

**A Low Power Health Monitor solution**

# Abstract

*Improved bio-marker and sensor information has been a topic of interest to enable accessible patient data via a continual monitoring system. In particular, medical personnel, such as physicians and nurses, who need rapid science data products, such as temperature and pulse rate created from remote sensing, are the key target user group for this technology. Our design solution is a low-power bio-metric area network capable of transmitting patient data via wireless telemetry supported by a battery platform for continuous operation. This portable design can reduce cost to patients and physicians, provide near-real time accessible patient data, and improve patient diagnosis and monitoring.*

# Team Members

Team 1:

Matt Kochevar

Jason Schiedermayer

Jay Wilson

Jesus Guerrero

James Northup

Joseph Mouwad

Team 2:

Jack Powell

Patrick Belon

Henok Tadesse

Yang Jiao

Michael Scherer

Thien Nguyen

Contents

[Abstract 2](#_Toc387926454)

[Team Members 2](#_Toc387926455)

[Introduction 4](#_Toc387926456)

[Aggregator-Node Communication Protocol 4](#_Toc387926457)

[Initial Design 5](#_Toc387926458)

[Bluetooth Low Energy Protocol 7](#_Toc387926459)

[Phone Applications 8](#_Toc387926460)

[Android 8](#_Toc387926461)

[iOS 10](#_Toc387926462)

[SQL Server/PHP Service 12](#_Toc387926463)

[Sensor Node Circuitry 13](#_Toc387926464)

[Temperature Sensor 15](#_Toc387926465)

[Pulse Sensor 17](#_Toc387926466)

[Energy Harvesting/Power Supply 18](#_Toc387926467)

[Aggregator 20](#_Toc387926468)

[Sensor Bands 22](#_Toc387926469)

[Aggregator Case 22](#_Toc387926470)

[Budget 23](#_Toc387926471)

[Milestones 25](#_Toc387926472)

[Conclusion 26](#_Toc387926473)

[Appendix A – Communication Protocol 27](#_Toc387926474)

[Appendix B - BLE 28](#_Toc387926475)

[Appendix C – iOS App 30](#_Toc387926476)

[Appendix D – Database 33](#_Toc387926477)

[Appendix E – Temperature Sensor 35](#_Toc387926478)

[Appendix F – Node PCBs 37](#_Toc387926479)

[Appendix G – Power Supply Data 38](#_Toc387926480)

[Appendix H – Aggregator PCBs 40](#_Toc387926481)

# Introduction

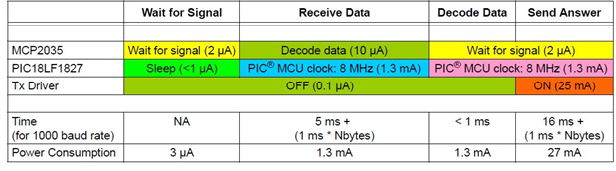
In the past century a rise in population can be seen partially due to advancement in Medical Technology. Deeper understanding of the human body, microorganisms and diseases has made the invention of revolutionary products, such the antibiotics, possible. This outstanding work of scientists and engineers has contributed to planet earth’s ability to accommodate an increased population. However, it is also the responsibly of scientists and engineers to continue innovation and make possible for our society to live a healthy life. Recently there has been another revolution in electronic technology that would allow for this direction. This technological revolution would help support many patients inexpensively, conveniently and timely.

Here at BioComm, it is our goal to combine the advancement in wireless technology with advanced medical technology, and provide convenience for patients to monitor their biometric information without creating disturbance of lifestyle. In addition, we enable doctors to remotely monitor their patient’s health. This provides continual monitoring of patients, such as with chronic conditions while minimizing time spent in the hospital. Due to an increase in the number of sickness, it is becoming more difficult and expensive to keep sick patients in the hospital. Not only is it inconvenient, but also unnecessary to keep patients experiencing mild symptoms in the hospital to be able to monitor them. BioComm solves this problem by using cutting edge technology that is available today. We provide a way to accurately measure different vitals of the patient, providing near-real time medical data to the doctor and to the patient wirelessly.

Our system involves small sensors placed on the patient’s body to measure specific vitals that the physician wants to monitor. These sensors transmit biometric information through the skin to an aggregator that receives and partially processes the data. When the patient’s smartphone is within a detectable range, the data is transmitted to the phone through Bluetooth Low Energy technology. Once the data is on the phone, processing is done onboard and the information is made available to the patient through a good looking User Interface (UI). This information is also pushed to the cloud environment immediately whenever the phone has connection to the internet. Once the information is on the cloud it is immediately made available to the doctor and can be accessed through a computer or any mobile device.

# Aggregator-Node Communication Protocol

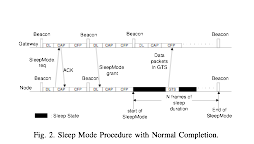
When deciding on the sensor network communication system several different technologies like Bluetooth, and Zigbee were considered, but it was decided to use Microchip's Bodycom technology because of how it was a low power system that didn't have interference with normal ISM band systems. Originally we wanted to implement an extremely low power protocol that involved the nodes deciding when they wanted to send data, but due to large amount of packet loss, and other reasons that will be expanded upon later, a protocol that involves the aggregator unit sending a ping to a sensor node and then having the sensor node respond with the data and repeating this for each sensor node was implemented instead. The reason for the Node deciding when they want to send is that is take up about 10% of power to be in sleep mode as it does for it to be in listening mode and it eliminates the issue with node waking up for short amounts of time when they receive a packet that isn't meant for them.

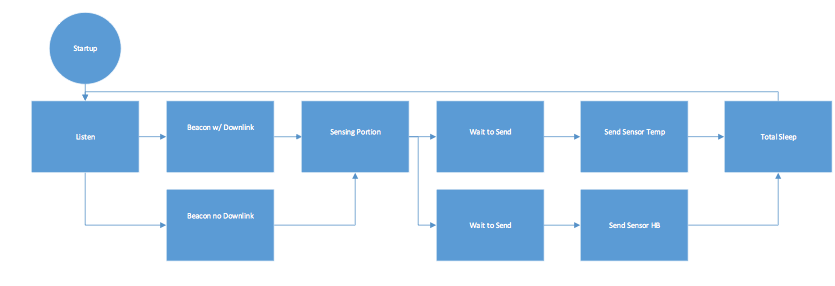
The table above shows the power requirements of the nodes where power is more of an issue due to their smaller form factor and need for less power requirements. With this system the node is in sleep the majority of the time waiting for a signal and uses about 3uA of power, but when it receives a signal is wakes up the microcontroller for 11 ms with our packet structure and then will decode the packet. During this time the microcontroller being on increases the power consumption to 1.3mA, and for this 11ms and then when the response is sent it runs at ~27mA for 26ms with our packet structure and will go back to sleep after it is done sending.

This protocol was built off of the template code that is available with the BodyComm development kit, but had to be expanded upon to be used in the sensor network. Protocol Appendix A1 shows one of the first thing that was implemented which was the packet structure. The BC\_SendData() command allowed the system to send a command, and data to a specific sensor node determined by the address field. The ECHO\_REQUEST command was used in order to request data from a specific node, and is used by the aggregator to request data from different nodes. The aggregator uses 1 byte of the data field in order to specify a packet Id. This is used for debugging purposes to determine if the node has missed a packet from the aggregator. When the node receives the packet from the aggregator it first determines if the address of the packet matches its own address if it doesn't then it disregards the packet and goes immediately back to sleep. If the packet is meant for that specific node it will then decode the packet and see what the command field of the packet is and if it is ECHO\_REQUEST will start creating a packet to send back to the aggregator as seen in protocol Appendix A2. This packet is structured so that the first byte of the data packet is determined by what kind of data it is, the next 2 bytes are for storing the 10 bits of the sensor data that we have retrieved previously and the last byte is used as a packet Id for determining how many packets have been missed from that node by the aggregator. After it creates this packet it will then send it back to the aggregator with the command ECHO\_RESPONCE, the system will go into sleep with the analog front end put into listening mode. The aggregator will then take this packet, decode it, and then take DataBuffer[0] through DataBuffer[2] and append a 0x0A to the front and a 0x0D to the end of the data and send it through UART to the BLE. The 0x0A and 0x0D are used as a start and stop byte so the BLE chip know what data it should send off to the phone and what data is noise or junk. After sending the data to the BLE chip it will then determine what sensor node it needs to collect data from next and repeat the same process as above.

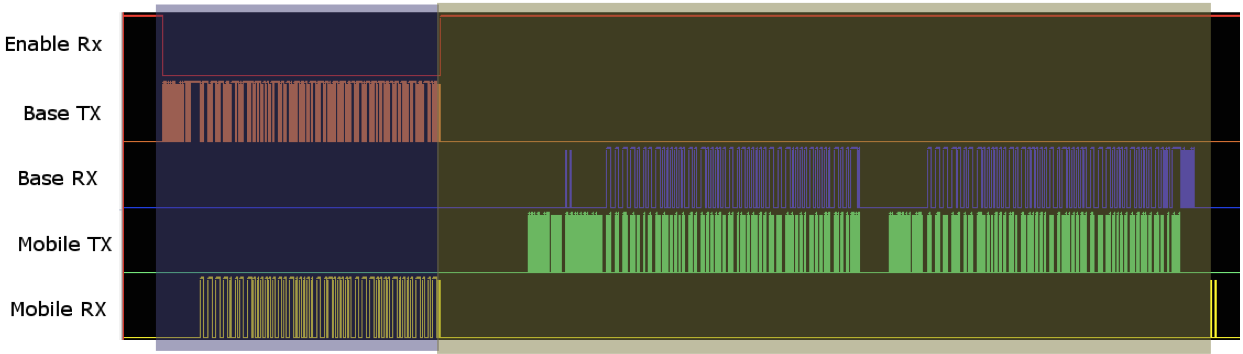
## Initial Design

The initial MAC layer protocol that was going to be implemented was inspired by BodyMac. BodyMac protocol is a Mac protocol intended to be used for ultra-low power, personal area networks, specifically those gathering biometric data. A simplified version of the protocol is shown below. This is figure 2 taken from the BodyMac paper on IEEE. The protocol is based

heavily off of a time slotted portion where nodes communicate to a master unit during specific time intervals. For the protocol for BioComm, it was decided to use a protocol similar to this. While the BodyComm system comes shipped with a MAC protocol in which nodes only respond to requests issued by the aggregator with their address on it, this has power consumption penalties that the team wanted to avoid. Specifically, whenever a node receives a packet from the aggregator unit, it must wake up from its low power sleep state to check the address given to it by the unit. If the address is its own, it complies with the command sent by the aggregator, otherwise it ignores the command and continues to sleep. The team wanted to avoid spurious wake ups, thus we thought a protocol in which nodes were only synchronized periodically by a beacon but sent information in response based off a time slot would save a lot of power. This would mean that the nodes would only be awake when they needed to be.

 Below is our teams’ implementation of the protocol. The nodes various states are enumerated below. The node has a listening state in which it has its external radio powered but in which the node is still asleep. Then after a beacon is received, either this beacon contains new timing information, in the event of a startup or new node added, or it does not contain timing information; this differentiates between beacon with downlink and beacon without downlink. The sensing portion is the time allotted for the node to complete its measurement and the wait to send portion is again when the node is asleep. After the node sends its data in its appropriate time slot, it enters a total sleep state in which its radio is turned off and the node goes to sleep only to turn its radio back on and enter the listening state moments before it expects the next beacon. Since vitals data only needs to be measured about every 2 hours, this total sleep state is when the node enters its maximum power savings.

Problems Encountered

 The above protocol implementation was attempted but did not succeed due to limitations in Microchip BodyComm technology that were discovered during the implementation stages of this project. Below is a figure detailing the problems experienced by the team. Here is an exchange of the aggregator unit communicating with the node. In Microchip’s documentation they are referred to as Base Unit and Mobile unit respectively. Note that in this figure the aggregator initiates the communication. Moments after the request is sent, there is a spike on the Base Rx line. First off, this line represents modulated data being received by the aggregator in response to the node. The spike which occurs in the preamble of the packet, is actually a command being sent from the PIC micro-controller to its Analog Front End. This command comes as part of a library implementation of the BodyComm’s datalink layer. What this library is performing is a programming of the analog front end and issuing a lock command to its Automatic Gain Control Circuit. Not coincidentally, there are two pre-ambles in this packet of data sent from the node. The first preamble coincides right during an Analog Front End lock command.

The problem that was encountered was that in Microchips default system, the node responds immediately after a request from the aggregator unit. However, in the BodyMac implementation the node responds in a time-slot interval after receiving a beacon from the aggregator. Due to the default implementation of BodyComm’s protocol, a delay in receiving a response from the node would mean that the Automatic Gain Control lock command would appear when there was nothing on the receive path.

Attempted Fixes

To address this issue an attempted fix was to remove the AGC locking command. The thought process was that the AGC could track the gain sufficiently during this time period. However, this did not resolve the case. At best 75 percent of packets were continuously dropped. The packets would appear at the receiver end of the aggregator but would not be demodulated by the analog front end.

Communication was established with Microchip and it was resolved that the AGC lock command occurring right during the preamble was not a coincidence. In fact at the time of this writing, Microchip is experiencing issues establishing a ground plane between the aggregator and node. Because of these issues, it was necessary to implement the more power expensive protocol discussed above.

