

SOLDIER STATUS MONITORING PROJECT

By

Sanghee Seo

Santhosh Vairavan

Yash Kulkarni

Final Report for ECE 445, Senior Design, Spring 2013

TA: Lydia Majure

01 May 2013

Project No. 25

Abstract

Soldier Status Monitor Project is a MIT Lincoln Labs sponsored project on the development of a portable device capable of continuous monitoring of body vitals. Important information such as body skin temperature and heart rate in beats per minute along with data of soldier motion is collected from calibrated sensors and transmitted to offsite remote location. The device comes with SD card slot for onsite capturing of data before transmission. The sponsors of this project are mainly interested in using this device to monitor soldiers in the field. The device therefore makes use of slim lithium polymer batteries that is used to capture vitals data and transmit it once every 15seconds. Lastly, the device also comes with a display system using LEDs to indicate battery level thresholds to require charging, vitals threshold of skin temperature and heart rate reaching undesirable values.

Contents

1. Introduction	1
2 Design.....	2
2.1 Overall Block Diagram	4
2.1.1 Sensor Module Block Diagram	5
2.1.2 Threshold Display Module Block Diagram	6
3. Design Verification	9
3.1 Level 1: Individual Component Testing.....	9
3.1.1 Pulse Sensor	9
3.1.2 Skin Temperature Sensor.....	10
3.1.3 Motion Sensor.....	10
3.2 Level 2: Individual Component Testing with Arduino	12
3.2.1 Pulse Sensor	12
3.2.2 Skin Temperature Sensor.....	13
3.2.3 Motion Sensor.....	14
3.2.4 SD/MMC Card Holder	14
3.2.5 Xbee Wireless Transmission Module	15
3.3 Level 3: Final Product Testing	16
4. Costs.....	17
4.1 Parts	17
4.2 Labor	18
5. Conclusion.....	19
5.1 Accomplishments.....	19
5.2 Uncertainties.....	19
5.3 Ethical considerations	19
5.4 Future work.....	20
References	21
Appendix A Open Source Individual Sensor Codes with Arduino	22
Appendix B Arduino Final Code	26
Appendix C Requirement and Verification Table.....	32

1. Introduction

The goal of this project is to build a portable device capable of monitoring body vitals and transmitting the information to an offsite remote location. The sponsors of this project were mainly interested in using this device to monitor soldiers out in the field.

i. Goals

1. Be able to gather heart rate, body temperature and movement data reliably.
2. Wireless communication of the sensor data to a central location at least 100 meters away.
3. Data Analysis software at remote location
4. Continuous monitoring of the soldiers vitals for at least 24 hours.

ii. Features

1. A portable and lightweight device that is easy to carry
2. LED lighting to display when user vitals reach critical threshold.
3. Non-invasive monitoring of soldier
4. Onsite storage of data on external SD/MMC card

iii. Project Subsections

We identified the following main focus areas for this project:

1. Identifying required sensors and its connections with the Arduino Uno microcontroller and designing custom PCB.
2. Data Packaging and Wireless Communication over 100 to 500m
3. Software Coding for analyzing and displaying received sensor data.

With the help of the sponsors we set two project goals:

1. Create a prototype device using identified individual components connecting to the MegaPro board using a breadboard.
2. Create a completely customized board for the device thereby making it even more compact and portable.

2 Design

Procedure

Our first thoughts for designing this device were to use a set of sensors that outputted an analog signal that we would use to convert into a data that was useful. We decided that a microcontroller was needed in this aspect to do the analog to digital conversion.

We wanted this to be easy to use and implement so we did not design the sensors. Our group decided to use off the shelf components that were tried and tested to get going. The goal was to get a prototyped board quickly as possible so that we could create our own board with a microcontroller chip. We wanted to create our own board because, to make this device feasible we needed it to be small, portable and wearable.

For our sensors we decided to use an infrared ir temperature sensor to measure skin temperature, a pulse sensor that could be attached to the earlobe or the fingertip and an accelerometer to measure acceleration. We chose these three sensors because they were the least non-invasive. Strapping the device to the body should be simple yet the device should also give reliable data too. We took those design parameters into consideration when choosing the sensors.

We chose a 2000 mAh 3.7 V lithium polymer battery because our sensors and the microcontroller needed at least a 3.3 V to be powered. We calculated that for at least 12 hours of use we needed 1000 mAh. Our device used 60 mAh and at 2000 mAh we get at least 24 hours of life using the following equation.

$$\text{Battery Life} = \frac{\text{Battery Capacity in Milli amps per hour}}{\text{Load Current in Milli Amps per hour}}$$

$$\text{Battery Life} = \frac{2000}{60} = 23.333 \text{ hours}$$

We thought double the capacity would allow for any unforeseen usages or drains on power. Plus we were only sending and sensing data every 15 seconds, so this would give us extra battery life too.

We used the xbee transceiver for transmitted and receiving data because it was a reliable and documented component that was easy to use.

Initially we started with the atmega2560 but switched to the atmega328 chip because of ease of use to prototype. The atmega2560 chip is only surface mount chip and we didn't know how to prototype this chip using a breadboard, but the atmega328 could attach to a breadboard and fit all our requirements.

And finally we wanted to store the data locally so that if something happened on the transmitting end and the receiving location did not get data at least there was a local backup that could be uploaded at a later point in time

Alternatives

There were certain design alternatives that were suggested as part of this project. The accelerometer while able to measure acceleration did not measure movement which is what we needed. In addition to the accelerometer if we added a gyroscope, magnetometer, and GPS we could get verified movement data.

Also the atmega328 chip was very limited in number of inputs. If we wanted to add more sensors we would have had to change the chip. In this aspect the atmega2560 chip would be recommended but again prototyping with this chip would be hard.

Additionally we thought that adding a thermistor to the ir temperature sensor would give us a better reading on body temperature. We could average both sensors to get a better temperature reading. Plus adding another sensor would give us a built in redundancy.

We also thought that adding an ECG to the pulse sensor would also give us a better reading on heart rate. The pulse sensor takes time to calculate the heart rate. With two heart rate sensors we could get a more reliable reading and also a built in redundancy if anything should happen to either sensor.

2.1 Overall Block Diagram

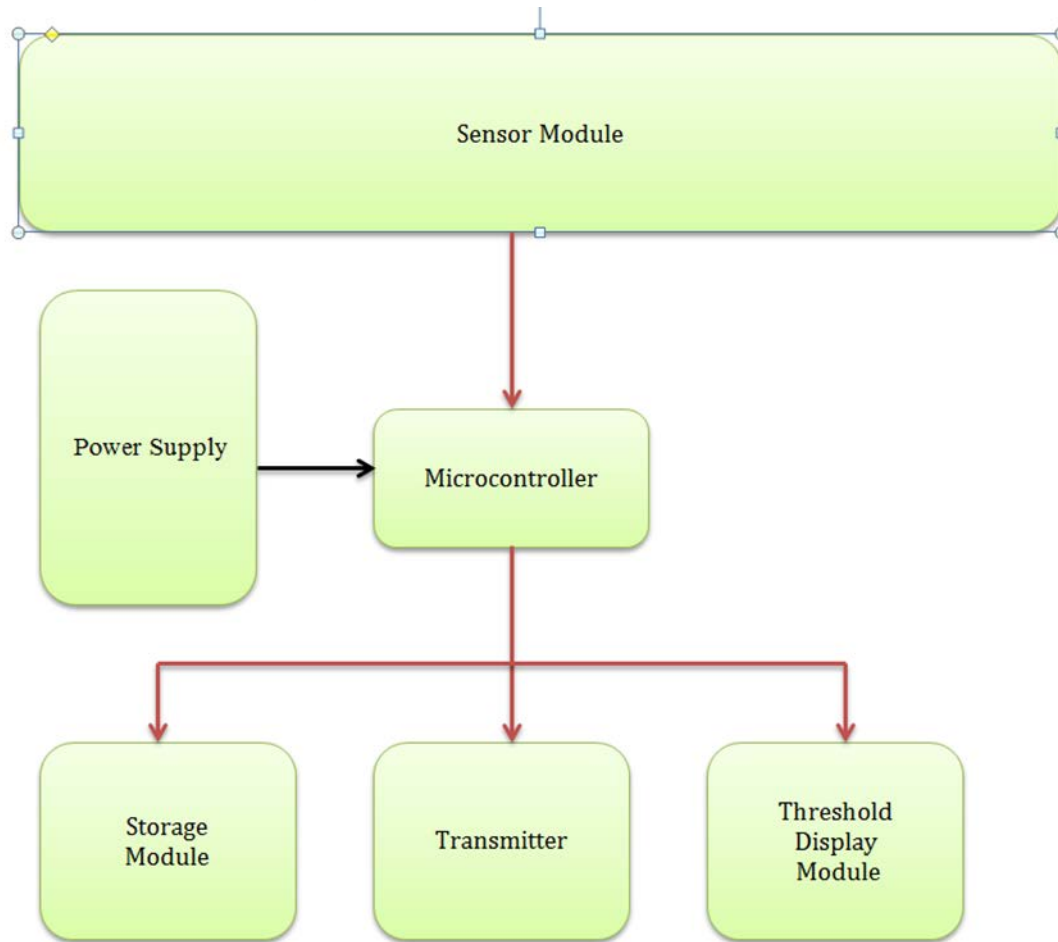


Figure 2.1 Overall Device Block Diagram

Sensor Module:

This module consists of three sensors that will monitor soldier body vitals. The sensors used are commercial off-the shelf sensors.

