210,19
9,183,173,166,157,148,135,122,111,105,95,88,73,75,56,47,42,25,26,15,8,0,135,1
43,148,154,164,172,180,185,205,212,219,233,249,250,257,268,283,286,297,300,31
6,315,321,330,335,337,349,348,355,372,357,367,354,362,360,362,359,346,345,329
,335,333,328,326,315,304,308,295,290,276,268,261,252,247,238,227,222,210,198,
194,181,175,172,159,145,138,135,120,111,111,88,83,78,63,56,44,26,19,16,12,-
1,159,167,173,179,198,198,209,222,242,244,250,261,274,277,286,286,304,304,308
,316,321,331,329,334,342,346,345,353,354,346,363,364,351,356,353,352,344,343,
340,332,332,321,318,322,310,294,297,289,284,272,260,265,254,238,234,225,214,2
04,202,187,177,173,157,149,135,127,126,112,100,86,80,70,65,53,42,40,20,23,8,-
4,-
11,151,158,167,175,188,203,213,217,240,243,246,258,270,275,278,286,295,304,31
1,316,320,328,332,332,338,340,347,343,348,348,341,355,352,349,346,347,343,338
,336,330,329,322,317,316,308,295,292,285,281,268,264,257,252,243,233,228,219,
206,200,190,179,170,160,153,145,140,122,124,108,96,91,91,75,65,58,48,39,28,24
,14,13};

int16 realData[101] = {0};
```

```
int16 imagData[101] = {0};
int16 twoCompRealData[80] = {0};
int16 twoCompImagData[80] = {0};
uint8 persons = 3;
uint8 trials = 5;
uint8 frequencies = 81;
char Names[] = "BradRojaMikeShad";
char LUTHex[] = "0123456789ABCDEF";
char IM[4] = "IRMP";
void main()
{
      int8 flag;
      uint16 count = 0;
      uint8 start = 0;
      uint8 j = 0;
      uint8 k = 0;
      I2C_1_MasterClearStatus(); /* Clear any previous status */
      I2C_1_Start();
      I2C_1_EnableInt();
      Timer_sleep_Start();
      sleep_ISR_Start();
      UART_Start();
      SW_Start();
      LCD_Start();
      int16 testReal[80] =
{35,37,43,42,41,46,47,44,44,45,46,47,49,44,47,43,43,44,39,40,42,40,42,39,35,3
6,33,32,22,27,27,25,24,24,19,16,17,12,7,9,6,5,3,0,3,-2,-8,-11,-6,-10,-20,-
20,-22,-24,-29,-29,-36,-35,-40,-44,-45,-45,-50,-53,-52,-57,-57,-61,-61,-67,-
67,-77,-72,-79,-76,-77,-85,-84,-87,-83};
      int16 testImag[80] =
{45,47,46,53,53,54,52,59,60,61,62,68,69,70,72,72,79,78,81,82,88,89,93,95,94,1
03,101,104,101,107,115,115,114,120,118,121,119,124,130,130,133,130,134,139,13
5,142,140,137,142,139,146,143,147,143,146,142,142,150,147,143,144,143,145,146
,145,141,145,139,123,140,139,138,134,133,130,135,135,131,141,133};
      uint16 fakeArray[10] = {65185, 65267, 65270, 0, 8, 23, 23, 27, 32, 35};
      int16 fakeTwos[10] = {0};
      //I2C_1_MasterWriteBuf(0x0d,0x82 , 12, I2C_1_MODE_COMPLETE_XFER);

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */

    CyGlobalIntEnable; /* Uncomment this line to enable global interrupts. */
      SW_StartConvert();
      while(1)
    {
            SW_StartConvert();
            if(SW_IsEndConversion(0))
            {
                  start = 255;//SW_GetResult8();
                  SW_StopConvert();
            }

      if (start > 127)
      {
            //set range 2 and PGA gain 1 : 00000111
            writeData(0x80, 0x07, 0);
            //LCD_ClearDisplay();
            LCD_Position(0,0);
```