# Bluetooth Low Energy Protocol

Once the information has reached the aggregator, this information is sent to the phone through Bluetooth Low Energy (BLE). We looked into different ways to send this data to the phone such as Wife, Classic Bluetooth, NFR and Zigbee, however BLE seemed to be the perfect fit. BLE is a new technology that is available today in newer smartphones. It is specifically designed to send small amount of data over longer distance. As its name specifies it also consumes very low amount of energy which makes it last for a long time with a single battery charge. The chip can also run on a coin cell battery which makes it very convenient.

There were many manufacturers and development kits available for Bluetooth low energy which included Texas Instrument, BlueGiga and many others. We chose the Nordic NRF51822 Bluetooth chip for our system because of the low cost of development and extended library and support. This chip comes with an ARM Cortex M0 Microcontroller which also provides us more computational power on the aggregator that is not available on the PIC. We communicated with the PIC on the aggregator using a simple UART protocol. Once the data got to the Nordic Chip it was sent to a BLE enabled smartphone through Bluetooth low energy.



Bluetooth low energy is a service based protocol where different peripheral devices connect to a master BLE device and get different services. Our system implements two services – one for temperature and one for pulse rate. Once more sensors are added to the BioComm System, more services would be available to the phone. If any BLE enable smartphone is not available then the Nordic Chip will continuously advertise until a smartphone connects to it.

# Phone Applications

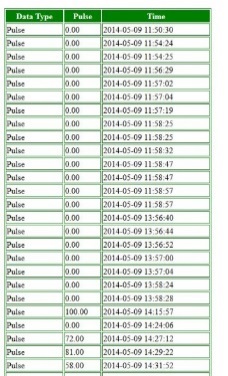
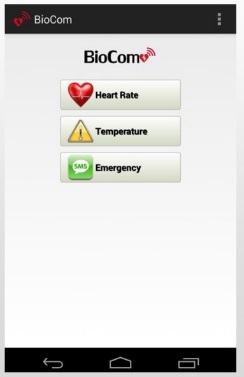
## Android

The Android application is created through the usage of Android SDK Tools. It is provided free of charge from google at: https://developer.android.com/sdk/index.html. Due to Android’s diversity, we hit the first hurdle right at the beginning. The Android SDK Tools require a large library of drivers in order to even allow the running of Eclipse, which in turn requires a special edition of Java Runtime Environment/Java Development Kit. This wouldn’t be a problem if you were running all your applications directly through the virtual machine. However, since the application will be eventually ported to a Samsung Galaxy smartphone, locating special drivers for the SDK and the Samsung drivers are required. With lots of tweaking and troubleshooting with the help of YouTube, eventually Eclipse ran without any errors. This allowed us to put a general idea of the application on the map as the details of the base structures were created.

As the base of the android application, we wanted to create the icon, and the main title screen. The focus of the first edition is to have the ability to pull up the heart rate and temperature data in one click of a button and in its place. In the image on the right sits a dummy graph that demonstrates if data was a pulled, it would graph all the points into a line scatter plot. This brings us to the second problem. Android supported multiple devices and most of the time, each have a different standard for screen size. So in order for the icon and the buttons to show clearly for each device, they must be resized for each type of screen size which varied from hdpi of small screen smartphone like the old blackberry screens all the way to xxxhdpi which now supports the phone/tablet of the Samsung Note II. After figuring the standard dimensions, the utilization of Photoshop is used to resize to that individually.

After some discussion, we decided to separate the iOS app for the patient and the android app for the doctor. This way, it can also alleviate the stress of both connecting to the Bluetooth and the SQL server. However, for the Android to not just connecting to the SQL server but also pushing data, brings us to the second main problem. This is one of the most important things next to the Bluetooth connection to the aggregator and the iPhone. For a while, the SQL database was created but none of us actually have tried to use it other than through a web page. This will also allow us to test not only the connection but the SQL database itself. The answer was finally with some help from the community at Stack Overflow forum. The result was the usage of Asynchronous Task which allowed the application to post and execute to the SQL server.

After some changes, we added in patient ID to be pushed and a pop up toast message that notifies us if the command was sent. The end result came to one of the final revision of the android application.

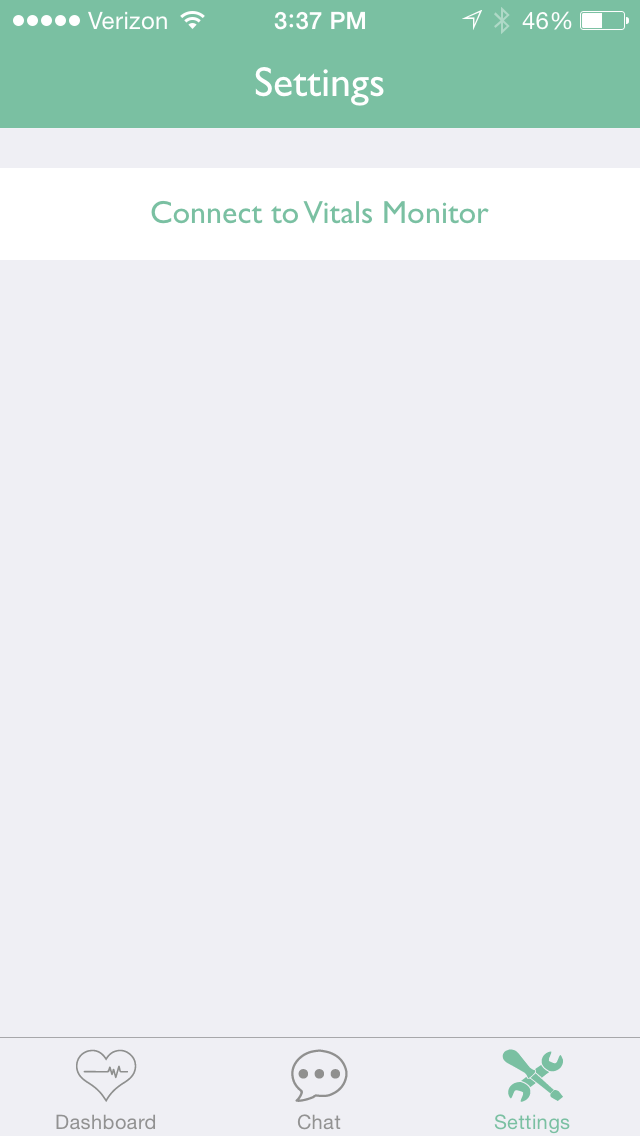
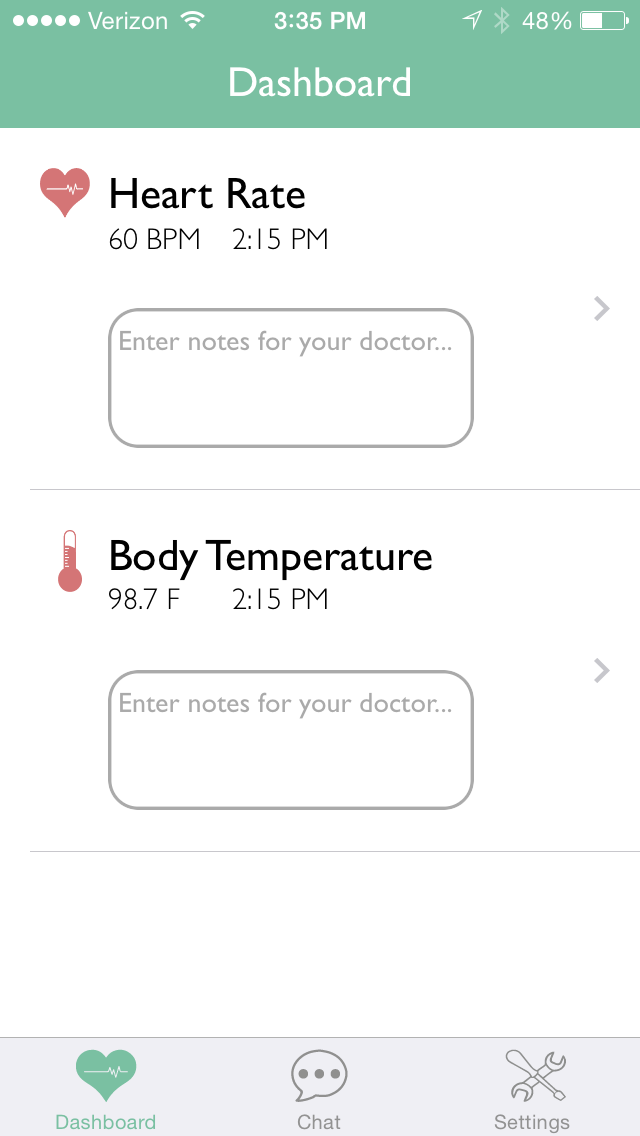


Above is the final rendition of the android application

The patient ID second is created so the doctor may enter in the ID and allow him to access the data collected for that specific patients. Next, it brings us to the main screen which has all new updated buttons with its visual updates. A texting button is added in this final rendition so if the doctor sees any anomalies in the patients reading, he/she can contact the patient immediately for a check-up. Finally, the last picture brings us the backend database where all the data is stored. Unfortunately, due to the formatting of the SQL coding, we were unable to figure out how to pull data in points and plot them on a graph. So instead, the application will access the PHP server directly via built-in web browser and access the patient data through a table format. In this case, we can see the type of data being pulse, the readings and the time stamp at when the readings were taken.

## iOS

An iOS consumer application was developed to allow the patient to see his/her vital signs and to facilitate relaying the information from the aggregator to the physician. While the aggregator could have potentially had another radio embedded in it such as Wi-Fi or LTE, it was decided that we could leverage a smartphone to provide the system with this service. This is mainly being that the smartphone is something that a user is already used to charging ever night. However, it was decided that the aggregator was not something that the user would feel comfortable charging, therefore most of its complex operations had to be slaved to the phone.

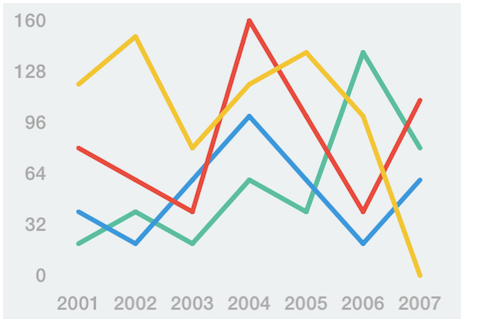
 Besides being able to process data and send the data to the physician, the smartphone application can also inform the user of his/her vital signs. While most of the time it is important to inform a physician, it is equally just as useful to inform the patient of his/her health. Therefore, they can make the right decisions given the circumstances. Because of this, most of the application user experience was designed to facilitate not only informing the patent and doctor of the state of the patients’ vitals, but also to facilitate greater communication between the physician and patient.

Design

Above is a set of screenshots that outline the application. In the main screen lies the main hub for the patient. Here is where they can view their vital signs, along with the time in which their vitals were taken. If the user clicks each tab bar, a graph appears to show the history of their vital signs. What is most neat about this portion of the interface is the notes section. Here is where a patient can inform their doctor about the events that were leading up to a vital measurement. For example, if the patient’s heart rate suddenly sky rocketed, the patient could inform the doctor that they went out on a run. This section is really to ensure that the doctor has as much information as they need to make an accurate assessment of the individual’s health.

The next page entails another communication hub for the physician and patient. Here is where the patient and doctor can have a more general discussion. While the aforementioned section was specifically for notes, this section is a space for question and answer between the patient and the doctor.

Lastly the settings view allows the patient to connect and disconnect to their vitals monitor to ensure that they are getting readings. In the future this section could implement battery level monitoring as well and other relevant information that the patient might need to know about their hardware device.

 A lot of care was taken into the visual elements of this design. While the design needed to be functional, it was also important that it was a space where the user would not be intimidated to enter. This way the patient gets their vitals data as quick as they need it. The application is really centered on the view of the patient data and therefore everything else is secondary. Through this a design was established that was friendly for the user to use. To aid in the construction of visual elements, several open source libraries were used. The libraries were obtained through Github and all dependency management was done through a dependency manager, CocoaPods. The open source libraries used in this project for the purposes of UI/UX are: FrameAccessor and GraphKit for the graphing capabilities and ChatController for the chat capability. These two libraries were particularly primitive in their abilities thus some time had to be devoted to rewriting the libraries code for the specific implementations of this project. The following figure shows a sample implantation of GraphKit’s graphing features.

While the data processing portion was the most critical to this application, most of the engineering time was spent in learning how to use the custom drawing and graphics libraries provided by Apple to achieve the look and feel that was desired for this application. The end result was a clean and polished application that gave the user a comfortable atmosphere to view their vital signs, which for some people could be a daunting experience in itself.