Microcontroller:

We are using an Arduino compatible microcontroller board: **Atmege328 chip**. Functions of the microcontroller is receiving data from sensors, packaging them and providing it to the transmitter for wireless communication. The arduino microcontroller will do an analysis on the vitals to see if they pass critical thresholds for each sensor and accordingly indicate on threshold display module.

Power Supply Module:

This module needs to be able to power the sensors, the microcontroller, the transmitter, and the display modules. We will make use of a **2000mAh Lithium Polymer battery**. It is a very slim, extremely

lightweight battery that outputs a nominal 3.7V at 2000mAh. Since our microcontroller and other sensors are running at 3.3V, this choice of battery is most suitable for fulfilling project goals of device being lightweight as well as a runtime of 24hours.

Storage Module

This module is responsible for storing all the data acquired from the sensors. We are incorporating a breakout board for SD-MMC cards with The Mega Pro board. The microcontroller will store the packaged data from all sensors into the SD card just before transmitting to base station via the transmitter.

Transceiver (XBee Pro 60mW Wire Antenna - Series 1 (802.15.4))

XBee-PRO products are engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. IEEE 802.15.4 standard offers the fundamental lower network layers. The modules require minimal power and provide reliable delivery of data between devices. The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other. One Xbee will be attached to the microcontroller packaging the data for transmission to be received at the computer side.

Threshold Display Module

This module will indicate to the soldier when certain vitals cross limit so the soldier knows what is going on in his body.

2.1.1 Sensor Module Block Diagram

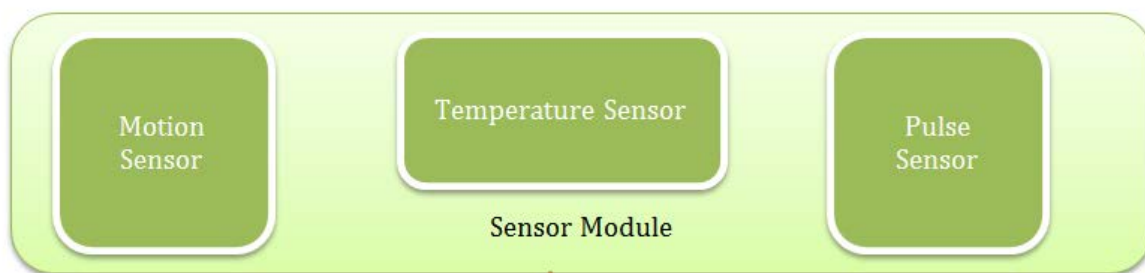


Figure 2.2 Sensor Module

1. Skin Temperature Sensor:

We will make use of a non-contact temperature sensing infrared thermometer-**MLX90614** to measure body temperature. This sensor gives an average temperature of all objects in its Field of View. This is an I²C device, which allows the sensor to output literal information with a 17bit resolution instead of giving a voltage reading.

Our second choice was the one-wire digital temperature sensor- **DS18B20**. While this sensor is also capable of outputting digital data of 9 to 12bit resolution, it measures ambient temperature around sensor

and therefore produced poor body temperature readings when held near skin and was discarded for this project.

2. Pulse Sensor:

We will make use of the **Pulse Sensor Amped** to monitor heart rate data. It combines a simple optical heart rate sensor with amplification and noise cancellation circuitry. The device will be connected to the ear lobe and will acquire heart rate in beats per minute.

3. Motion Sensor:

We will make use of the triple axis accelerometer- **ADXL335**. Placed vertical so that z-axis is always aligned with forward motion of soldier, the sensor will output the acceleration of the soldier.

2.1.2 Threshold Display Module Block Diagram

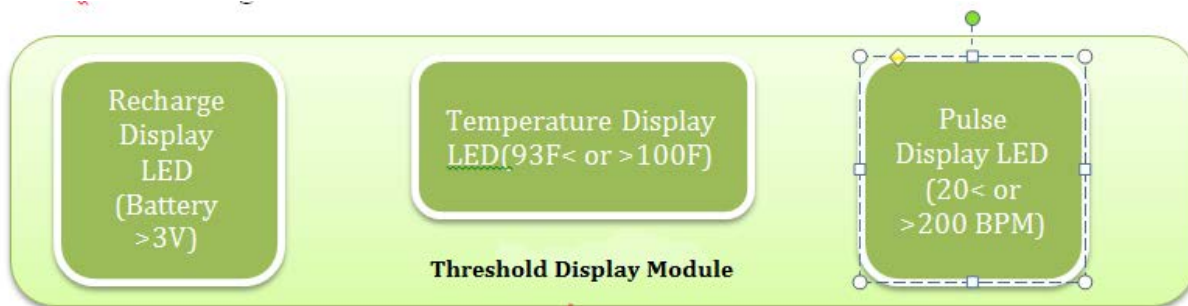


Figure 2.3 Threshold Display Module

Threshold Display Module:

This display module will be present on the soldier to indicate important information. We will make use of three different colored LEDs to indicate body vitals crossing a pre-set threshold value or if the battery does not have sufficient charge. The LEDs are controlled by the microcontroller which is responsible for checking if input data from sensors and battery monitoring has crossed threshold values

LED COLOR	Function
RED	Indicates when pulse rate crosses 220 bpm or below 20 bp,
GREEN	Indicates body temperature crossing 100°F or below 93° F
BLUE	Indicates when battery voltage falls below 3V

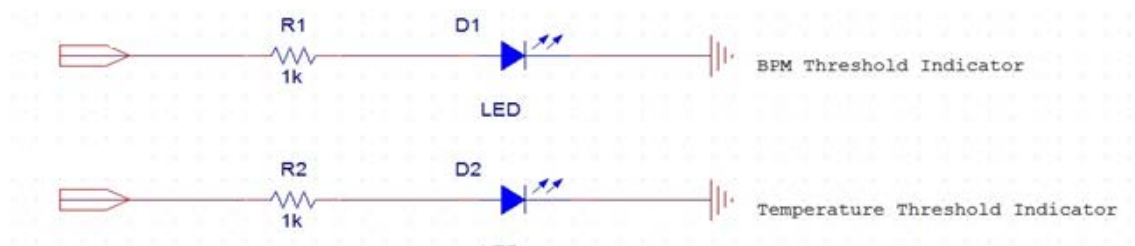


Figure 2.4. Schematic for BPM and Temperature Threshold Display

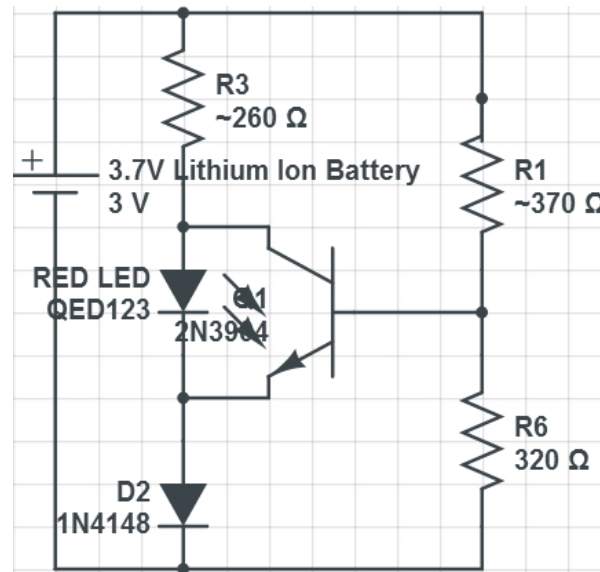


Figure 2.5 Battery Status Circuit

Battery Status Threshold Circuit

This circuit is responsible for indicating a low status of battery. A full battery is around 3.7v. When the battery goes below 3.0v, the red LED illuminates as an indication of the low status of the battery.

Verification

1. LED does not light when battery is above 3.0v

- The trigger voltage = $V_{trig} = V_{on}(D2) + V_{beon} = 1.4v$
- When $V_{cc} > 3.0v$, by the voltage divider rule,
- $V_b = V_{cc} \times (R2 / (R1 + R2))$
- $= 3.7 \times (370 / (690))$

- $= 1.98\text{v} > V_{\text{trig}} = 1.4\text{v}$

Since $V_b > V_{\text{trig}}$, BJT turns on and the current does not go through red LED.