```
LCD_PutChar(LUTHex[11]);
j = 0;
k = 0;
count = 0;

//set settling cycles
writeData(0x8a, 0x07, 0);
writeData(0x8b, 0xff, 0);


//start frequency of 1kHz
writeData(0x82, 0, 0);
writeData(0x83, 0x83, 0);
writeData(0x84, 0x12, 0);

//increment by 1 kHz
writeData(0x85, 0, 0);
writeData(0x86, 0x83, 0);
writeData(0x87, 0x12, 0);

//points in frequency sweep (100)
writeData(0x88, 0, 0);
writeData(0x89, 0x64, 0);


//standby '10110000'
writeData(0x80, 0xb0, 1);


//initialize sweep
writeData(0x80, 0x10, 1);

//sleep
if (sleep_flag == 1)
{
      sleep_flag = 0;
      Timer_sleep_ReadStatusRegister();
}

//start sweep
writeData(0x80, 0x20, 1);

flag = 0;
while((flag & 0x04) != 4)
{
//if(I2C_1_MasterStatus() & I2C_MSTAT_WR_CMPLT)
      while ((flag & 0x06) == 0)
      {
            //poll for flag
            flag =readData(0x8f);

      }

//valid data 00000010
if ((flag & 0x02) == 2)
{
      uint16 tmpl;
```

```
                  uint16 tmpu;

                  //real data
                  tmpu = readData(0x94);
                  tmpl = readData(0x95);

                  realData[count] = (tmpu<<8) + tmpl +256;
                  /*if ((count<91) && (count >= 10))
                  {
                        TwoComplement(&realData[count], &twoCompRealData[j],
1);
                        //twoCompRealData[k] = realData[count];
                        j++;
                  }*/


                  //imag data
                  tmpu = readData(0x96);
                  tmpl = readData(0x97);

                  imagData[count] = (tmpu<<8) + tmpl;

                  if ((count<91) && (count >= 10))
                  {
                        TwoComplement(&realData[count], &twoCompRealData[k],
1);
                        TwoComplement(&imagData[count], &twoCompImagData[k],
1);
                        sendUart(0, twoCompImagData[k]);
                        sendUart(1, twoCompRealData[k]);
                        sendUart(2, magnitude(twoCompRealData[k],
twoCompImagData[k]));
                        sendUart(3, phase(twoCompRealData[k],
twoCompImagData[k]));

                        k++;
                  }

                  /*sendUart(0, imagData[count]);
                        sendUart(1, realData[count]);
                        sendUart(2, magnitude(realData[count],
imagData[count]));
                        sendUart(3, phase(realData[count], imagData[count]));

                  */

                  count = count + 1;

                  //increment count
                  writeData(0x80, 0x30, 1);
                  LCD_ClearDisplay();

                  //kill flag
                  //flag = flag & 0x02;
                  flag =readData(0x8f);
            }
            if ((flag & 0x04) == 4)
            {
                  //power down
                  writeData(0x80, 0xa0, 0);
            }
            }




            }
                  //PrintName(Classify(testDataRealMike,
testDataImagMike,80));
                  PrintName(Classify(twoCompRealData, twoCompImagData,81));


      } //while loop

} //main loop


int8 readData(uint8 reg)
{
      int8 tmp;
      I2C_1_MasterSendStart(0x0d, 0);
      I2C_1_MasterWriteByte(0xb0);
      I2C_1_MasterWriteByte(reg);
      I2C_1_MasterSendStop();

      I2C_1_MasterSendStart(0x0d, 1);
      tmp = I2C_1_MasterReadByte(0);
      I2C_1_MasterSendStop();

      return tmp;
}

void writeData(uint8 reg, uint8 dat, uint8 upperFlag)
{
            if (upperFlag == 1)
            {
                  int8 temp;
                  temp = readData(reg);
                  dat = (temp&0x0F) | dat;
            }