Data Processing

Careful design implementation were considered to ensure that the data processing portion of this application were completely separate from the visual code of the application. The two more important classes involved in data processing were the BCMBLEManager class and the BCMSensorManager class. The BLEManager class lies on the bottom of the applications architecture. It handles all input to the device and communication from Bluetooth Low Energy. From this the application can connect to a vitals monitor via BLE, search for the characteristics and services provided by the vitals monitor, then request to be notified on the changing of those values and furthermore receive data and update the higher layers that data has been received. A print out of the BCMBleManager class is available in the appendix. While most of the connection handling is boilerplate code for handling Bluetooth devices, Michael M. Kroll’s open source BLE library was referenced to learn how to do this. T

After the BLEManager receives and processes the data, it packs it into a packet with a time stamp and forwards it to the higher level of the applications code with is the SensorManager class. Here the class understands whether this packet came from a heart rate or body temperature monitor. It is here where the application achieves abstraction from the layer that deals with the Bluetooth driver and decides what to do with the data and how to interpret it. With this, if the communication method was ever changed, the only thing that would need to change in this application would be the BLE Driver in BCMBLEManager class. The rest of the application is abstracted from those details. In the appendix is a snippet of code from the BCMSensorManager class which receives a packet from the BCMBLEManager class, decodes whether it came from a temperature or heart rate node and then interprets the data as such. From this, it forwards the data to the back end data base. An example of data interpretation would be that the BLEManager class only receives raw ADC values from the Bluetooth chip on the aggregator. From this the BCMSensorManager class will run the sensors transfer function knowing that the data came from the temperature sensor.

The end product was a well-designed application both from the engineering and user perspective. From engineering perspective, open source libraries are utilized to get the maximum amount of power and flexibility and to reduce engineering time spent re-inventing the wheel. Also the application is elegantly architected such that if implementation details change as far as the communication to the aggregator is concerned, whether it be from Bluetooth or to Wi-Fi in the future, the entire application doesn’t have to change, only the BCMBLEManager class which acts as the Bluetooth driver for the application. From the user perspective, the application looks polished and clean and allows a comfortable and easy to navigate environment in which the user can view their vital signs and open up greater communication with their doctors.

# SQL Server/PHP Service

In order to have the doctor obtain data from multiple patients a MySQL database using a php web service to handle data push and pull was implemented. The design called for a database that was able to communicate with multiple devices using the same service. It had to at minimum be able to hold data for different patient and data types and allow the doctor to view different patient's data. To accomplish this the MYSQL database had 3 main tables, one for patient information, one for patient data, and one for doctor information. The patient data table holds data for patient Id #'s to allow index referencing between the data and the patient quickly, and data for the patient's name. The patient data table contains all the different types of data that would be collected from the different sensors, because it would require the creation of a new table every time a new sensor was added there is a field of the table that determines what kind of data each member of the table is. There are also fields that deal with data value, patientId number, and a timestamp field for when the data was taken. The doctor table contains information for the doctor and his password.

To make it so queries cannot just be sent to the MySQL server it was made to be ran locally hosted with a PHP service running on the same machine to allow communication between devices. The service takes information it receives from HTTP request using POST arguments to obtain data. By using HTTP it allows any device that can browse the internet to interact with the service. The service itself though requires certain POST arguments to be set in order for it to start interacting with the database. These arguments can be seen in the database Appendix D. When the correct POST arguments are set it will package the data passed along with the POST arguments is queries and then either push or pull data depending on the query. For instance, if you look at Database Appendix D.1 you can see that PatientId, DataType, and DataValue must be set for the service to push data, and then it will pack them into an SQL query, send it to the database, and commit changes to the database.

Pulling data uses the same type of interaction but we set different POST arguments and through this query the server for data. The service will then take the returned information from the server and send it back through the response portion of HTTP. It also will display it on the webpage of the service as a graph for when accessing it through a browser, which can be seen in database Appendix D 3.

Originally the server was running on a MSSQL server, but because PHP doesn't contain the calls for interacting with MSSQL by default it was switched over to a MYSQL server and allowed the database to be ran on SDSU's Unix Volta server. This was beneficial due to that it enabled us to have the data available more often and at a much faster rate than previous hosting. Other issues we ran into didn't involve the database as much as the amount of time we had to get devices communicating with the database. The service itself has several calls enabled for alert messages and creating new patients, but there wasn't enough time to get these less critical portions of the database working on the mobile devices. The table page shown in Appendix D.3 that was used for browsers was also used for the android application when pulling data, because of issues that arose with android receiving the correct response, and limited time constraints.

# Sensor Node Circuitry

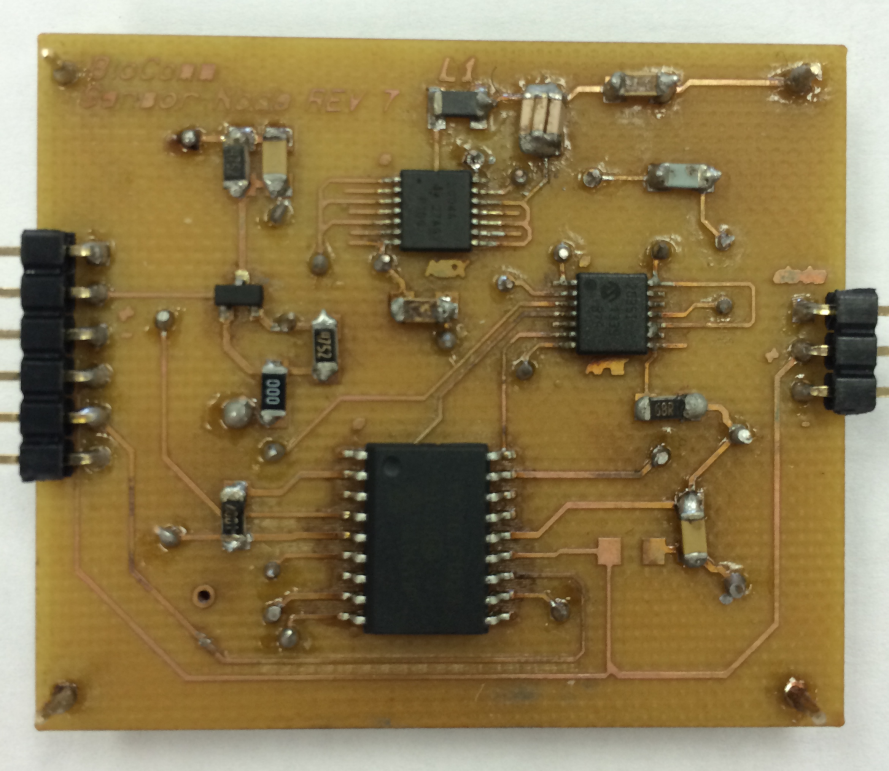
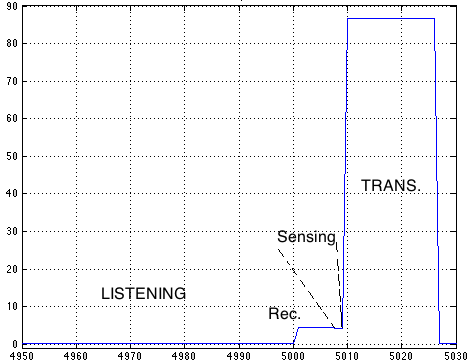
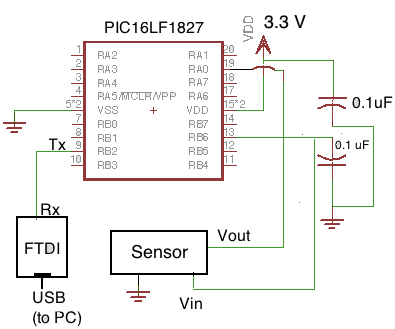
Before incorporating any designs, the functionality of the original BodyComm development kit provided by Microchip was tested. It operates using a CR2035 3V coin cell battery, and when functional, only one node can communicate with the aggregator: it receives 125 kHz signal from the body using a coupling pad placed on the bottom of the node, and emits an 8 MHZ signal to the body using the same coupling pad. The original circuit consists mainly of a MCP2035 Analog Front-End (AFE) device that shall be responsible of listening to and receiving the 125kHz signal from the body; a Hex Inverter, the SN74LVC04; and a microcontroller, Microchip’s extreme low power management PIC16LF1827 with nanoWatt XLP technology.

Figure - Sensor Node PCB

Once an understanding of the BodyComm kit was had, the communication protocol was then modified to an open architecture that would allow more than one node to communicate with the base unit. Once arbitrary data was transmitted and received between the sensor node and the aggregator the appropriate circuitry was added for both the temperature sensor and the pulse sensor, each integrated to their own node. Each node has the exact same circuitry as the original node’s circuit with a few exceptions. Two indication LED's were removed. For the Temperature and pulse sensor nodes the free pins were used to power the sensor directly for the microcontroller. Using Mentor graphics, a PCB layout was created for the two new individual nodes that contain both the original circuit and the sensors’ circuitry, shown in Figure 1. More detailed schematics can be seen in Appendix F.

With the efforts of finding an energy-harvesting device to power the nodes, and even after switching the plan to a coin cell battery, minimizing the node’s power consumption was a priority. Therefore, being able to control the sensing process by turning off the sensors when no measurements are needed can greatly serve our purpose. Thus, we decided to power the sensors immediately from one of the microcontroller’s GPIO Pins, as shown in the diagram of Figure-2. Pin13 of the microcontroller powers the sensor and the output of the senor is connected to Pin19 of the microcontroller, which is the input to the analog-to-digital converter (ADC). As measurements were be monitored frequent and large fluctuations were seen on the output of the ADC. These fluctuation were caused be high level noise in the system. To combat the system noise, 0.1 µF bypass capacitors were added at the power inputs to the sensors on the sensor node PCB and the sensor PCB. This greatly improved the stability of the output of the ADC and allowed accurate measurement to be conducted in order to greatly reduce the noise.



Power Consumption:

Figure -3: Typical Power Consumption (mW) Vs Time(ms)

Figure -2: Node’s simplified diagram

The node is subject to four main phases: listening, receiving, sensing, and transmitting. When listening, only the AFE is powered on and is in standby mode, while the microcontroller is in sleep mode and the sensor is powered off. The microcontroller wakes up when receiving data. The sensor is only powered on in the sensing phase, where the AFE is back to standby mode. In the transmission phase the sensor is powered off again. The power consumption in each phase is graphed in Figure -3 and represented in detail in the table of Figure -4. Most of the time the node is listening, and for the fact that all the devices in that phase are either off, in sleep mode, or in standby mode, a decent amount of energy is saved.

Temperature Sensor

In order to determine skin temperature a couple methods were researched. The sensor had to be ultra-low power, small, durable, reliable, and cost effective. The STLM20, a device from STMicroelectronics, was chosen due to its 8μA maximum quiescent supply current, small size, low price. One downfall for the STLM20 was its wide temperature range. To ensure completion of the project, a temperature sensor implementing a thermistor was designed as a failsafe but required 1.6mA during operation.

The STLM20 is contain in a SOT23-5L package utilizing three pins, Vcc, ground, and Vout. To help eliminate line noise on Vcc a 0.1μF bypass capacitor was added between Vcc and ground. Using Mentor Graphics, a PCB design was laid out in and printed to allow testing and characterization of the sensor. To determine the accuracy of the STLM20 a urethane conformal coating was added to the PCB and the sensor was placed in an emersion bath. A Fluke 52II was used as the calibrating device. With the assistance of a hot plate, the temperature of the bath was slowly increased from 21°C to 42°C. The test region of interest is 32°C to 39°C with steps of 0.1°C. Figure 4 shows the linear operation of the STLM20 over the operating range.

**Figure 4**

Vout was then entered into Equation 1 to determine the temperature of the bath.

Appendix E shows the output at each step along with the difference between the Fluke 52II and the percent difference. The average difference between the Fluke and the sensor was 0.046°C giving an average percent difference of 0.133%. The data shows that the sensor is accurate enough over the temperature range of interest. The one concern is the small voltage swing of 83mV between 32°C and 39°C. The STLM20 was also tested at two locations on the body, the inner-upper arm and mid-torso. Again the Fluke was used to as the standard. The arm location caused large fluctuations in Vout. This was due to the thermal venting around the arm during movement. The help combat the situation a thermally-doped silicon pad was placed on the sensor to act as a thermal couple. The torso provided more stable results and once accompanied with the thermal pad the results were as stable and as accurate as the emersion bath. The torso became the permanent location of the temperature sensor, but not just for measurement reliability. With two different sensors at two different locations, the functionality of BodyComm would be able to be tested at different parts of the body.

As a contingency, a less efficient design using a thermistor was tested which is shown below. The low resistance of this thermistor means that the supply current will need to be increased drastically as compared to the STLM20 but the voltage swing between the minimum temperature and the maximum increased drastically. Using a Wheatstone bridge and a differential amplifier the output voltage was measured against the temperature using an emersion bath. Figure 5 shows the data. The output voltage increases as temperature increase. Using this method the total voltage swing is 1.116V. This method would have utilized a look up table to determine the temperature saving power in terms of the microcontroller, but the overall power consumption of the thermistor design would not be recovered. The total current used by this design would be 1.6μA. This design will only be used as a last resort.