2. LED lights when battery is below 3.0v

- When $V_{\text{cc}} < 3.0\text{v}$, $V_b < V_{\text{trig}}$

Therefore, BJT turns off and current flows through red LED and illuminates.

3. Design Verification

Verification Process is broken up into three levels from testing individual components to successful testing of final product.

3.1 Level 1: Individual Component Testing

At this stage, individual sensor testing is given importance to check for accuracy and compatibility verifications.

3.1.1 Pulse Sensor

- **INPUT:** 3 – 5V
- **OUTPUT:** Single Wire output is an analog fluctuation in voltage.

Simulations:

We breadboard the pulse sensor to a DC power supply outputting 5V and connect the pulse sensor analog output pin to an oscilloscope. Our goal is to find successive moments of instantaneous heart beat from graphical display on oscilloscope. We then measure the time between these moments, called the Inter Beat Interval (IBI). Heart Rate (HR) is the interval between successive heart beats (I.B.I.).

IBI is inversely related to HR by the equation

$$HR = 60000 / IBI ;$$

where IBI is calculated in milliseconds.

The two figures below show pulse spikes at regular intervals corresponding to regular heart beats. Using the above equation we can see that Heart Rate is between 60 to 100bpm as desired.

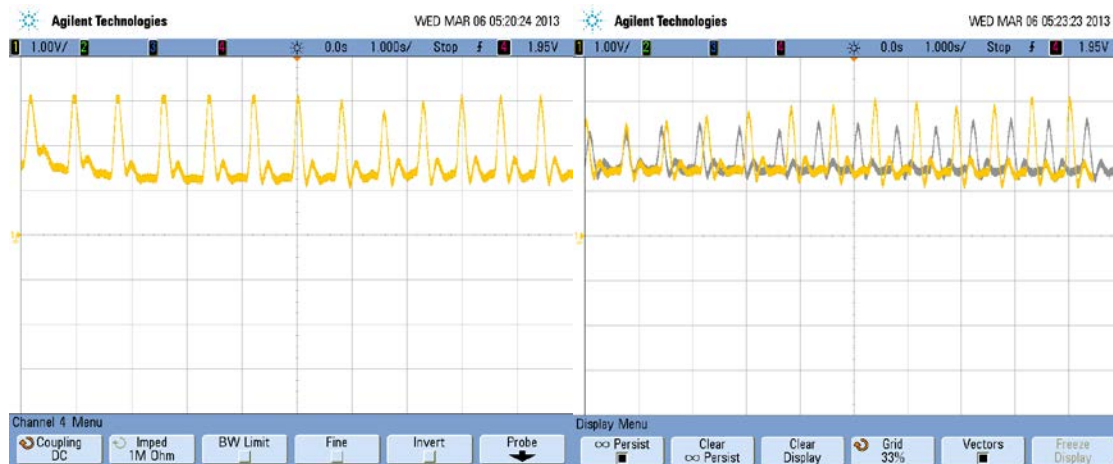


Fig 3.1 Pulse Sensor Amped simulating Heart Rate.

The next two simulations are done under constant ambient light and then in dark room with no ambient light. We can notice that the signal output is constant under both conditions. The heart signal from our pulse sensor is an analog fluctuation in voltage, by measuring the reflected light from the skin. More light and signal goes up, less light the opposite. If the amount of light incident on the sensor remains constant, the signal value will remain close to midpoint range.

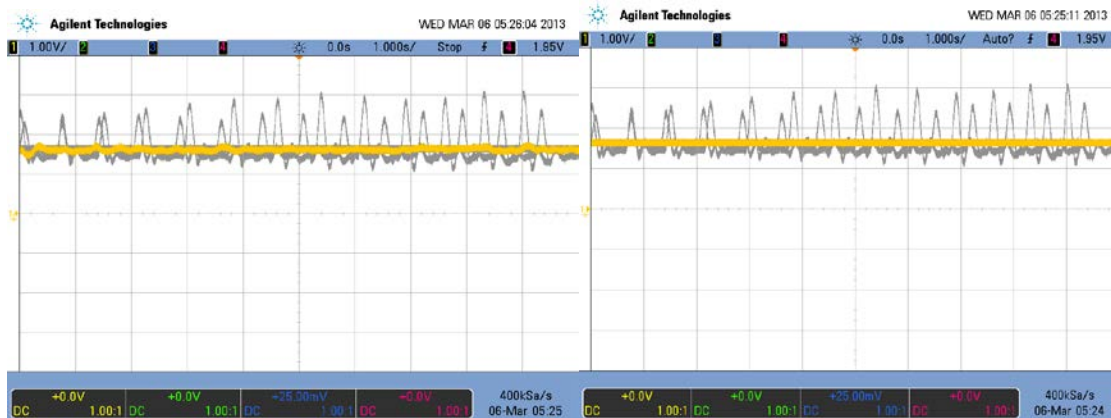


Fig 3.2 Pulse Sensor Amped signal testing at constant Light and No Light.

3.1.2 Skin Temperature Sensor

- **INPUT:** 3V; Clock Input
- **OUTPUT:** Single Pin digital output (SDA)

The MLX90614 is an I²C device, which allows the sensor to output literal information with a 17bit resolution instead of giving a voltage reading. 4.7 K Ω pull up resistors are required to be connected from SDA to power and one from SCL to power. Further verification is done in level 2.

3.1.3 Motion Sensor

We make use of a three axis accelerometer

- **INPUT:** 1.8V to 3.7VDC.
- **OUTPUT:** Accelerometer outputs analog voltage depending on sensed value in each axis.

Simulations:

We power the accelerometer through a 3.3v dc supply and connect the analog outputs of each axis from the sensor to an oscilloscope. In the simulations below:

Yellow=X-axis; Green=Y-axis; Magenta=Z-axis.

Reading the [ADXL335 datasheet](#) we see that on 3.3V power, we should expect an axis to read 1.65V when it has zero acceleration, and the voltage should typically change by 330 mV per G of acceleration.

Therefore to test the sensor accuracy, we allow the sensor to sit stable so that at a time only one axis would point in the direction of acceleration due to gravity and thus we see one signal is roughly $\pm 330\text{mV}$ of the other two axes that are stable at 1.6V(0acceleration)