            I2C_1_MasterSendStart(0x0d, 0);
            I2C_1_MasterWriteByte(reg);
            I2C_1_MasterWriteByte(dat);
            I2C_1_MasterSendStop();
}

void TwoComplement(uint16* dataInBuf, int16* dataOutBuf, uint8 count)
{
      uint8 i = 0;
      uint16 tmp = 0;
      for (i=0; i<count; i++)
      {
            tmp = dataInBuf[i];
```

```c
            if (tmp > 32767)
            {
                  dataOutBuf[i] = tmp - 65536 +256;
            }
            else
            {
                  dataOutBuf[i] = tmp;
            }

      }

}


int16 magnitude(int16 dataReal, int16 dataImag)
{

      float tmp = pow(pow(dataReal,2)+pow(dataImag,2), 0.5);
      return tmp;
}

int16 phase(int16 dataReal, int16 dataImag)
{
      float tmp = atan(1.0*dataImag/dataReal);
      return tmp;
}

void sendUart(uint8 isReal, int16 dat)
{
      UART_PutChar(IM[isReal]);
      UART_PutChar(LUTHex[dat >> 12 & 0xf]);
      UART_PutChar(LUTHex[dat >> 8 & 0xf]);
      UART_PutChar(LUTHex[dat >> 4 & 0xf]);
      UART_PutChar(LUTHex[dat & 0xf]);

}

uint8 Classify(int16* dataReal, int16* dataImag, uint8 freqCount)
{

      uint8 p = 0;
      uint8 t = 0;
      uint8 f = 0;
      float tmpD = 0;
      float tmpP = 0;
      int16 testC = 0;
      int16 testS = 0;
      uint8 d = 0;
      uint8 k = 5;
      float shortestDist[5] = {1e9, 1e9, 1e9, 1e9, 1e9};
      uint8 shortestDistPer[5] = {0};
      uint8 personCount[4] = {0,0,0,0};
      int8 ii = 0;
      uint8 tmp = 0;

      for (p = 0; p<persons; p++)
      {
            for(t =0; t<trials; t++)
            {

                  tmpD = 0;
                  for(f = 0; f<frequencies; f++)
                  {

                        testC = dataReal[f];
                        testS = dataImag[f];

                        tmpP = pow((testC - TestDataC[p*frequencies*trials +
t*frequencies + f]),2) + pow((testS - TestDataS[p*frequencies*trials +
t*frequencies + f]),2);
                        tmpD = tmpD + tmpP;

                  }

                  tmpP = pow(tmpD, 0.5);

                  d = 0;
                  while( d < k && shortestDist[d] < tmpP)
                  {
                        d = d + 1;
                  }

                  if ( d < k)
                  {
                        for (ii = k-1; ii > d; ii--)
                        {
                              shortestDist[ii] = shortestDist[ii-1];
                              shortestDistPer[ii] = shortestDistPer[ii-1];
                        }

                        shortestDist[d] = tmpP;
                        shortestDistPer[d] = p;
                  }
            }

      }

      for(ii = 0; ii<k; ii++)
      {
            tmp = shortestDistPer[ii];
            personCount[tmp] = personCount[tmp]+1;
      }

      tmp = 0;

      for(ii = 1; ii<persons; ii++)
      {
            if(personCount[ii] > personCount[tmp])
            {
                  tmp = ii;
            }
      }

      return tmp;
```

```c
}

void PrintName(uint8 namePos)
{
      LCD_ClearDisplay();
      LCD_Position(1,0);
      LCD_PutChar(Names[namePos*4]);
      LCD_PutChar(Names[namePos*4 + 1]);
      LCD_PutChar(Names[namePos*4 + 2]);
      LCD_PutChar(Names[namePos*4 + 3]);
}