**Figure 5**

# Pulse Sensor

Initial designs used a low power thin-film piezoelectric sensor for pulse rate. Multiple designs of varying amplification and filtering were used, including active and passive filtering using operational amplifiers, however too much noise was present to distinguish intelligent pulse data. Following designs made use of a buzzer type piezoelectric device as initial tests proved a more reliable signal. However similar symptoms of noise existed in which extensive filtering could not eliminate. Further research investigation suggested using an optical device, but at the expense of power consumption. This being a piezoelectric device which consumes tens of microamps of current versus LED devices that consume in upwards of 100mA. An inexpensive commercial sensor was later obtained as a contingency. Additionally, a PCB was designed similar to the commercial product to meet design requirements, however power consumption was exceedingly high, and thus our method for obtaining an accurate pulse rate utilized the commercial optical sensor.

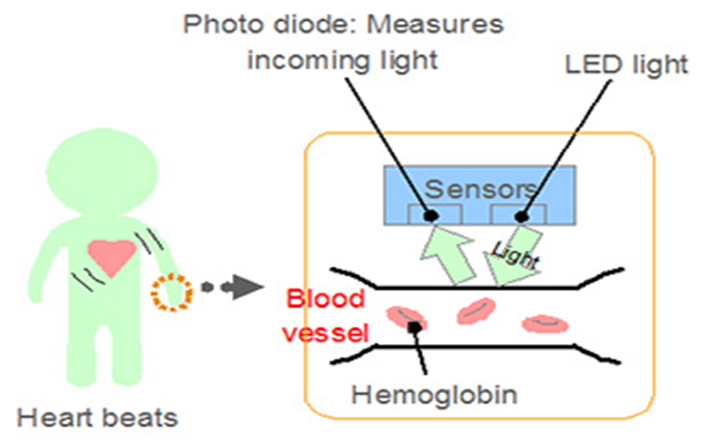
The optical sensor satisfies design needs of being low-power, accurate, and minimally invasive. It is a 3 pin device (Vcc, Gnd, and Vout) operating from 3 to 5 volts, has a maximum current draw of 4.5mA, and is approximately the size of a dime (Figure 6). Sallen-Key third order filtering is used to isolate desired frequencies pertinent to an average pulse rate range, and includes a passive low-pass and active high-pass filter. This range is 0.8 to 4Hz, which equates to approximately 42 to 240 beats per minute (bpm).This sensor is LED (565nm) driven and detects via a photodiode. As the heart beats, blood courses through the veins of the human body. The blood density naturally varies as it flows, more specifically the density of hemoglobin, which are the oxygen transport in red blood cells of all warm-blooded vertebrae. Since the heart beats in a rhythmic fashion, thus the density varies as such. The process for detecting heart beat begins with the LED emitting light into the epidermis (Figure 7).

Figure 6

The light emits into the layers where blood vessels reside. The reflected light from the hemoglobin is detected by the photodiode, and is linearly proportional to the density of hemoglobin in the blood vessel. The varying density equates to a proportional voltage output from the sensor, which is delivered to the PIC microcontroller. Processing occurs onboard the PIC before transmission to the Aggregator unit. The code functionality can be seen in Appendix B.

Figure 7

The processing functions on the basis of a 500Hz sampling rate, or an interrupt service routine every 2ms. The sensor output voltage into the ADC is seen in Figure 8. For improved resolution a 10-bit ADC value is used. The maximum voltage, *P*, represents the maximum hemoglobin density while the minimum similarly is represented as the trough, or *T*. To ensure a true heartbeat, and avoid unnecessary dichroitic noise, a 50% amplitude threshold detection, based on *P* and *T*, is set. Detection occurs on the rising edge, and 10 successful pulses are stored to take a running average. The final running average data is transmitted to the Aggregator.



Figure 8

Testing functionality included using a function generator to simulate a cardiac signal at 1.4Hz, or 84bpm, into the PIC and reading data using a UART. Test site configurations varied between the finger and inner arm bicep. Testing on the finger was optimal and little noise was present. The upper arm was ideal for our product, and testing revealed accurate results, however the signal was more susceptible to noise from body movements such as bone collisions and the flexing of tendons. This site location concluded that for concept purposes, nominal testing should occur while the patient is at rest. Comparing the simulated against live testing on the patient resulted in accuracy of ±1bpm, satisfying design goals.

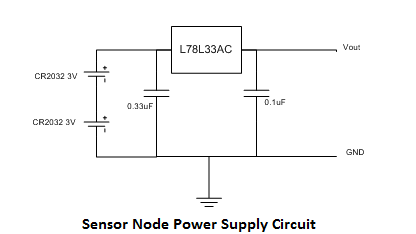
# Energy Harvesting/Power Supply

Our device was designed to be worn for extended periods of time without a lot of user upkeep maintenance. This means that we wanted to minimize charging or the replacement of batteries. To accomplish this we focused on keeping the system as low power as we could. Another aspect of our design was to eliminate the use of interconnecting wires for our device. This meant that each sensor node and the aggregator had to have their own power supply.

Our sensor nodes use the PIC16LF1827 microcontroller which has been designed to be low powered. This component comes with nanoWatt XLP Technology which allows it to have extremely low-power management. It has a sleep mode that draws only 30nA, a watch dog timer that draws only 500nA and the Timer1 oscillator draws 600nA at 32kHz. The microcontroller is also capable of operating between 1.8V to 3.6V. The sensor node circuit also contains an analog front end device, MCP2035-I/ST, which senses signals from the aggregator. The analog front end uses 2 to 5 µA while waiting for a signal and 10µA while operating. There is also a Hex Inverter, SN74LVC04APWR, which is used to transmit signals to the aggregator. This component is only powered while transmitting and draws 50mA.

A sensor is also connected to the sensor node controller board through headers which are connected to the microcontroller GPIO pins. Since the power going to the sensors is controlled by the microcontroller the sensors are only powered for the time to take a reading. For sensors we used a temperature sensor, STLM20, and an open sourced pulse sensor design. The temperature sensor draws 4.8 to 8µA and the pulse sensor draws 20mA when they are in operation.

Initially we wanted to use some form of energy harvesting to power the sensor node because we had such a low current draw and would be operating at only 3.3V. Research was done on Seebeck and Peltier devices which would be used as thermoelectric generators. The problem that we had with these devices is that we would need to operate with a very small temperature difference compared to the devices specifications. Also the temperature difference would depend on the wearer’s skin temperature and the environment that they were in. Thermoelectric generator technology was not a viable direction for our project. We also looked into using a piezoelectric generator but because we didn’t want wires and due to the fact that the mobility of the consumer might be an issue we decided to not use it. One option that we did like was using ambient RF energy to power our sensor nodes.

We found a RF energy harvester by Powercast that produced a regulated output voltage up to 5.25V and that had a 50mA output current. This was the P2110 – 915MHz RF Powerharvester™ Receiver. It had an operation down to -11.5dBm input power and we believed that it would be perfect. We also thought that we could tune it to another frequency band if we didn’t have enough ambient energy at the 915MHz band. The operating band of the harvester also included 850MHz which is used by AT&T for cell phones, so we thought that we would be fine. We purchased the development kit for the harvester and began experimenting to verify its operation. We came to find that there was not enough ambient energy for our device to harvest. Doing some more preliminary research may have allowed us to realize this sooner. We decided to try and re-tune the device but found out once we contacted Powercast that it could not be re-tuned by us and that they could tune if for us but it would cost $500. Due to our budget this was not an option. We also tried to use a transmitter to power the device but we could only get a range of 6 feet. We decided that we did not have the time or resources to devote more time to the RF harvester and so it was abandoned.

Since we still needed to power our devices we decided to go with our fall back plan which was to use coin cell batteries. The BodyComm development kit nodes used a single 3V CR2032 as a power supply. Our design included the use of sensors that required a higher voltage and because we wanted to have an extended battery life we decided to use something a little more robust. We also wanted to implement a regulated voltage source to prevent voltage droop from affecting our design. Our final design was to use two 3V CR2032 coin cell batteries in series which would give us a 6V nominal output. Then we used a voltage regulator, L78L33AC, to bring the voltage down to 3.3V for our circuit to operate off of. By having a regulated supply we had a more stable circuit response and improved the reliability of or device.

The aggregator is a bit more robust and so the power supply needed to handle all of the transmitting through the BLE and also the communication with the multiple sensor nodes. A further breakdown of the aggregator power needs can be seen in the appendix. A lithium-polymer battery was chosen because it is robust and is rechargeable. The battery will be connected directly to the aggregator PCB and will be housed in the same casing.

# Aggregator

As is common among all senior design projects at San Diego State, it is necessary to design, layout and fabricate a printed circuit board. The purpose of the aggregator printed circuit board was to miniaturize the circuitry in order to make the overall device as small and non-intrusive as possible to the wearer’s daily life. Furthermore, to make the device as low power as possible, it was necessary to minimize the number of features on the board.

Using Microchip’s BodyComm system as a basis from which to design, the initial steps taken were those to minimize the overall power consumption of the device. This was done by determining which of the original features were necessary and which were not. As an initial measure, the USB Bridge (MCP2200-I/SS), the 12MHz oscillator driving the FTDI chip, the MCP1703-5002E/MB 5V LDO regulator, the 8.192MHz crystal, and the LCD screen present on the original development kit were all removed along with all of the accompanying supporting circuitry. Furthermore, in an effort to further reduce power draw, the transmit and receive paths of the system were driven with 3.3V as opposed to the 5V used on the development board. Also, to simplify the board all of the push-buttons on the original development kit board were removed. They were deemed not necessary as there is no outside trigger to send signals from the aggregator itself. The important features of the system, however were left intact.

After obtaining the trimmed down version of the circuit, several features had to be added to achieve the design goals set forth for the project. The main feature added to the initial prototype was a Bluetooth Low Energy module (MBH7BLZ02). Also, this being a prototype board, it was necessary to be able to probe several of the pins on each of the different chips on the board. Therefore, the MCP2035 Analog Front-End, the PIC16LF1829, and the MBH7BLZ02 had header pins connecting to the primary signal paths on each of the chips. Also, in order to be able to adequately test the function of the board, the overall circuit was made excessively large, thus the form factor of the prototype board was 6”x4”. The schematic and layout for the prototype board can be found in Appendix H.

Due to the fact that the boards that were created at San Diego State did not come with plated vias, measures were taken to reduce the number of vias overall on the board. In order to accomplish this, zero ohm resistors were used throughout the board as jumpers. These resistors were not included in the original schematic, however they were as needed in order to facilitate laying out the printed circuit board.

The prototype printed circuit board was then tested, and it was verified that the transmit and receive paths on the board, along with the digital circuitry performed satisfactorily. There were several points brought up during its testing, however. It was determined that 3.3V was not sufficient voltage to power the transmit and receive paths in the system, thus it was necessary to put those voltages back to 5V. This required the addition of another linear voltage regulator; the UA7805C from Texas Instruments was chosen to perform this task and added to the board along with the necessary bypass capacitors which were necessary to filter the DC power. Also, the additional functionality of charging the intended Li-Polymer battery was specified for the board. To accomplish this, it was necessary to add a dedicated Li-Polymer battery tender IC (BQ2057WSN), a USB port, as well as a power MOSFET (NDS8434). This final circuitry was then reduced in size from the original 6”x4” to a much smaller 3”x2”. As stated previously it was necessary to miniaturize the circuit as much as possible in order to make the device comfortable, wearable, and non-intrusive. The schematic and layout for the final aggregator can be found in Appendix H.

Another concern that arose from the original prototype was that the BLE would be introducing massive amounts of RF energy across the entire board. This could potentially corrupt the very low power signals that were being transmitted and received by the aggregator unit. In order to minimize noise introduced by the Bluetooth Low Energy module, it was determined that the module should be isolated from the rest of the board, on its own printed circuit board. Furthermore, it was also determined that programming the BLE module only required two pins, the SWDIO and the SWDCLK pins on the MBH7BLZ02. Therefore, those two pins, along with a single UART data pin, as well as 3.3V and ground supplied by the aggregator module were the only pins necessary for the operation of the BLE module. As a precaution, more pins were made accessible in case more were required, if they were deemed unnecessary, header pins were simply not soldered into their respective holes. As with the rest of the board, it was necessary to make the printed circuit board for the module as small as possible. In designing the board, though, exceptions had to be made for the built in antenna on the module, it could not be obstructed by the copper cladding on the board itself, therefore it was necessary to have no traces or ground planes in the vicinity of the antenna. This optimized the performance of the antenna and ultimately increased range. Also, due to the fact that it was isolated from the rest of the aggregator and being powered through jumper wires from the aggregator, it was necessary to provide bypass capacitors close to the chip to ensure a clean DC power source to the BLE module. Schematics and layout of the Bluetooth Low Energy printed circuit board can be seen in Appendix H.

The last part that was designed isn’t exactly a printed circuit board, though creation of the board was conducted through the use of Mentor Graphics Expedition. That part is the capacitive coupling pad. It acts as the antenna of the system, launching electromagnetic waves into the surface of the skin to the sensor nodes distributed throughout the body, and then subsequently receiving signals from the nodes. It was realized that the circuitry on the aggregator was tuned to a particular capacitance, that value being the accepted value of the capacitive coupling pad. Therefore, the custom coupling pad was designed to mimic the uncoupled capacitance of the Microchip version of the pad. Its design consists of interdigital fingers of copper approximately fifty mils wide, separated by approximately 50mils. When the custom pad was initially utilized in the system, it was realized that the coupled capacitance of the board was far larger than it should’ve been. Therefore, it was determined that the coupling pad required a conformal coat to prevent the skin of the user from shorting the signal directly to ground. As a necessity of Mentor Graphics Expedition, a schematic was required to create the layout of the coupling pad. That design can be seen in Appendix G.