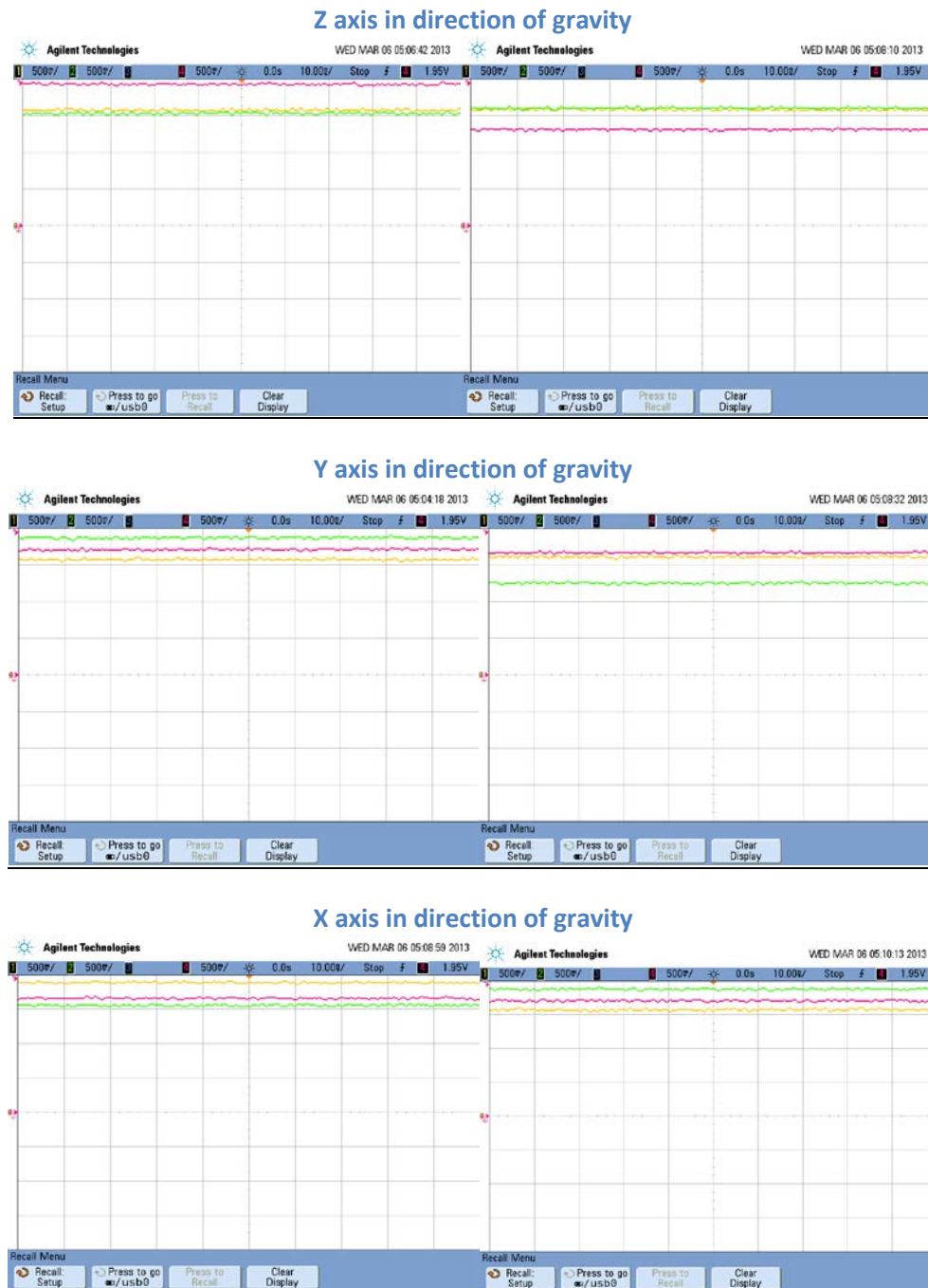


Fig 3.3 Accelerometer Output Checking for all-axis

3.2 Level 2: Individual Component Testing with Arduino

In this stage, importance is given to verification of individual component compatibility with arduino Uno while power supplied from battery.

3.2.1 Pulse Sensor

The calibrated Pulse Sensor Amped sensor came with open source arduino compatible software and a Processing sketch. The code can be found in Appendix A.1. We connect the Pulse Sensor data pin to A0 on Uno board, sensor power pin to 3.3V port on Uno board and Ground pin to GND on board. Note that we are required to provide the Uno Aref pin with supplied power to ensure correct Analog to Digital conversion from Uno board.

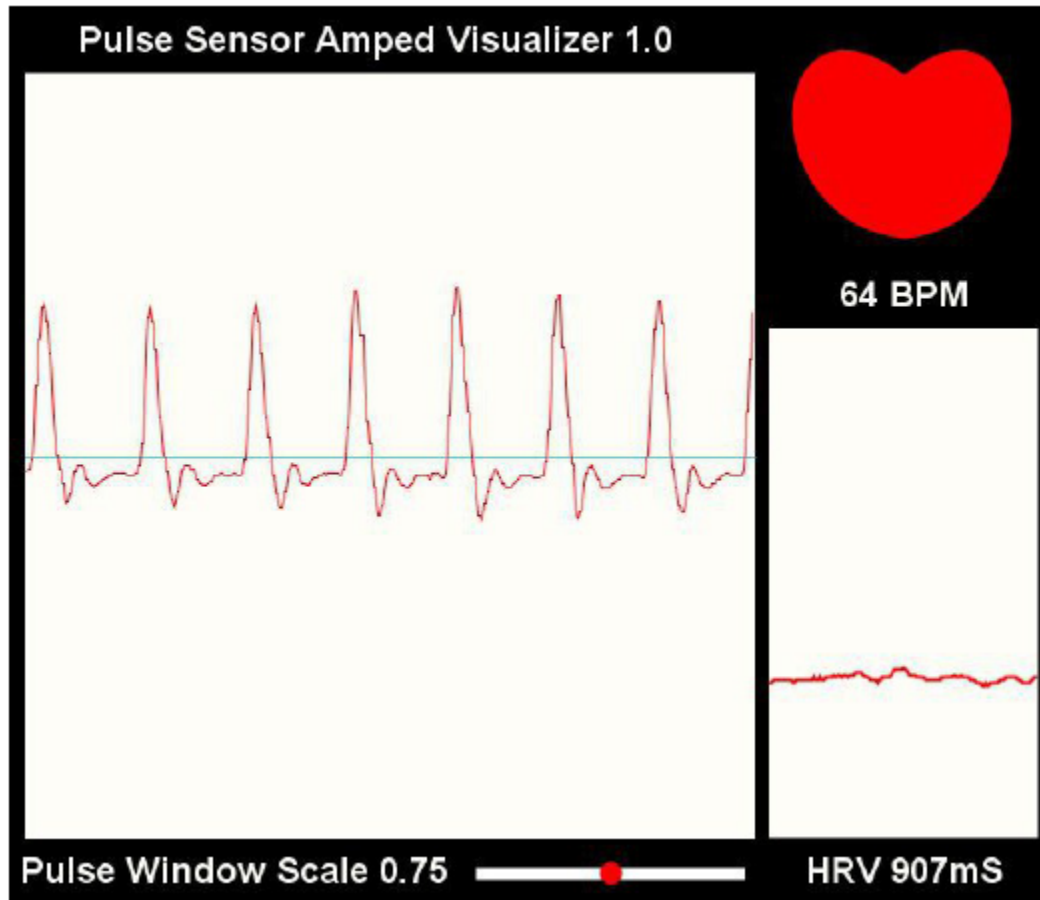


Fig 3.4 Processing Sketch for Pulse Sensor Amped

For final Soldier Status Monitor Device we extract the bpm information from the code and only display the numerical data.

3.2.2 Skin Temperature Sensor

We make use of the open source BILDR sample code for the mlx 90614, found in Appendix A.2

We connect SCL to digital 21, SDA to digital 20, Power to 3.3V and Ground to GND on Arduino Uno board. Below is a snapshot of the arduino output from the Infrared sensor:

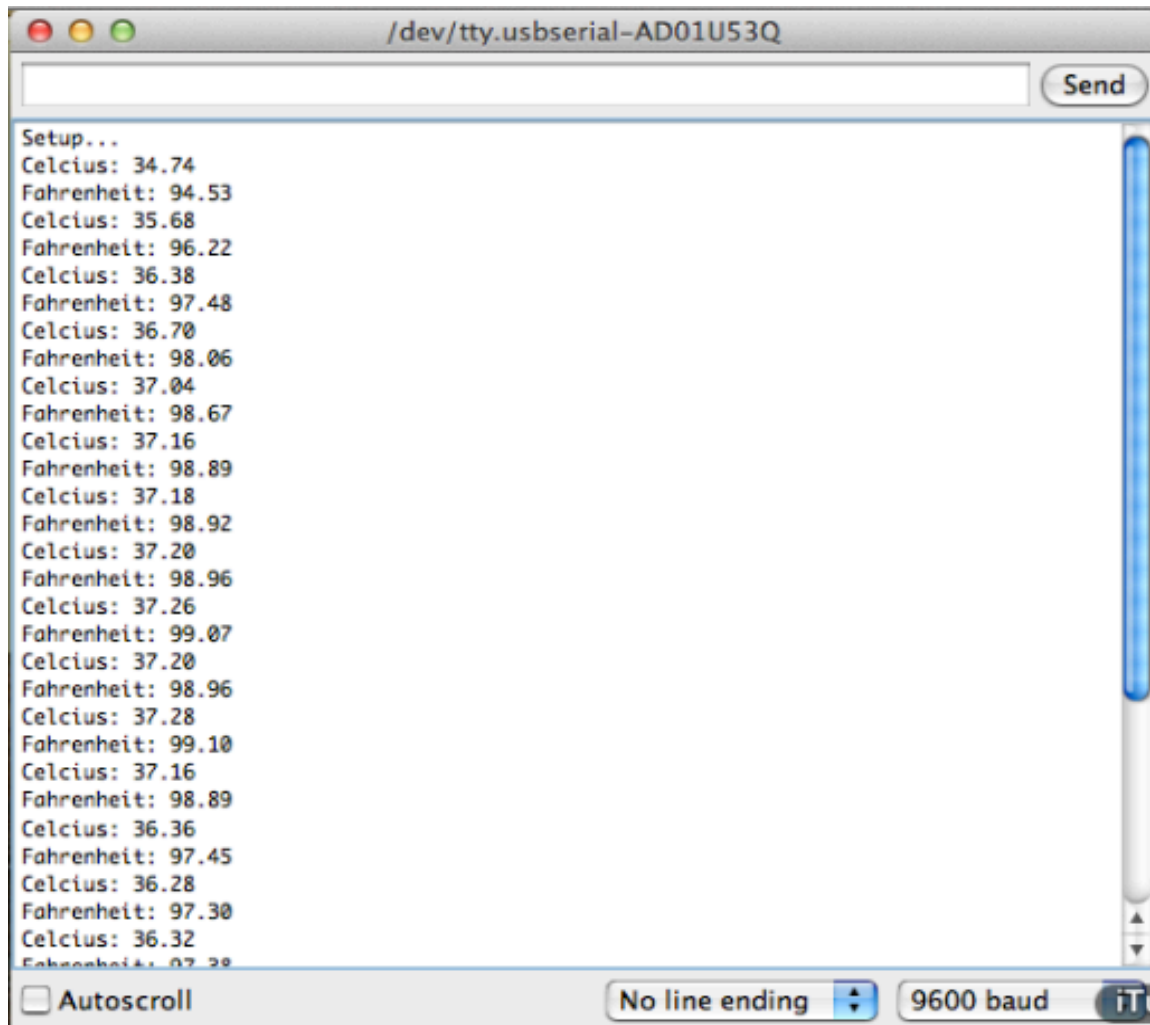


Fig 3.5 MLX90614 literal output of Temperature through Arduino serial port

3.2.3 Motion Sensor

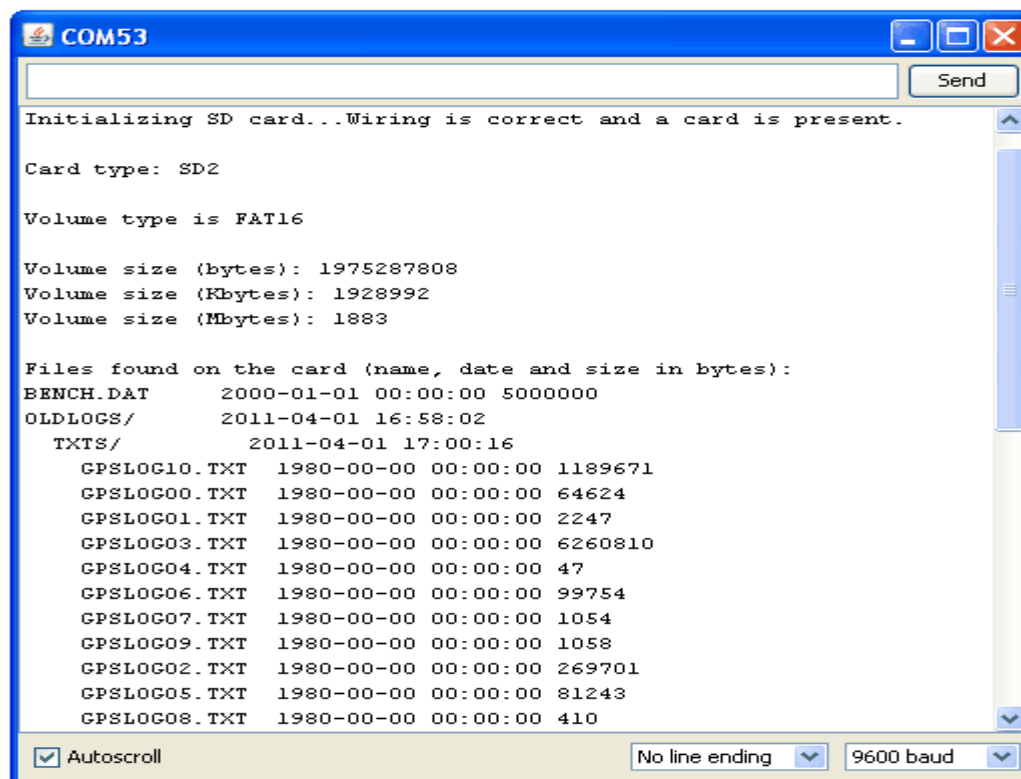
Again, we make use of the BILDR open source code to test our adxl 335. We power the Accelerometer through the uno board at 3.3V, Z axis analog output is connected to A1, Y axis output to A2 and X axis output is connected to A3 on the Uno board. We get output in ADC units that the software code converts to gravitational (g) units. The BILDR code can be found in appendix A.3

```
value of X/Y/Z:
254
256
313
voltage:
1.24
1.25
1.52
acceleration of X/Y/Z:
-0.04 g
0.00 g
1.09 g
```

Fig 3.6 ADXL335 acceleration outputs on each axis in ADC units, voltage and Gunits

3.2.4 SD/MMC Card Holder

SD/MMC card socket requires pull resistors for Clk pin and data pin. Required voltage is 1.8V to 3.6V We make use of the arduino sample codes for SD/MMC card to check successful connection with Arduino Uno board. The code checks if SD/MMC card is present in slot, and if so then it outputs card information card formatting and files present on the card. The code can be found in appendix A.4.



```
COM53
Initializing SD card...Wiring is correct and a card is present.

Card type: SD2

Volume type is FAT16

Volume size (bytes): 1975287808
Volume size (Kbytes): 1928992
Volume size (Mbytes): 1883

Files found on the card (name, date and size in bytes):
BENCH.DAT      2000-01-01 00:00:00 5000000
OLDLOGS/       2011-04-01 16:58:02
  TXTS/        2011-04-01 17:00:16
    GPSLOG10.TXT 1980-00-00 00:00:00 1189671
    GPSLOG00.TXT 1980-00-00 00:00:00 64624
    GPSLOG01.TXT 1980-00-00 00:00:00 2247
    GPSLOG03.TXT 1980-00-00 00:00:00 6260810
    GPSLOG04.TXT 1980-00-00 00:00:00 47
    GPSLOG06.TXT 1980-00-00 00:00:00 99754
    GPSLOG07.TXT 1980-00-00 00:00:00 1054
    GPSLOG09.TXT 1980-00-00 00:00:00 1058
    GPSLOG02.TXT 1980-00-00 00:00:00 269701
    GPSLOG05.TXT 1980-00-00 00:00:00 81243
    GPSLOG08.TXT 1980-00-00 00:00:00 410

Autoscroll No line ending 9600 baud
```

Fig 3.7 SD card read output

3.2.5 Xbee Wireless Transmission Module

XBee-PRO products are engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. IEEE 802.15.4 standard offers the fundamental lower network layers. The modules require minimal power and provide reliable delivery of data between devices. The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other. One Xbee will be attached to the microcontroller packaging the data for transmission to be received at the computer side. The computer Xbee module will be powered by the computer itself using a ftdi to read and power the Xbee module. The microcontroller Xbee power will be supplied by the battery and will be connected to the microcontroller. Xbee is configured through special software called X-CTU. In X-CTU, we set baud rate, PAN ID, and coordinator-end device setting for a proper data transfer.