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

MATLAB Classifier

```matlab
%assume training data is such that each person has 10 data points in order
%in two matrices (within each is 10 frequencies in order)

freqs = 80;
trials = 5;
k = 5;
persons = 3;

shortestDist = 1e6*ones(k,1);
shortestDistPer = 1e6*ones(k,1);

if 1
    % assume real/imag data is an amplitude/phase
    TestDataC = zeros(freqs*trials*persons,1);
    TestDataS = zeros(freqs*trials*persons,1);


    % Real Data
    nam = {'bd', 'rj', 'mk'};

    count = 1;
    for ii = 1:length(nam)
        name = nam{ii};

        for jj = 1:trials
            dati = load([name 'Imag' num2str(jj) '-T']);
            datr = load([name 'Real' num2str(jj) '-T']);

            TestDataC(count:count+freqs-1) = datr.dataReal(1:80);
            TestDataS(count:count+freqs-1) = dati.dataImag(1:80);

            count = count + freqs;
        end

    end
```

```
end

%real data
%load 'bradImag5'
%load 'bradReal5'
datC = dataReal(1:80);
datS = dataImag(1:80);



if 1
    %% print to file
    realDataStr = num2str(TestDataC(1));
    imagDataStr = num2str(TestDataS(1));
    for ii = 2:length(TestDataC)
        realDataStr = [realDataStr ',' num2str(TestDataC(ii))];
        imagDataStr = [imagDataStr ',' num2str(TestDataS(ii))];
    end

    fileID = fopen('realTestData.csv','w');
    fprintf(fileID,'%s',realDataStr);
    fclose(fileID);

    fileID = fopen('imagTestData.csv','w');
    fprintf(fileID,'%s',imagDataStr);
    fclose(fileID);



    testRealDataStr = num2str(datC(1));
    testImagDataStr = num2str(datS(1));

    for ii = 2:length(datC)
        testRealDataStr = [testRealDataStr ',' num2str(datC(ii))];
        testImagDataStr = [testImagDataStr ',' num2str(datS(ii))];
    end

    fileID = fopen('classifyRealTestData.csv','w');
    fprintf(fileID,'%s',testRealDataStr);
    fclose(fileID);

    fileID = fopen('classifyImagTestData.csv','w');
    fprintf(fileID,'%s',testImagDataStr);
    fclose(fileID);
end
% %test data
% v = 1:10;
% u = 7:16;
% TestDataC = [repmat(v', 10,1) + randn(100,1)*1; repmat(2*v', 10,1)+
randn(100,1)*1; repmat(3*v', 10,1)+ randn(100,1)*1;repmat(4*v', 10,1)+
randn(100,1)*1];
% TestDataS = [repmat(u', 10,1)+ randn(100,1)*1; repmat(2*u', 10,1)+
randn(100,1)*1; repmat(3*u', 10,1)+ randn(100,1)*1;repmat(4*u', 10,1)+
randn(100,1)*1];
%
% datC = zeros(freqs,1);
% datS = zeros(freqs,1);
```

```
%
% %test data
% v = 3*(1:10);
% u = 3*(7:16);
% datC = v' + randn(10,1)*1;
% datS = u' + randn(10,1)*1;

personCount = zeros(persons,1);

for p = 1:persons

    for t = 1:trials
        tmpD = 0;
        for f = 1:freqs
            testC = datC(f);
            testS = datS(f);

            tmpD = tmpD + (testC - TestDataC((p-1)*freqs*trials + (t-1)*freqs
+ f))^2 + (testS - TestDataS((p-1)*freqs*trials + (t-1)*freqs + f))^2;

        end
        D = sqrt(tmpD);

        d = 1;
        while d <= k && shortestDist(d) < D
            d= d+ 1;
        end


        if d <= k
            for ii = k:-1:d+1
                shortestDist(ii) = shortestDist(ii-1);
                shortestDistPer(ii) = shortestDistPer(ii-1);
            end
            %disp(D)
            shortestDist(d) = D;
            shortestDistPer(d) =  p;
        end
    end

end

%count person
for ii = 1:k
    tmp = shortestDistPer(ii);
    personCount(tmp) = personCount(tmp) + 1;
end

%get winner
classifier = 1;
for ii = 2:persons
    if ( personCount(ii) > personCount(classifier))
        classifier = ii;
    end
end
```

```
%disp winner
disp(classifier)
```

---

i http://www.ti.com/lit/an/sboa060/sboa060.pdf