# Sensor Bands

The packaging of the sensors needed to be comfortable, durable, and reusable. An elastic material was selected to make the bands that would go on the arm and the chest. The armband would house the pulse sensor, controller PCB, and the power supply. The chestband would house the temperature sensor, controller PCB, and the power supply. Figure 9 shows the inside of the chestband (top) and the outside of the band (bottom). The coupler pad was placed outside of the band for the chestband since it was worn over a shirt for demo purposes and under normal circumstances would be mounted inside the band. The chestband would normally be worn on the skin under clothing, which is how the armband is worn. Each band consists of one large pocket lined with Velcro that holds the parts in place. The ease of access allow for simple sensor change outs or the quick swaps of faulty parts of dead batteries. The bands are adjustable and washable allowing for multiple uses by different patients lower the cost and need for material.

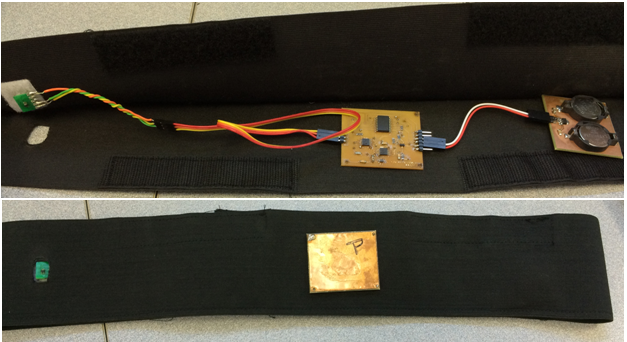
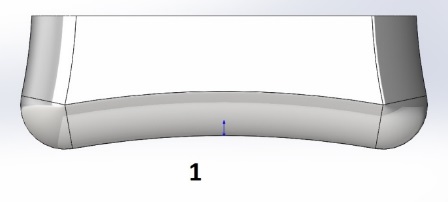
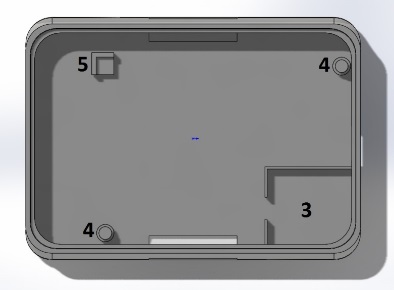
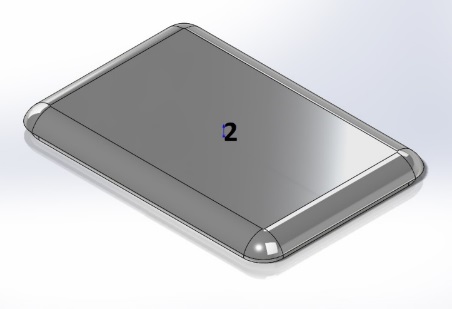


Figure 9

# Aggregator Case

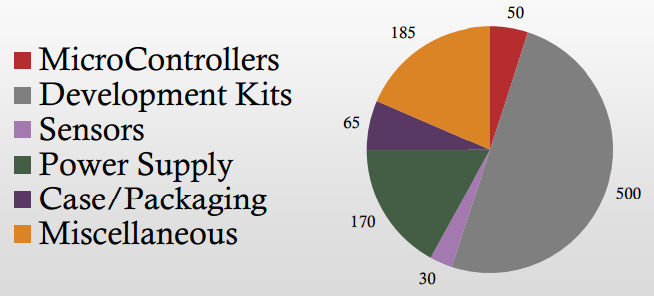
The aggregator is made up of four individual parts: the main PCB, the BLE PCB, the capacitive coupling pad and the battery. All four parts were to be assembled within the case for the user to wear, with the exception of the capacitive coupling pad being connected from the outside of the case. The aggregator case was designed to be small, light and non-invasive as possible for the user’s convenience. To design the case, the program SolidWorks was chosen for designing because it was simple to use and gave a 3 dimensional display of the design. For fabricating the case, plastic was the best choice because it was light and cheap to use. Also the SDSU engineering department also has its own industrial 3 dimensional printer that can use so it was very easy to build the case for the aggregator. In order to ensure that the device was comfortable to wear, a slight concave curvature (1) was incorporated to the bottom side of the case. The lid (2) to the case was designed in such a way to provide a mounting location for the battery.

Furthermore, in order to maintain isolation between the aggregator printed circuit board and the Bluetooth Low Energy PCB, it was necessary to include walls(3) between the two boards. If necessary, metal foil could have been incorporated into the case to further improve isolation. Unfortunately, due to the fact that the battery was very large in comparison to the aggregator board, the case has excessive room.

What proved to be the most challenging was getting the measurements just right to mount the main PCB in the casing. The main PCB had two holes that are used for holding it in place without having it move within the case. Two mounting columns (4) were created to hold the PCB in place by using those holes. The mounting columns were also designed to hold the main PCB up right so the USB that was beneath the PCB did not interfere with inside surface of the case. Also a spare mounting platform (5) was created to add more support to the PCB and two walls were created to insure that the PCB does not move around the case.

# Budget

The budget adhere to team BioComm was a total of $1000 and was split to $500 to each team; Team 1 hardware and Team 2 software. Each team had different product in order to complete the project. As seen on the figure below the pie chart illustrates how the budget was allocated throughout the project. In order to stay within the $1000 budget the team was able to use school devices such as capacitors and resistors. Additionally money was saved with the ability of using the PCB (Printed Circuit Board) machine for all the desired circuit boards needed. Not only did this allow for the elimination of outside companies but it also allowed allocating a majority of the budget to the developmental kits needed for the project to succeed.

As mentioned prior in the report it was vital for the team to plan contingencies as issues may occur. This was the $185 of the budget set aside in order to prepare for these issues. The energy harvesting unit was part of the power supply budget constraints. However, as mentioned earlier in the repot the energy harvesting unit did not work as desired and coin cells where used as the team’s contingency. Not only was the miscellaneous portion of the budget set aside to plan for contingencies but also any extra parts that needed to re-ordered.

Money was also saved by recreating commercial devices used in the project. This was a clear advantage as parts are less cost effective then purchasing a $25 unit compared to $10 for the recreation. This had a large part of the ability to having printed circuit boards available at no cost. This was also advantages by allowing manipulate devices to adhere to the completeness of the project and design.

With the many specifications needed for this project the team was able to stay under budget by a total of $100. This was a great success as we had unexpected outcomes from previous designs and delays.

# Milestones

|  |  |  |
| --- | --- | --- |
| Date | Milestones | Completion |
| March 25 | One Node Communicates with Aggregator | http://www.clipartbest.com/cliparts/RcA/KGn/RcAKGnLXi. |
| April 1 | Multiple Nodes Communicate with Aggregator | http://www.clipartbest.com/cliparts/RcA/KGn/RcAKGnLXi. |
| April 3 | Temperature Sensor Operational | April 10 |
| April 7 | BLE communication with Smart Phone | http://www.clipartbest.com/cliparts/RcA/KGn/RcAKGnLXi. |
| April 7 | Power Supply Operational | April 17 |
| April 14 | Mobile device saves to Back End Database | http://www.clipartbest.com/cliparts/RcA/KGn/RcAKGnLXi. |
| April 14 | Temperature sensor Handshakes with sensor node | April 18 |
| April 18 | Pulse Sensor Operational | http://www.clipartbest.com/cliparts/RcA/KGn/RcAKGnLXi. |
| April 18 | Sensor Base Station Communications | April 19 |
| April 21 | App Retrieves and Display information for Database | April 22 |
| April 28 | Aggregator board miniaturization completed | http://www.clipartbest.com/cliparts/RcA/KGn/RcAKGnLXi. |
| May 5 | Functional Prototype | http://www.clipartbest.com/cliparts/RcA/KGn/RcAKGnLXi. |

These are the milestones we sought to achieve and have completed by specified dates. However, due to some challenges and failures we were ultimately delayed by the subsequent contingencies plans we had to implement, which prevented us from reaching previously established milestones.

From the chart above, our first issue arose on April 3 with the temperature sensor; we initially could not obtain the right degree of accuracy due to the locating of the temperature sensor on the brachial artery. From testing we found that moisture and any sudden movement caused the sensor to display unusable data. By moving the temperature sensor to the chest and adding a thermal couple we were able to resolve the issue, which was completed on April 10.

The next issue we encountered was on April 7 with the power supply. Initially we wanted to use RF Energy harvesting to power our BodyComm unit but from initial tests, we found that the frequency band, for which the development kit was tuned, did not provide sufficient amount of power needed so we decided to go with the next feasible option which involved using two coin cell batteries. And that was implemented on April 17.

Another challenge that occurred for us was on April 14, when we could not establish a data link between the temperature sensor and the sensor node. Rewriting and eliminating errors within the program code on the sensor node in order to establish the right protocols between the two units resolved this issue. That was completed on April 18th.

On the same day, a similar protocol issue arose again. However, this time, it was with the sensor base station. Fortunately, this only required a minor adjustment by rewriting and eliminating errors within the program code and that was completed on the April 19.

Lastly on April 21, we had one minor issue that involved the App Retrieves and Display information not uploading to the database. This involved a simple correction in the RTX programming code and that was completed on April 22.

With any creation of a legitimate product it requires much trial and error nonetheless we are pleased with our results and end product.

# Conclusion

Medical technology is advancing on a daily basis. The next step is improving doctors’ ability to track their patients’ medical histories. Doing so will greatly improve their ability to diagnose conditions present in their patients, but more so, it will allow doctors to prevent conditions from developing in the first place by allowing them to see biometric data relevant to their patient on a steady basis.

The device presented here performs exactly that task. It non-invasively measures temperature and pulse wirelessly, then it sends it off to an offsite database. The doctors can now access this database and view their patients’ vital signs in real time.

Through this device, patients are afforded the freedom of not having to come into the hospital just for routine post-operative check-ups. Instead, in the comfort of their own homes, their doctors are able to monitor them and catch any mishaps before they occur.

Shown with this device are two example sensors, temperature and pulse. This system can easily be expanded to provide a greater variety of vital sign measurements. Soon, measurements that required trips to the hospital, i.e. electrocardiograms, electroencephalographs, blood oxygen content, or blood pressure could all be incorporated with this system. This would make trips to the hospital exceptionally rare, but actually increase the overall health of the patients using the device.

# Appendix A – Communication Protocol

Protocol Appendix A1

uint8\_t BC\_SendData(BodyCom\_Commands\_t cmd, uint8\_t \* Address, uint8\_t timeout,uint8\_t Length, uint8\_t \* Data) {

// Create temporary packet

MDLL\_PacketData\_t tempPacket;

// Reset timer

BCOM\_cmdTimer = 0;

// Load timemout value

BCOM\_cmdTimeout = timeout;

// Build packet

tempPacket.Command = cmd;

tempPacket.Address[0] = Address[0];

tempPacket.Address[1] = Address[1];

tempPacket.Address[2] = Address[2];

tempPacket.Address[3] = Address[3];

tempPacket.DataLength = Length;

for(int i = 0; i<Length; i++)

{

tempPacket.DataBuffer[i]= Data[i];

}

// Send packet

return MDLL\_sendPacket(&tempPacket);

}

Protocol Appendix A2

else if (MDLL\_PacketDataBuffer.Command == ECHO\_REQUEST) {

// Copy the entire packet

memcpy(&tempPacket, &MDLL\_PacketDataBuffer, sizeof (MDLL\_PacketData\_t));

// And replace only the command and address to be sure

tempPacket.Command = ECHO\_RESPONSE;

tempPacket.Address[0] = MDLL\_DeviceAddress[0];

tempPacket.Address[1] = MDLL\_DeviceAddress[1];

tempPacket.Address[2] = MDLL\_DeviceAddress[2];

tempPacket.Address[3] = MDLL\_DeviceAddress[3];

tempPacket.DataLength = 4;

if(MDLL\_DeviceAddress[3]==0x06)

tempPacket.DataBuffer[0] = TEMPERATURE;

else if (MDLL\_DeviceAddress[3]==0x05)

tempPacket.DataBuffer[0] = HEARTRATE;

tempPacket.DataBuffer[1] = higherValue;

tempPacket.DataBuffer[2] = lowerValue;

tempPacket.DataBuffer[3] = counter;

// Send Packet

counter++;

return MDLL\_sendPacket(&tempPacket);

}

# Appendix B - BLE

1. **Nordic Chip Code – main function**
2. **int** main(**void**)
3. {
4. uint32\_t err\_code;
6. // configure uart
7. simple\_uart\_config(RTS\_PIN\_NUMBER, TX\_PIN\_NUMBER, CTS\_PIN\_NUMBER, RX\_PIN\_NUMBER, HWFC);
9. timers\_init();
10. gpiote\_init();
11. buttons\_init();
12. ble\_stack\_init();
13. bond\_manager\_init();
15. // Initialize Bluetooth Stack parameters
16. gap\_params\_init();
17. advertising\_init();
18. services\_init();
19. sec\_params\_init();
21. // Start advertising
22. advertising\_start();
24. // Enter main loop
25. **for** (;;)
26. {
27. // continuously check for data from uart
28. get\_uart\_data();
29. }
30. }