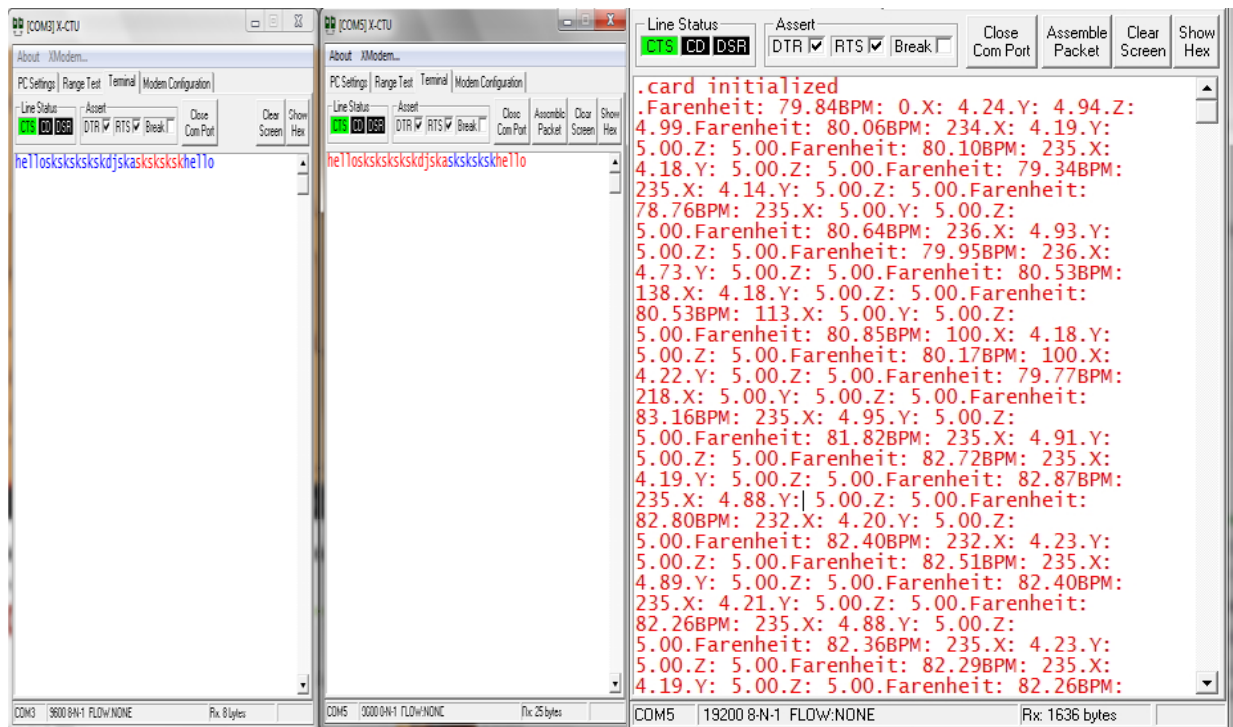


Fig 3.8 Wireless data transfer through xbee module

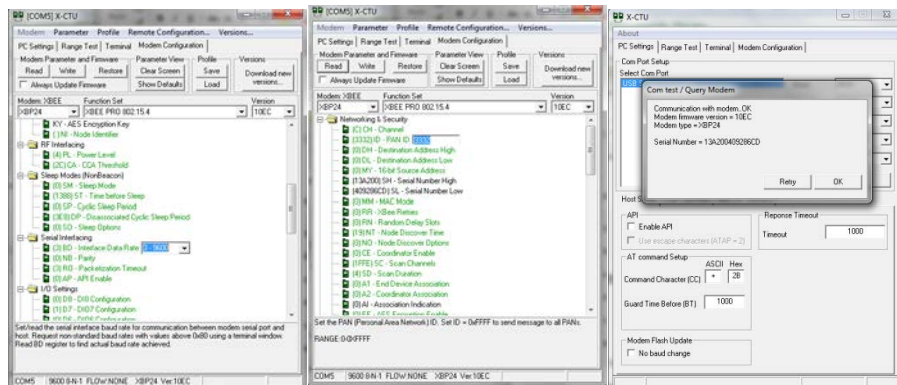


Fig 3.9 Configuration via X-CTU

3.3 Level 3: Final Product Testing

At this stage, we upload the updated final code onto the ATMEGA 328 microcontroller chip and connect all three sensors, SD/MMC slot, and the Xbee transmission module to the microcontroller. The code reads data from the three sensors every 15seconds, package it together, save it on SD card datalog.txt file and then transmits the package to remote location via Xbee module can be found in Appendix B.1.

We verify the final product functioning by the following steps:

- Check transmitted data at remote location with onsite saved data on SD/MMC card
 - This proves correct transmission and saving on SD/MMC card
- Check transmitted pulse sensor data with calibrated Walgreens OxyWatch device
- Check transmitted Temperature data IR Thermometer from Home Depot

4. Costs

4.1 Parts

ii. Parts

Vendor	Vendor Part #	Description	Quantity	Price	Total
Sparkfun	WRL-08742	XBee Pro 60mW Wire Antenna - Series 1 (802.15.4)	2	37.95	75.90
Sparkfun	WRL-11216	XBee Pro 60mW PCB Antenna - Series 1 (802.15.4)	1	37.95	37.95
Sparkfun	SEN-11574	Pulse Sensor Amped	3	24.95	74.85
Sparkfun	SEN-09269	Triple Axis Accelerometer Breakout - ADXL335	3	24.95	74.85
Sparkfun	SEN-11050	Temperature Sensor - Waterproof (DS18B20)	3	9.95	29.85
Sparkfun	BOB-11403	Breakout Board for SD-MMC Cards	2	9.95	19.90
Sparkfun	DEV-10744	Mega Pro 3.3V	2	44.95	89.90
Sparkfun	PRT-08483	Polymer Lithium Ion Battery - 2000mAh	2	16.95	33.90
Sparkfun	PRT-10161	USB LiPoly Charger - Single Cell	2	14.95	29.90
Sparkfun	PRT-09749	JST Right-Angle Connector - Through-Hole 2-Pin	2	0.95	1.90
Sparkfun	DEV-09873	FTDI Basic Breakout - 3.3V	2	14.95	29.90
Sparkfun	CAB-11301	USB Mini-B Cable - 6 Foot	2	3.95	7.90
Sparkfun	SEN-09570	Infrared Thermometer - MLX90614	3	19.95	59.85
Sparkfun	DEV-10524	ATmega328 with Arduino Optiboot (Uno)	2	5.5	11
Sparkfun	COM-00536	Crystal 16MHz	2	0.95	1.9
Sparkfun	COM-09265	Triple Axis Accelerometer - ADXL335	3	9.95	29.85
Sparkfun	PRT-11362	SD/MMC Socket for Secure Digital Disk or Multi Media Cards	3	1.95	5.85
Sparkfun	PRT-08533	USB micro USB SMD Connector	3	1.5	4.5
Sparkfun	PRT-08612	JST Right Angle Connector – White	3	0.95	2.85
Sparkfun	COM-00674	max 1555 LiPoly USB Charger IC	3	1.95	5.85
				total	628.35

Table 4.1: Parts List with Individual Cost

4.2 Labor

Member	\$/hour	Hours/week	Total of hours	Subtotal(\$)
Sanghee Seo	40	20	320	12800
Santhosh Vairavan	40	20	320	12800
Yash Kulkarni	40	20	320	12800
Grand Total (x 2.5)				96000

Table 4.2: Labor Hours and Salary Rate

5. Conclusion

5.1 Accomplishments

Overall, we have achieved all of the goals we have set at the very beginning. We are able to gather heart rate, body temperature, and movement data reliably and send the data wirelessly over 100m for more than 12 hours. First, we created a prototype device using identified individual components connecting to the Arduino Uno board using a breadboard. Furthermore, we have designed our own custom PCB to incorporate microcontroller and sensors and eventually met one of our goals to make the device as portable, compact and light as possible. In addition to the intermediate completion, we have also added LED lightings to display when user vitals reach critical threshold for the sensors and battery making the device even more practical and to use.

5.2 Uncertainties

As we were approaching to the end, we were uncertain if we could finalize our device with PCB. However, instead of waiting for the ordered PCB to be done, we took a chance of making our own PCB and were eventually able to finalize our product as planned. In these kinds of uncertain situation, we learned to take a step forward in such tight time schedule.

5.3 Ethical considerations

Along this project, we served IEEE code ethics to conduct an ethical and professional act as follows:

- 1. to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;*
- 6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;*
- 7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;*

We recognize how important it is to follow the code of ethics.

We, team25, are responsible for making decisions that focus on safety and well-being of the public.

We maintain and broaden our understanding of recent technologies.

We act professional to assist the members, seek for help and accept criticism to improve.

Our device will not harm any users in any ways following the code of ethics.

5.4 Future work

We have achieved all of goals we set at the beginning of the semester. However, there are many potential improvements that could be done for creating a better device.

- (a) Add a gyroscope, magnetometer, and GPS to improve motion detection
- (b) Increase the number of temperature sensors to obtain more accurate skin temperature data
- (c) Add electrode ECG measurement in addition to pulse oximetry
- (d) Upgrade to Atmega2560 as Atmega328 has limited ports with no room for expansion.
- (e) Design a case that is wearable

References

1. IR thermometer schematic : <https://www.sparkfun.com/products/9570>
2. Pulse Sensor Amped schematic : <https://www.sparkfun.com/products/11574>
3. Triple Axis Accelerometer Breakout schematic: <https://www.sparkfun.com/products/9269>
4. USB poly charger schematic: <https://www.sparkfun.com/products/10161>
5. Power for Mega Pro 3.3v Schematic: <https://www.sparkfun.com/products/10744>
6. Mega Pro 3.3v pin layout : <https://www.sparkfun.com/products/10744>
7. Breakout Board for SD-MMC Cards Schematic : <https://www.sparkfun.com/products/11403>
8. Xbee to Microcontroller Schematic : <https://www.sparkfun.com/products/8742>
9. Xbee flow from Microcontroller to Computer : <https://www.sparkfun.com/products/8742>

Appendix A Open Source Individual Sensor Codes with Arduino

In this section of the appendix we list the codes we used for level 2 design verification. Here, we tested each sensor's compatibility with the arduino uno board.

Pulse Sensor Amped code

```
/*
>> Pulse Sensor Amped 1.1 <<
This code is for Pulse Sensor Amped by Joel Murphy and Yury Gitman
www.pulsesensor.com
>>> Pulse Sensor purple wire goes to Analog Pin 0 <<<
Pulse Sensor sample acquisition and processing happens in the background via Timer 2 interrupt. 2mS sample
rate.
PWM on pins 3 and 11 will not work when using this code, because we are using Timer 2!
The following variables are automatically updated:
Signal : int that holds the analog signal data straight from the sensor. updated every 2mS.
IBI : int that holds the time interval between beats. 2mS resolution.
BPM : int that holds the heart rate value, derived every beat, from averaging previous 10 IBI values.
QS : boolean that is made true whenever Pulse is found and BPM is updated. User must reset.
Pulse : boolean that is true when a heartbeat is sensed then false in time with pin13 LED going out.
```

This code is designed with output serial data to Processing sketch "PulseSensorAmped_Processing-xx"

The Processing sketch is a simple data visualizer.

All the work to find the heartbeat and determine the heartrate happens in the code below.

Pin 13 LED will blink with heartbeat.

If you want to use pin 13 for something else, adjust the interrupt handler

It will also fade an LED on pin fadePin with every beat. Put an LED and series resistor from fadePin to GND.

Check here for detailed code walkthrough:

<http://pulsesensor.myshopify.com/pages/pulse-sensor-amped-arduino-v1dot1>

Code Version 02 by Joel Murphy & Yury Gitman Fall 2012

This update changes the HRV variable name to IBI, which stands for Inter-Beat Interval, for clarity.

Switched the interrupt to Timer2. 500Hz sample rate, 2mS resolution IBI value.

Fade LED pin moved to pin 5 (use of Timer2 disables PWM on pins 3 & 11).

Tidied up inefficiencies since the last version.

```
*/
```

```
// VARIABLES
```

```
int pulsePin = 0;           // Pulse Sensor purple wire connected to analog pin 0
int blinkPin = 13;          // pin to blink led at each beat
int fadePin = 5;             // pin to do fancy classy fading blink at each beat
int fadeRate = 0;           // used to fade LED on with PWM on fadePin
```

```
// these variables are volatile because they are used during the interrupt service routine!
```

```
volatile int BPM;           // used to hold the pulse rate
volatile int Signal;        // holds the incoming raw data
volatile int IBI = 600;     // holds the time between beats, the Inter-Beat Interval
volatile boolean Pulse = false; // true when pulse wave is high, false when it's low
volatile boolean QS = false; // becomes true when Arduino finds a beat.
```

```
void setup(){
  pinMode(blinkPin,OUTPUT); // pin that will blink to your heartbeat!
```

```

pinMode(fadePin,OUTPUT);    // pin that will fade to your heartbeat!
Serial.begin(115200);       // we agree to talk fast!
interruptSetup();           // sets up to read Pulse Sensor signal every 2mS
// UN-COMMENT THE NEXT LINE IF YOU ARE POWERING The Pulse Sensor AT LOW VOLTAGE,
// AND APPLY THAT VOLTAGE TO THE A-REF PIN
//analogReference(EXTERNAL);
}

void loop(){
  sendDataToProcessing('S', Signal); // send Processing the raw Pulse Sensor data
  if (QS == true){                  // Quantified Self flag is true when arduino finds a heartbeat
    fadeRate = 255;                 // Set 'fadeRate' Variable to 255 to fade LED with pulse
    sendDataToProcessing('B',BPM); // send heart rate with a 'B' prefix
    sendDataToProcessing('Q',IBI); // send time between beats with a 'Q' prefix
    QS = false;                     // reset the Quantified Self flag for next time
  }

  ledFadeToBeat();

  delay(20);                        // take a break
}

void ledFadeToBeat(){
  fadeRate -= 15;                   // set LED fade value
  fadeRate = constrain(fadeRate,0,255); // keep LED fade value from going into negative numbers!
  analogWrite(fadePin,fadeRate);    // fade LED
}

void sendDataToProcessing(char symbol, int data ){
  Serial.print(symbol);             // symbol prefix tells Processing what type of data is coming
  Serial.println(data);             // the data to send culminating in a carriage return
}

```

Accelerometer (adxl335) code

```

// these constants describe the pins. They won't change:
const int xpin = A1;           // x-axis of the accelerometer
const int ypin = A2;           // y-axis
const int zpin = A3;           // z-axis (only on 3-axis models)

int sampleDelay = 500; //number of milliseconds between readings

void setup()
{
  // initialize the serial communications:
  Serial.begin(9600);

  //Make sure the analog-to-digital converter takes its reference voltage from
  // the AREF pin
  analogReference(EXTERNAL);
}

```

```

pinMode(xpin, INPUT);
pinMode(ypin, INPUT);
pinMode(zpin, INPUT);
}

void loop()
{
  int x = analogRead(xpin);

  //add a small delay between pin readings. I read that you should
  //do this but haven't tested the importance
  // delay(1);

  int y = analogRead(ypin);

  //add a small delay between pin readings. I read that you should
  //do this but haven't tested the importance
  // delay(1);

  int z = analogRead(zpin);

  //zero_G is the reading we expect from the sensor when it detects
  //no acceleration. Subtract this value from the sensor reading to
  //get a shifted sensor reading.
  float zero_G = 512.0;

  //scale is the number of units we expect the sensor reading to
  //change when the acceleration along an axis changes by 1G.
  //Divide the shifted sensor reading by scale to get acceleration in Gs.
  float scale = 102.3;

  Serial.print(((float)x - zero_G)/scale);
  Serial.print("\t");

  Serial.print(((float)y - zero_G)/scale);
  Serial.print("\t");

  Serial.print(((float)z - zero_G)/scale);
  Serial.print("\n");

  // delay before next reading:
  //delay(sampleDelay);
}

```

Infrared Sensor (mlx90614)

```

#include <i2cmaster.h>

void setup(){
  Serial.begin(9600);
  Serial.println("Setup...");

  i2c_init(); //Initialise the i2c bus
  PORTC = (1 << PORTC4) | (1 << PORTC5); //enable pullups
}

void loop(){
  int dev = 0x5A<<1;
  int data_low = 0;
  int data_high = 0;
  int pec = 0;

  i2c_start_wait(dev+I2C_WRITE);
  i2c_write(0x07);

  // read
  i2c_rep_start(dev+I2C_READ);
  data_low = i2c_readAck(); //Read 1 byte and then send ack
  data_high = i2c_readAck(); //Read 1 byte and then send ack
  pec = i2c_readNak();
  i2c_stop();

  //This converts high and low bytes together and processes temperature, MSB is
  a error bit and is ignored for temps
  double tempFactor = 0.02; // 0.02 degrees per LSB (measurement resolution of
  the MLX90614)
  double tempData = 0x0000; // zero out the data
  int frac; // data past the decimal point

  // This masks off the error bit of the high byte, then moves it left 8 bits
  and adds the low byte.
  tempData = (double)((((data_high & 0x007F) << 8) + data_low));
  tempData = (tempData * tempFactor)-0.01;

  float celcius = tempData - 273.15;
  float fahrenheit = (celcius*1.8) + 32;

  Serial.print("Celcius: ");
  Serial.println(celcius);

  Serial.print("Fahrenheit: ");
  Serial.println(fahrenheit);

  delay(1000); // wait a second before printing again
}

```

Appendix B Arduino Final Code

```
#include <i2cmaster.h>
#include <SoftwareSerial.h>
#include <SD.h>

const int xpin = A1;          // x-axis of the accelerometer

const int ypin = A2;          // y-axis

const int zpin = A3;          // z-axis (only on 3-axis models)

int sampleDelay = 500; //number of milliseconds between readings

// VARIABLES
int pulsePin = 0;             // Pulse Sensor purple wire connected to analog pin 0
int blinkPin = 13;            // pin to blink led at each beat
int fadePin = 5;              // pin to do fancy classy fading blink at each beat
int fadeRate = 0;             // used to fade LED on with PWM on fadePin

// these variables are volatile because they are used during the interrupt service routine!
volatile int BPM;             // used to hold the pulse rate
volatile int Signal;          // holds the incoming raw data
volatile int IBI = 600;       // holds the time between beats, the Inter-Beat Interval
volatile boolean Pulse = false; // true when pulse wave is high, false when it's low
volatile boolean QS = false;  // becomes true when Arduino finds a beat.

const int chipSelect = 10;

SoftwareSerial xbee(3,2);
void setup(){
  // initialize the serial communications:

  Serial.begin(115200);
  xbee.begin(19200);
  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin is set to
  // output, even if you don't use it:
  pinMode(10, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
```

```

    return;
}
Serial.println("card initialized.");

//Make sure the analog-to-digital converter takes its reference voltage from

// the AREF pin

analogReference(EXTERNAL);
i2c_init(); //Initialise the i2c bus
PORTC = (1 << PORTC4) | (1 << PORTC5); //enable pullups

pinMode(xpin, INPUT);

pinMode(ypin, INPUT);

pinMode(zpin, INPUT);
pinMode(blinkPin, OUTPUT);    // pin that will blink to your heartbeat!
pinMode(fadePin, OUTPUT);    // pin that will fade to your heartbeat!
// Serial.begin(115200);      // we agree to talk fast!
interruptSetup();            // sets up to read Pulse Sensor signal every 2mS
// UN-COMMENT THE NEXT LINE IF YOU ARE POWERING The Pulse Sensor AT LOW VOLTAGE,
// AND APPLY THAT VOLTAGE TO THE A-REF PIN
analogReference(EXTERNAL);
}

void loop(){
  File dataFile = SD.open("datalog.txt", FILE_WRITE);
  int x = analogRead(xpin);

  //add a small delay between pin readings. I read that you should

  //do this but haven't tested the importance

  // delay(1);

  int y = analogRead(ypin);

  //add a small delay between pin readings. I read that you should

  //do this but haven't tested the importance

  // delay(1);