**2. Bluetooth Low Energy – Code that gets data through UART and sends it through BLE.**

1. **void** get\_uart\_data()
2. {
3. uint8\_t rx\_data[MAX\_TEST\_DATA\_BYTES];
4. uint8\_t rd;
5. uint8\_t count;
7. **while**(**true**)
8. {
9. rd = simple\_uart\_get();
10. **if**(rd == 0x0A) // filtering garbage data from aggregator touchpad
11. {
12. // get data
13. **for**(count = 0; ( rd = simple\_uart\_get() ) != 0x0D ; count++ )
14. rx\_data[count] = rd;
16. // sent to uart for debug
17. **for**(uint8\_t i = 0; i < count; i++)
18. simple\_uart\_put(rx\_data[i]);
20. **if**(rx\_data[0] ==    TEMPERATURE\_DATA) // first byte is the type of measurement
21. {
22. m\_cur\_temp = \*(uint16\_t\*)(&rx\_data[1]); // get temperature data
23. }
24. **else** **if**(rx\_data[0] == HEART\_RATE\_DATA)
25. {
26. m\_cur\_heart\_rate = rx\_data[2]; // get heart rate data (needs only one byte)
27. }
28. }
29. }
30. }

**3. Bluetooth Low Energy - code that handles BLE Events**

1. **static** **void** on\_ble\_evt(ble\_evt\_t \* p\_ble\_evt)
2. {
3. uint32\_t        err\_code;
4. **static** uint16\_t m\_conn\_handle = BLE\_CONN\_HANDLE\_INVALID;
6. **switch** (p\_ble\_evt->header.evt\_id)
7. {
8. **case** BLE\_GAP\_EVT\_CONNECTED:
9. led\_stop();
11. m\_conn\_handle = p\_ble\_evt->evt.gap\_evt.conn\_handle;
13. // Initialize the current heart rate to the average of max and min values. So that
14. // everytime a new connection is made, the heart rate starts from the same value.
15. m\_cur\_heart\_rate = 0xFFFF;
17. m\_cur\_battery\_level = 100;
19. // Start timers used to generate battery and HR measurements.
20. application\_timers\_start();
22. // Start handling button presses
23. err\_code = app\_button\_enable();
24. APP\_ERROR\_CHECK(err\_code);
25. **break**;
27. **case** BLE\_GAP\_EVT\_DISCONNECTED:
28. // Since we are not in a connection and have not started advertising, store bonds
29. err\_code = ble\_bondmngr\_bonded\_centrals\_store();
30. APP\_ERROR\_CHECK(err\_code);
32. // start advertising if disconnected
33. advertising\_start();
34. **break**;
36. **case** BLE\_GAP\_EVT\_SEC\_PARAMS\_REQUEST:
37. err\_code = sd\_ble\_gap\_sec\_params\_reply(m\_conn\_handle,
38. BLE\_GAP\_SEC\_STATUS\_SUCCESS,
39. &m\_sec\_params);
40. APP\_ERROR\_CHECK(err\_code);
41. **break**;
43. **case** BLE\_GAP\_EVT\_TIMEOUT:
44. **if** (p\_ble\_evt->evt.gap\_evt.params.timeout.src == BLE\_GAP\_TIMEOUT\_SRC\_ADVERTISEMENT)
45. {
46. led\_stop();
48. nrf\_gpio\_cfg\_sense\_input(HR\_INC\_BUTTON\_PIN\_NO,
49. BUTTON\_PULL,
50. NRF\_GPIO\_PIN\_SENSE\_LOW);
52. nrf\_gpio\_cfg\_sense\_input(HR\_DEC\_BUTTON\_PIN\_NO,
53. BUTTON\_PULL,
54. NRF\_GPIO\_PIN\_SENSE\_LOW);
56. system\_off\_mode\_enter();
57. }
58. **break**;
60. **default**:
61. // No implementation needed.
62. **break**;
63. }
64. }

# Appendix C – iOS App

- (void)peripheral:(CBPeripheral \*)peripheral didUpdateValueForCharacteristic:(CBCharacteristic \*)characteristic error:(NSError \*)error {

BtLog(@"");

CBUUID \*match = [CBUUID UUIDWithString: BCMHeartRateCharacteristicString];

CBUUID \*tempMatch = [CBUUID UUIDWithString:bcmBodyTemperatureCharacteristicString];

BCMBLEDataPacket \*dataPacket = [[BCMBLEDataPacket alloc] init];

uint16\_t dataFormatted;

NSDateFormatter \*dateFormat = [[NSDateFormatter alloc]init];

[dateFormat setDateFormat:@"h:mm:ss a"];

NSString \*dateString = [dateFormat stringFromDate:[NSDate date]];

NSData \*data = [characteristic value];

if([characteristic.UUID isEqual:match])

{

dataFormatted = \*(uint8\_t\*)([data bytes]);

dataPacket.fromShield = YES;

dataPacket.characteristicUUID = characteristic.UUID;

dataPacket.data = data;

dataPacket.formattedDate = dateString;

dataPacket.date = [NSDate date];

dataPacket.dataFormatted = dataFormatted;

dataPacket.characteristicUUID = characteristic.UUID;

[[NSNotificationCenter defaultCenter] postNotificationName:NOTIFICATION\_BLE\_SHIELD\_CHARACTERISTIC\_VALUE\_READ object:dataPacket];

//[peripheral readValueForCharacteristic:self.tempCharacteristic];

}

else if([characteristic.UUID isEqual:tempMatch])

{

dataFormatted = \*(uint16\_t\*)([data bytes]+1);

if(dataFormatted != 0xFFFF){

NSLog(@"temp rate update");

dataFormatted = CFSwapInt16BigToHost(dataFormatted);

dataPacket.fromShield = YES;

dataPacket.characteristicUUID = characteristic.UUID;

dataPacket.data = data;

dataPacket.formattedDate = dateString;

dataPacket.date = [NSDate date];

dataPacket.dataFormatted = dataFormatted;

dataPacket.characteristicUUID = characteristic.UUID;

[[NSNotificationCenter defaultCenter] postNotificationName:NOTIFICATION\_BLE\_SHIELD\_CHARACTERISTIC\_VALUE\_READ object:dataPacket];

}

}

//Formatting of data happens here.

//Based on sensor, able to apply transfer function.

-(void)sensorValueUpdated:(NSNotification \*) note

{

CBUUID \*match = [CBUUID UUIDWithString: BCMHeartRateCharacteristicString];

CBUUID \*tempMatch = [CBUUID UUIDWithString:bcmBodyTemperatureCharacteristicString];

BCMBLEDataPacket \*packet = [note object];

if([packet.characteristicUUID isEqual:match])

{

BCMSensorData \*sensorUpdated = [self.sensors objectForKey:BCMHeartRateCharacteristicString];

NSNumber \*reading = [NSNumber numberWithFloat:packet.dataFormatted];

sensorUpdated.lastRecordDate = packet.formattedDate;

sensorUpdated.valueUnsigned = packet.dataFormatted;

NSDate \*date = packet.date;

[sensorUpdated.collectionOfValues addObject:reading];

[sensorUpdated.dateTimes addObject:date];

NSString \*dataString = [self getRawHexString:packet.data];

NSLog(@"Got data from heart rate Characteristic");

NSLog(dataString);

[self pushToDatabase:sensorUpdated];

}

else if([packet.characteristicUUID isEqual:tempMatch])

{

BCMSensorData \*sensorUpdated = [self.sensors objectForKey:bcmBodyTemperatureCharacteristicString];

uint16\_t rawSensorValue = packet.dataFormatted;

float sensorVoltageValue = rawSensorValue\*tempSensorADCStepSize;

float bodyTemperatureCel = sqrtf((2196200)+(((1.8639 - sensorVoltageValue)\*1000000)/(3.88))) -1481.96;

NSNumber \*reading = [NSNumber numberWithFloat:bodyTemperatureCel];

sensorUpdated.value = bodyTemperatureCel;

sensorUpdated.lastRecordDate = packet.formattedDate;

sensorUpdated.valueUnsigned = packet.dataFormatted;

NSDate \*date = packet.date;

[sensorUpdated.collectionOfValues addObject:reading];

[sensorUpdated.dateTimes addObject:date];

NSData \*data = packet.data;

NSString \*dataString = [self getRawHexString:packet.data];

NSLog(@"Got data from Body Temp Characteristic");

NSLog(dataString);

[self pushToDatabase:sensorUpdated];

}

NSData \*newReading = packet.data;

NSString \*newReadingString = [self getRawHexString:newReading];

uint8\_t v = 0;

[[packet data] getBytes:&v length:sizeof (v)];

[[NSNotificationCenter defaultCenter] postNotificationName:@"NOTIFICATION\_SENSOR\_CHANGED\_READING" object:nil];

}

# Appendix D – Database

Database Appendix.1

//############# Pushes New Data to the server########################

if (isset($\_POST["PatientId"]) && isset($\_POST["DataType"]) &&isset($\_POST["DataValue"]))

{

$PatientId = $\_POST["PatientId"];

$DataType = $\_POST["DataType"];

$DataValue = $\_POST["DataValue"];

$this->db->query("INSERT INTO `biocom`.`HealthData` (`DataId`, `PatientId`, `DataType`, `DataValue`, `DataTimeStamp`) VALUES (NULL, '".$PatientId."', '".$DataType."', '".$DataValue."', CURRENT\_TIMESTAMP);");

$this->db->commit();

}

Database Appendix.2

////########PULLS DATA FROM SERVER, DISPLAYS AS TABLE ON WEBPAGE AND RESPONDS WITH DATA

elseif(isset($\_POST["PullDataRequest"])&&isset($\_POST["PatientIdPULL"])&&isset($\_POST["DataType"]))

{

$IdHold = $\_GET["PatientIdPULL"];

if($\_GET["PullDataRequest"] == "Pull\_Health")

{

$dataType = $\_GET["DataType"];

$sql = 'SELECT Datatype, DataValue, DataTimeStamp FROM `biocom`.`HealthData` WHERE `PatientId` = '.$IdHold.' AND Datatype=\''.$dataType.'\' ORDER BY `DataId` ASC';

$stmt = $this->db->prepare($sql);

$stmt->execute();

$stmt->bind\_result( $Datatype, $DataValue, $DataTimeStamp);

echo "<table style=\"width:400px\">";

echo"<tr><th>Data Type</th><th>".$dataType."</th><th>Time</th></tr>";

while ($stmt->fetch())

{

$result = $result." {[".$Datatype."] [".number\_format($DataValue,2)."] [".$DataTimeStamp."]}";

echo "<tr>";

echo "<td>".$Datatype."</td>";

echo "<td>".number\_format($DataValue,2)."</td>";

echo "<td>".$DataTimeStamp."</td>";

echo "</tr>";

}

echo "</table>";

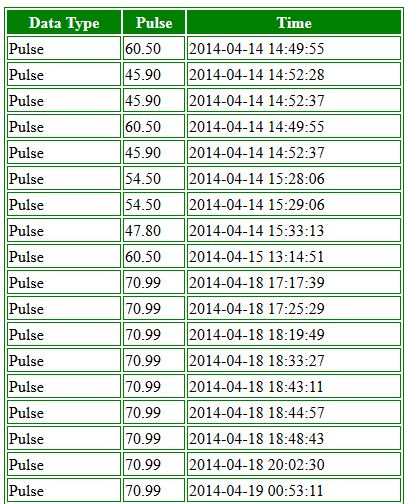
$stmt->close();

$this->sendResponse(200, json\_encode($result));

}

}

Database Appendix.3



# Appendix E – Temperature Sensor

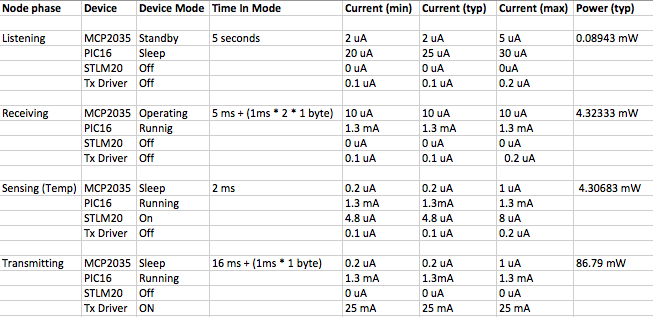
**STLM20 Emerison Bath Data:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Emersion Bath using STLM20 @ 2.7V** | | | | |
|
| Fluke 54II Temp | Voltage Out | Output Temperature | Difference from Fluke and STLM20 | % Difference |
|
| 32.0 | 1.4876 | 32.0 | 0.009 | 0.027 |
| 32.1 | 1.4860 | 32.1 | 0.028 | 0.086 |
| 32.2 | 1.4845 | 32.3 | 0.055 | 0.171 |
| 32.3 | 1.4826 | 32.4 | 0.117 | 0.362 |
| 32.4 | 1.4813 | 32.5 | 0.117 | 0.363 |
| 32.5 | 1.4805 | 32.6 | 0.096 | 0.294 |
| 32.6 | 1.4794 | 32.7 | 0.089 | 0.273 |
| 32.7 | 1.4785 | 32.8 | 0.066 | 0.201 |
| 32.8 | 1.4776 | 32.8 | 0.042 | 0.129 |
| 32.9 | 1.4765 | 32.9 | 0.036 | 0.109 |
| 33.0 | 1.4756 | 33.0 | 0.012 | 0.037 |
| 33.1 | 1.4744 | 33.1 | 0.014 | 0.043 |
| 33.2 | 1.4733 | 33.2 | 0.008 | 0.024 |
| 33.3 | 1.4727 | 33.3 | 0.041 | 0.124 |
| 33.4 | 1.4712 | 33.4 | 0.014 | 0.041 |
| 33.5 | 1.4707 | 33.4 | 0.071 | 0.212 |
| 33.6 | 1.4690 | 33.6 | 0.027 | 0.079 |
| 33.7 | 1.4683 | 33.6 | 0.067 | 0.199 |
| 33.8 | 1.4675 | 33.7 | 0.099 | 0.293 |
| 33.9 | 1.4660 | 33.8 | 0.072 | 0.211 |
| 34.0 | 1.4648 | 33.9 | 0.070 | 0.205 |
| 34.1 | 1.4638 | 34.0 | 0.085 | 0.248 |
| 34.2 | 1.4624 | 34.1 | 0.066 | 0.192 |
| 34.3 | 1.4612 | 34.2 | 0.064 | 0.186 |
| 34.4 | 1.4600 | 34.3 | 0.062 | 0.179 |
| 34.5 | 1.4585 | 34.5 | 0.034 | 0.099 |
| 34.6 | 1.4573 | 34.6 | 0.032 | 0.093 |
| 34.7 | 1.4563 | 34.7 | 0.047 | 0.136 |
| 34.8 | 1.4554 | 34.7 | 0.071 | 0.204 |
| 34.9 | 1.4544 | 34.8 | 0.086 | 0.246 |
| 35.0 | 1.4534 | 34.9 | 0.101 | 0.289 |
| 35.1 | 1.4524 | 35.0 | 0.116 | 0.331 |
| 35.2 | 1.4514 | 35.1 | 0.131 | 0.373 |
| 35.3 | 1.4501 | 35.2 | 0.121 | 0.342 |
| 35.4 | 1.4488 | 35.3 | 0.110 | 0.312 |
| 35.5 | 1.4474 | 35.4 | 0.091 | 0.258 |
| 35.6 | 1.4458 | 35.5 | 0.056 | 0.156 |
| 35.7 | 1.4444 | 35.7 | 0.037 | 0.103 |
| 35.8 | 1.4434 | 35.7 | 0.052 | 0.145 |
| 35.9 | 1.4421 | 35.9 | 0.042 | 0.116 |
| 36.0 | 1.4409 | 36.0 | 0.040 | 0.110 |
| 36.1 | 1.4395 | 36.1 | 0.021 | 0.058 |
| 36.2 | 1.4386 | 36.2 | 0.044 | 0.123 |
| 36.3 | 1.4374 | 36.3 | 0.043 | 0.118 |
| 36.4 | 1.4360 | 36.4 | 0.024 | 0.066 |
| 36.5 | 1.4351 | 36.5 | 0.048 | 0.130 |
| 36.6 | 1.4337 | 36.6 | 0.029 | 0.078 |
| 36.7 | 1.4327 | 36.7 | 0.044 | 0.120 |
| 36.8 | 1.4316 | 36.7 | 0.051 | 0.137 |
| 36.9 | 1.4300 | 36.9 | 0.015 | 0.040 |
| 37.0 | 1.4289 | 37.0 | 0.022 | 0.058 |
| 37.1 | 1.4275 | 37.1 | 0.003 | 0.008 |
| 37.2 | 1.4265 | 37.2 | 0.018 | 0.048 |
| 37.3 | 1.4251 | 37.3 | 0.001 | 0.002 |
| 37.4 | 1.4235 | 37.4 | 0.036 | 0.097 |
| 37.5 | 1.4222 | 37.5 | 0.047 | 0.124 |
| 37.6 | 1.4214 | 37.6 | 0.014 | 0.038 |
| 37.7 | 1.4203 | 37.7 | 0.008 | 0.021 |
| 37.8 | 1.4193 | 37.8 | 0.007 | 0.020 |
| 37.9 | 1.4183 | 37.9 | 0.023 | 0.060 |
| 38.0 | 1.4170 | 38.0 | 0.013 | 0.033 |
| 38.1 | 1.4155 | 38.1 | 0.015 | 0.038 |
| 38.2 | 1.4144 | 38.2 | 0.008 | 0.021 |
| 38.3 | 1.4132 | 38.3 | 0.010 | 0.025 |
| 38.4 | 1.4119 | 38.4 | 0.020 | 0.051 |
| 38.5 | 1.4105 | 38.5 | 0.038 | 0.100 |
| 38.6 | 1.4094 | 38.6 | 0.032 | 0.082 |
| 38.7 | 1.4082 | 38.7 | 0.033 | 0.086 |
| 38.8 | 1.4073 | 38.8 | 0.009 | 0.024 |
| 38.9 | 1.4062 | 38.9 | 0.003 | 0.007 |
| 39.0 | 1.4048 | 39.0 | 0.021 | 0.054 |
|  |  |  |  |  |
|  |  | average | 0.046 | 0.133 |

# Appendix F – Node PCBs

# Appendix G – Power Supply Data

**Sensor Node Power Data**



**Aggregator Power Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Device Mode** | **uA** | **Volts** | **3V(uA)** | **1.8V(uA)** | **State Power Total (3V) (mW)** | **State Power Total (1.8V) (mW)** |
| On | 2.5 | 9 |  |  | 60.0295 |  |
| On | 2 | 5 |  |  |  |  |
| On |  |  | 20000 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Sleep |  |  | 0.2 |  | 0.01475 | 0.01376 |
| Sleep |  |  | 0.05 | 0.02 |  |  |
| Off | 2.02 | 5 |  |  |  |  |
| On |  |  | 0.8 | 0.5 |  |  |
| Off |  |  | 0 | 0 |  |  |
| Off |  |  | 0 |  |  |  |
| Off | 0 | 0.3 |  |  |  |  |
| Off | 0 | 5 |  |  |  |  |
| Off | 0 | 5 |  |  |  |  |
| Off | 0 | 1 |  |  |  |  |
| Off |  |  | 0.5 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Standby |  |  | 2 |  | 6 | 6 |
| On(External 8.192MHz) |  |  | 336 | 192 |  |  |
| On | 50000 | 0.1 |  |  |  |  |
| On |  |  | 0.8 | 0.5 |  |  |
| Off |  |  | 0 | 0 |  |  |
| Off |  |  | 0 |  |  |  |
| Off | 0 | 0.3 |  |  |  |  |
| Off | 0 | 5 |  |  |  |  |
| Off | 0 | 1 |  |  |  |  |
| Off |  |  | 0.5 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Standby |  |  | 2 |  | 9011 | 9011 |
| On (8 Mhz) |  |  | 336 | 192 |  |  |
| Off | 2.02 | 5 |  |  |  |  |
| On |  |  | 0.8 | 0.5 |  |  |
| Off |  |  | 0 | 0 |  |  |
| On | 2400 | 3 | 2400 |  |  |  |
| On | 30000430 | 0.3 |  |  |  |  |
| On | 460 | 5 |  |  |  |  |
| On | 500 | 1 |  |  |  |  |
| Off |  |  | 0.5 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| On |  |  | 10 |  | 9011.181 | 9010.7481 |
| On (8 Mhz) | 336 | 3 | 336 | 192 |  |  |
| Off | 2.02 | 5 |  |  |  |  |
| On | 0.5 | 1.8 | 0.8 | 0.5 |  |  |
| Off |  |  | 0 | 0 |  |  |
| On |  |  | 2400 |  |  |  |
| On | 30000430 | 0.3 |  |  |  |  |
| On | 460 | 5 |  |  |  |  |
| On | 500 | 1 |  |  |  |  |
| Off |  |  | 0.5 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Sleep |  |  | 0.5 |  | 1 | 0.5906 |
| On |  |  | 336 | 192 |  |  |
| Off | 2.02 | 5 |  |  |  |  |
| On |  |  | 0.8 | 0.5 |  |  |
| Off |  |  | 0 | 0 |  |  |
| Off |  |  | 0 |  |  |  |
| Off | 0 | 0.3 |  |  |  |  |
| Off | 0 | 5 |  |  |  |  |
| Off | 0 | 1 |  |  |  |  |
| Off |  |  | 0.5 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Sleep |  |  | 0.5 |  | 25 | 25 |
| On (8 Mhz) |  |  | 336 | 192 |  |  |
| Off | 2.02 | 5 |  |  |  |  |
| On |  |  | 0.8 | 0.5 |  |  |
| Off |  |  | 0 | 0 |  |  |
| Off |  |  | 0 |  |  |  |
| Off | 0 | 0.3 |  |  |  |  |
| Off | 0 | 5 |  |  |  |  |
| Off | 0 | 1 |  |  |  |  |
| Off |  |  | 8100 |  |  |  |