```

```

int z = analogRead(zpin);

//zero_G is the reading we expect from the sensor when it detects
//no acceleration. Subtract this value from the sensor reading to
//get a shifted sensor reading.
float zero_G = 512.0;

//scale is the number of units we expect the sensor reading to
//change when the acceleration along an axis changes by 1G.
//Divide the shifted sensor reading by scale to get acceleration in Gs.
float scale = 102.3;

int dev = 0x5A<<1;
int data_low = 0;
int data_high = 0;
int pec = 0;

i2c_start_wait(dev+I2C_WRITE);
i2c_write(0x07);

// read
i2c_rep_start(dev+I2C_READ);
data_low = i2c_readAck(); //Read 1 byte and then send ack
data_high = i2c_readAck(); //Read 1 byte and then send ack
pec = i2c_readNak();
i2c_stop();

//This converts high and low bytes together and processes temperature, MSB is a error bit and is
ignored for temps
double tempFactor = 0.02; // 0.02 degrees per LSB (measurement resolution of the MLX90614)
double tempData = 0x0000; // zero out the data
int frac; // data past the decimal point

// This masks off the error bit of the high byte, then moves it left 8 bits and adds the low byte.
tempData = (double)((((data_high & 0x007F) << 8) + data_low));
tempData = (tempData * tempFactor)-0.01;

float celcius = tempData - 273.15;
float fahrenheit = (celcius*1.8) + 32;
Serial.print("Fahrenheit: ");
Serial.println(fahrenheit);
xbee.print("Fahrenheit: ");

```

```

xbee.print(fahrenheit);
if (dataFile)
{
  dataFile.print("Fahrenheit: ");
  dataFile.print(fahrenheit);
  dataFile.print("\n");
}
// if the file isn't open, pop up an error:
else
{
  Serial.println("error opening datalog.txt");
}
Serial.print(((float)x - zero_G)/scale);
xbee.print("X: ");
xbee.print(((float)x - zero_G)/scale);
xbee.print("\t");
Serial.print("\t");
if (dataFile)
{
  dataFile.print("X: ");
  dataFile.print(((float)x - zero_G)/scale);
  dataFile.print("\t");
}
// if the file isn't open, pop up an error:
else
{
  Serial.println("error opening datalog.txt");
}

```

```

Serial.print(((float)y - zero_G)/scale);
Serial.print("\t");
xbee.print("Y: ");
xbee.print(((float)y - zero_G)/scale);
xbee.print("\t");
if (dataFile)
{
  dataFile.print("Y: ");
  dataFile.print(((float)y - zero_G)/scale);
  dataFile.print("\t");
}
// if the file isn't open, pop up an error:
else
{
  Serial.println("error opening datalog.txt");
}

```

```

Serial.print(((float)z - zero_G)/scale);
Serial.print("\n");
xbee.print("Z: ");

```



```

xbee.print(((float)z - zero_G)/scale);
xbee.print("\n");
if (dataFile)
{
  dataFile.print("Z: ");
  dataFile.print(((float)z - zero_G)/scale);
  dataFile.print("\n");
}
// if the file isn't open, pop up an error:
else
{
  Serial.println("error opening datalog.txt");
}

Serial.print("BPM: ");
xbee.print("BPM: ");
Serial.print(BPM);
xbee.print(BPM);
Serial.print("\n");
xbee.print("\n");

if (dataFile)
{
  dataFile.print("BPM: ");
  dataFile.print(BPM);
  dataFile.print("\n");
  dataFile.close();
}
// if the file isn't open, pop up an error:
else
{
  Serial.println("error opening datalog.txt");
}

// delay before next reading:

//delay(sampleDelay);
//sendDataToProcessing('S', Signal); // send Processing the raw Pulse Sensor data
//if (QS == true){ // Quantified Self flag is true when arduino finds a heartbeat
//  fadeRate = 255; // Set 'fadeRate' Variable to 255 to fade LED with pulse
//  sendDataToProcessing('B',BPM); // send heart rate with a 'B' prefix
//  sendDataToProcessing('Q',IBI); // send time between beats with a 'Q' prefix
//  QS = false; // reset the Quantified Self flag for next time
//}

ledFadeToBeat();

delay(5000); // take a break
}

```

```

void ledFadeToBeat(){
  fadeRate -= 15;           // set LED fade value
  fadeRate = constrain(fadeRate,0,255); // keep LED fade value from going into negative
  numbers!
  analogWrite(fadePin,fadeRate);    // fade LED
}

void sendDataToProcessing(char symbol, int data ){
  if(symbol=='B')
  {
    Serial.print(": ");      // symbol prefix tells Processing what type of data is coming
    Serial.println(data);    // the data to send culminating in a carriage return
  }
}

```

Appendix C Requirement and Verification Table

Below is a list of Requirements and Verifications that were prepared by my team for the purpose of testing final product. The table helped reach the goals that were set in the start. All of our requirements were successfully verified during the team product demo.

Table C.1 Requirements and Verification Table

Block Title	Requirements	Verification
Heart Rate Sensor	1. “beats per minute” (bpm) to be $(\pm)2\%$ of actual heart bpm a. Displays peaks with constant time interval b. Maximum voltage should be 3.3V	1. Use calibrated off-the-shelf heart rate monitor to compare bpm a. Use oscilloscope to verify constant time interval between peaks b. Use oscilloscope to check voltage
Motion Sensor	1. It needs to generate analog output in x,y and z axis a. When stationary, x and y axis needs to output 1.6v and z axis 1.93v b. When moving, every 1g of acceleration corresponds to an additional 0.33v	1. Use oscilloscope to verify the outputs of x, y and z axis a. Use oscilloscope to check voltage at stationary b. Use oscilloscope to check voltage when moving by at least 1g
IR sensor	1. It needs to give out 2% of actual body temperature a. Data out from IR sensor gives us 17bits bit depth resolution	1. Use off-the-shelf thermometer to verify a. Use oscilloscope to check 17-bit resolution
Solder Vitals Supply	1. Battery should power microcontroller, sensors, and transmitter for 12 hours a. Battery maintains 3.0v to 3.3v for 12hours b. Battery should provide 2.15mA	1. Verify by powering microcontroller, sensors and transmitter for 12 hours a. Use multimeter to check voltage of battery b. Use multimeter to check current from charger circuit to the battery
Storage Module (SD card)	1. Stores packaged data from the sensors a. Stored data should be in hex and written to a text file	1. Write known data in text file to SD card a. Connect to a computer and check
Threshold Display Module	1. if (data received to microcontroller) is bigger (preset threshold value), LED turns on a. LED(red, pin13) blinks (Pulse) b. LED(green, pin12) blinks (Temperature) c. LED(blue, pin11) blinks (Battery)	1. Send data that would cause overflow a. Set pulse rate $> 220\text{bpm}$ and <20 b. Body temperature $> 100^{\circ}\text{F}$ and <93 c. Battery voltage $< 3\text{V}$

Microcontroller	<ol style="list-style-type: none"> 1. Powered from the battery <ol style="list-style-type: none"> a. Microcontroller should be powered at 3.3v 2. Microcontroller takes digital input from sensors and converts it to digital input <ol style="list-style-type: none"> a. The packed digital data will be stored in SD card 	<ol style="list-style-type: none"> 1. Power LED lights up <ol style="list-style-type: none"> a. Use oscilloscope to check Vcc is 3.3v 2. Use oscilloscope to display analog signals and manually calculate <ol style="list-style-type: none"> a. Check the hexadecimal digital data in SD card
Transmitter	<ol style="list-style-type: none"> 1. Transmitter should be powered by the battery <ol style="list-style-type: none"> a. Transmitter requires 3.3v 2. Accurate data transmission over 100m <ol style="list-style-type: none"> a. When transmitting, a sinusoidal wave is outputted b. Transmitter requires 63mW (18dBm) 	<ol style="list-style-type: none"> 1. Use multimeter to check Vcc has a positive voltage <ol style="list-style-type: none"> a. Use multimeter to check Vcc pin is at 3.3v 2. Check receiving Xbee module if receiving data is same as transmitting data <ol style="list-style-type: none"> a. Use oscilloscope to check outputted wave is sinusoidal b. Use oscilloscope to check power required is 63mV
Receiver	<ol style="list-style-type: none"> 1. Receiver should be powered by the computer <ol style="list-style-type: none"> a. Receiver requires 3.3v 2. Accurate data transmission over 100m <ol style="list-style-type: none"> a. When receiving, a sinusoidal wave is shown 	<ol style="list-style-type: none"> 1. Use multimeter to check Vcc has a positive voltage <ol style="list-style-type: none"> a. Use multimeter to check Vcc pin is at 3.3v 2. Check receiving Xbee module if receiving data is same as transmitting data <ol style="list-style-type: none"> a. Use oscilloscope to check outputted wave is sinusoidal
Computer Display	<ol style="list-style-type: none"> 1. Displays received data in a graphical form 	<ol style="list-style-type: none"> 1. Use calibrated off-the-shelf monitor to show correct measurement