# Appendix H – Aggregator PCBs

Figure -Prototype Aggregator Schematic

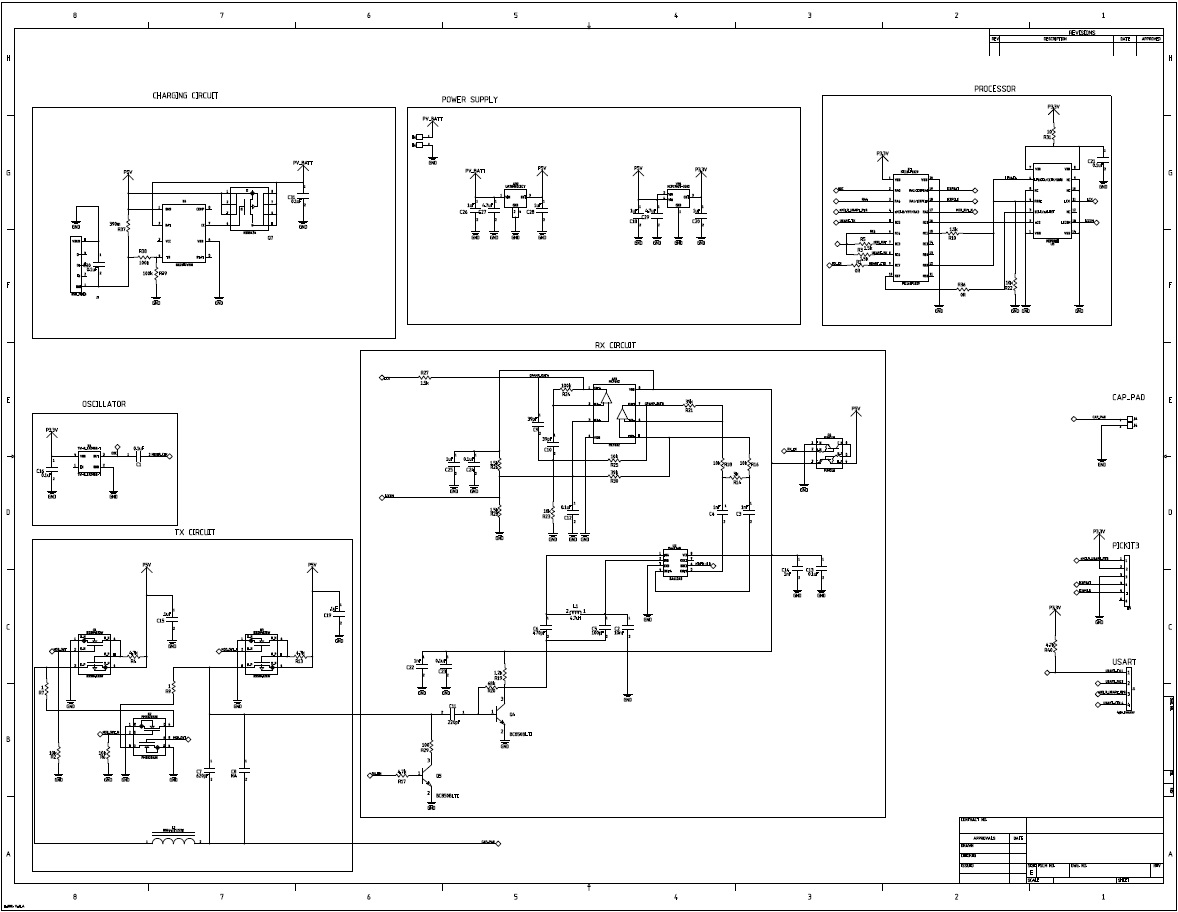


Figure 3-Final Aggregator Schematic

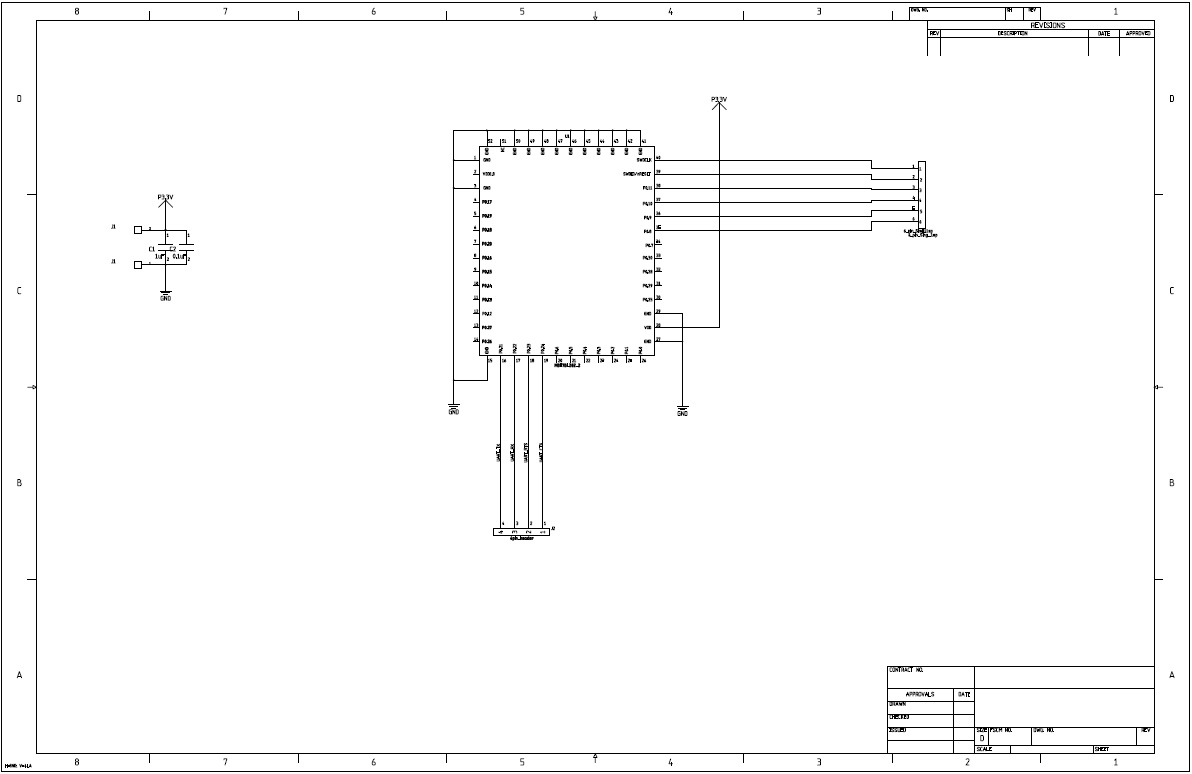
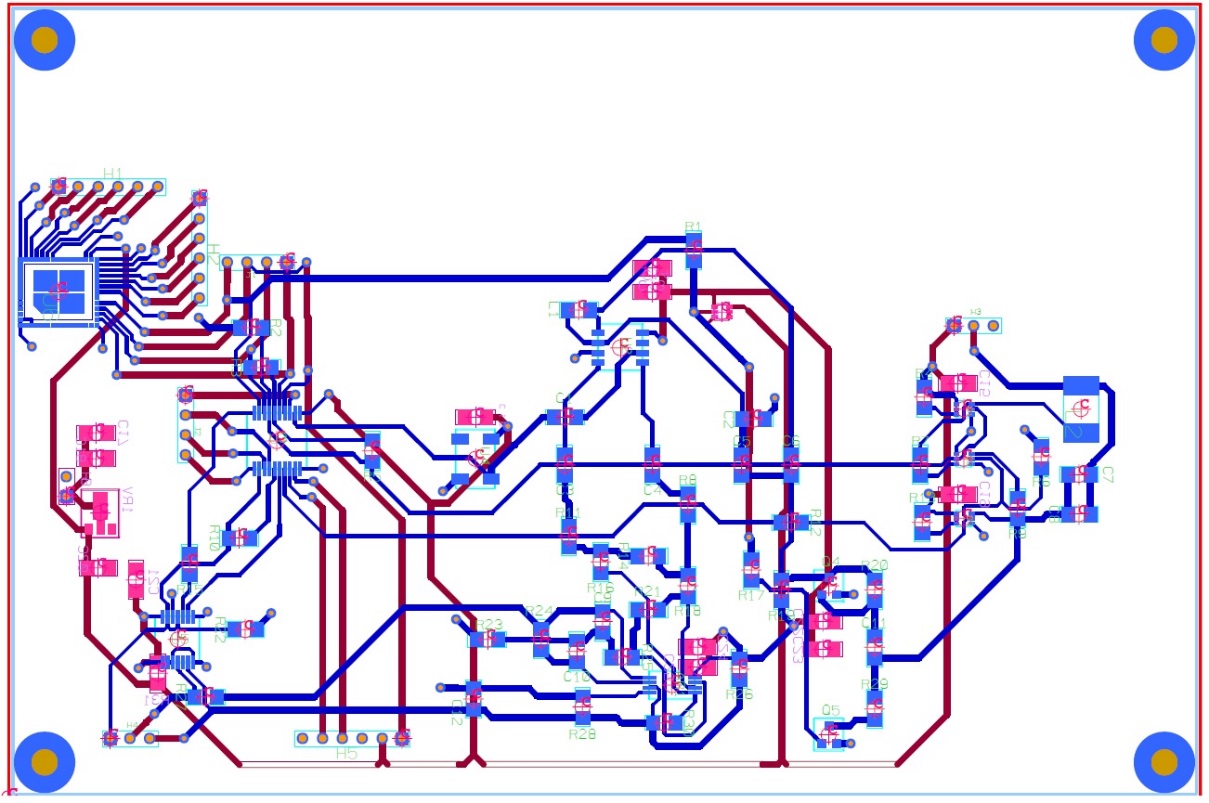


Figure 4-Bluetooth Low Energy Schematic



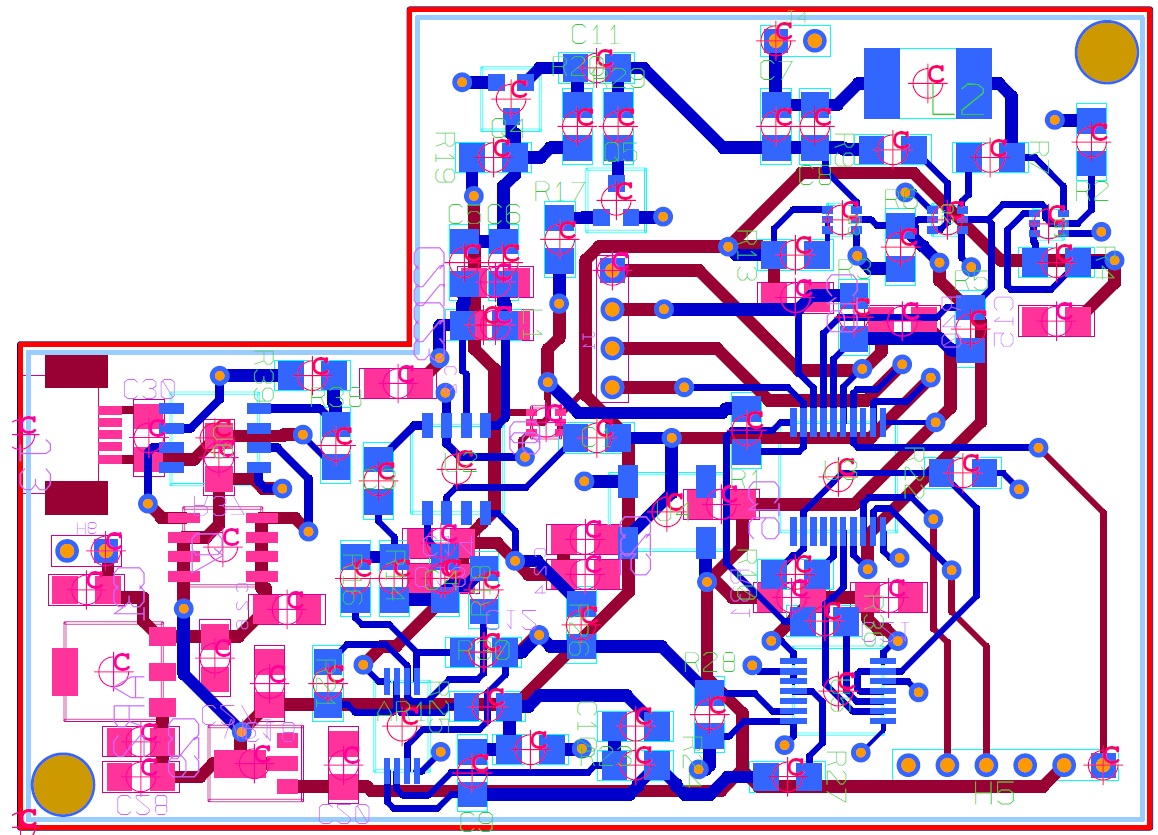
Figure 5-Prototype Aggregator PCB(not to scale

Figure 6-Final Aggregator PCB(not to Scale)

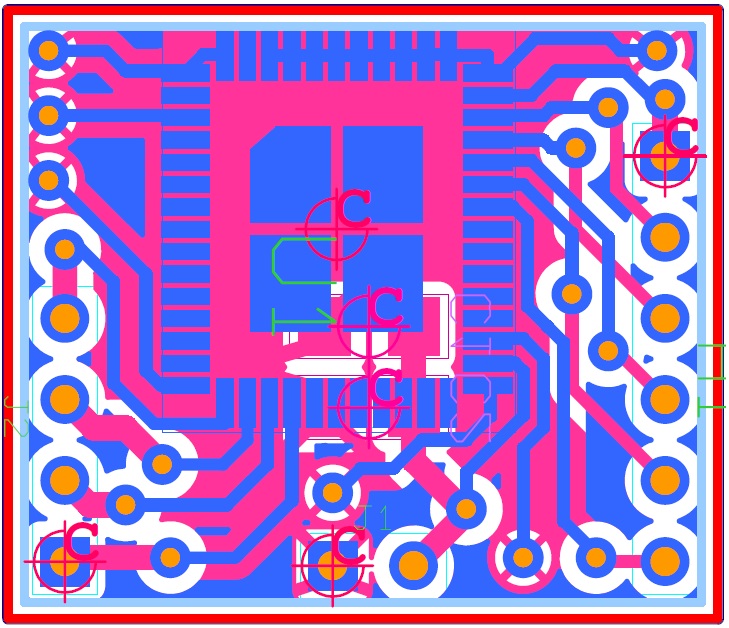


Figure 7-BLE PCB(not to scale)

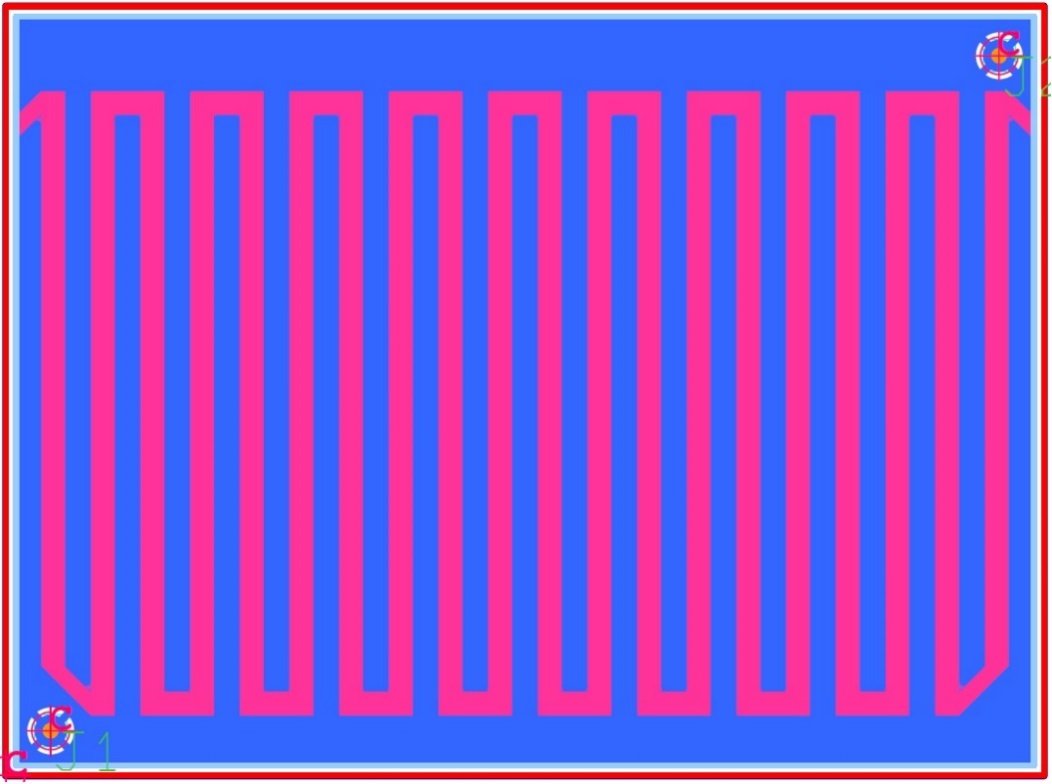


Figure 8-TouchPad PCB(not to scale)