

Little Vampire Glucometer Documentation

Electrical Engineers:

Michael Williams

Matthew Henne

Christopher Homa

Mechanical Engineer:

Ishmael Amegashie

Software Engineer:

Sofyan Saputra

Table of Contents

3. Introduction	4
3.1 Problem Statement and Proposed Solution.....	4
3.2 High Level Description of Solution	5
3.3 Meeting Expectations	7
4. Detailed System Requirements	7
4.1 Overview of Final Glucometer System Requirements	7
4.1.1 General Features.....	7
4.1.2 Hardware Design.....	8
4.2 Overview of LCD System Requirements	8
4.3 Overview of Bluetooth System Requirements	8
4.4 Overview of EEPROM System Requirements.....	8
4.5 Overview of Software System Requirements	8
4.6 Overview of Mechanical System Requirements	10
5. Detailed Project Description.....	10
5.1 System Theory of Operation.....	10
5.2 System Block Diagram.....	14
5.3 Design and Operation of Glucometer Subsystem	14
5.3.1 Reading the Test Strip	14
5.3.2 Temperature Sensing.....	17
5.3.3 Determining the Regression Formula.....	18
5.3.4 Timestamp.....	23
5.4 Design and Operation of LCD Subsystem	24
5.4.1 Subsystem Overview.....	24
5.4.2 Subsystem Design	24
5.5 Design and Operation of BLE Subsystem	27
5.6 Design and Operation of EEPROM Subsystem	28
5.7 Design and Operation of Software Subsystem.....	32
5.7.1 Database and Skeleton Interface.....	32
5.7.2 Bluetooth Integration.....	36
5.7.3 Data Analytics and App Visuals	38
5.8 Design and Operation of Mechanical Subsystem.....	41
5.8.1 Main Design.....	41
5.8.2 Lancing Device	41
5.8.3 Test Strip Cartridge	43
5.9 Design and Operation of the Lancing Device.....	43
5.9.1 Requirements:.....	43
5.9.2 Approach:	43
5.9.3 Challenges:	44
5.10 Power	49
6. System Integration Testing.....	50
6.1. Subsystem Integration Testing	50
6.1.1 Hardware Subsystems.....	50

6.1.2 Software Subsystems.....	51
6.2. Design Requirement Demonstration.....	52
7. User Manual.....	53
7.1 Hardware.....	53
7.2 Lancing Device.....	54
7.3 Software	55
8. To-Market Design Changes.....	57
8.1 Software	57
8.2 Hardware.....	58
9. Conclusions	59
10. Appendices	60
Appendix A. Hardware Schematics and Board Layouts	60
A1. Top Board	60
A2. Bottom Board	62
Appendix B. Microcontroller Software Reference	64
Appendix C. App Software Reference	64
C1. Programming Languages	64
C2. Integrated Development Environments (IDEs).....	65
C3. Miscellaneous Software	65
Appendix D. Electrical Parts Data Sheets	65
D1. Glucometer & General Parts.....	65
D2. Bluetooth Low Energy	66
Appendix E. Mechanical Parts Data Sheets	67

3. Introduction

The world is becoming more connected everyday. People are using technology to monitor everything from their activity to their diets. We want to design a marketable diabetes blood glucose monitor that connects to any smartphone through Bluetooth and also combines three products (glucometer, lancing device, and test strip cartridge) into a single compact device. This will bring blood glucose monitoring to the 21st century and makes the lives of diabetics easier.

3.1 Problem Statement and Proposed Solution

Diabetics need to monitor their blood sugar on a regular basis and deliver insulin or glucose to their body since it does not do this naturally for them. Controlling diabetes is hard and requires constant attention. Part of this difficulty stems from the fact that it can be tough to remember the last time you took a blood sugar or gave insulin. If you are a newly diagnosed diabetic, this is extremely difficult, especially for young children. However, remembering is the least of a diabetics worry. They have to remember to bring their blood glucose monitoring kit, which includes a glucose monitor, test strips, a lancing device, alcohol cleaning pads, and something to record their blood sugar. All which are easy to forget and may require maintenance or replacement, not to mention they all must be carried and are bulky. The day in the life of a diabetic is

consumed with managing their disease. We want to help free up some of the worry with our blood glucose monitoring system, which will be compact, connected, easy to use, and hard to forget.

Our solution focuses on providing an all-in-one device to measure and manage blood glucose levels. Our device will include a lancing device, a glucose meter, and a test strip cartridge, all of which will attach to existing insulin pens. The result is one single device that diabetics will carry with them, as opposed to the current solution where they are responsible for three separate devices – lancing device, glucometer, and pen. This device will also enable users to upload their readings to a mobile app. This allows for easy tracking of glucose levels over time and also provides a platform for sharing reading data with doctors or, for children, parents.

3.2 High Level Description of Solution

Our solution contains two main components – hardware and software. The hardware component consists of a watch-like device that contains the glucometer, lancing device, and test-strip holder. There are five main subsystems for the hardware component. These are listed below, along with a brief description for each:

- Glucometer
 - The glucometer primarily consists of an ADC that samples voltage levels over a short time period and an associated regression algorithm to calculate glucose levels. A temperature sensor is also used because the regression algorithm is temperature-dependent.
- User Interface
 - This device has a user interface of 2 buttons and an LCD screen, and the buttons are used to guide the user through the process of measuring glucose levels and then connecting and transferring to a smartphone.
- Bluetooth Low Energy

- An nRF8001 is used to provide Bluetooth Low Energy functionality to the device, enabling saved readings to be sent to a smartphone and submitted to a database for each user.
- EEPROM
 - An EEPROM is used to store data on the device before transferring it to a smartphone. The main pieces of data stored in the EEPROM are the current time, which can be displayed on the LCD, glucose readings, and the timestamps of readings.
- Casing and Mechanical Components
 - The mechanical aspect of this project is important because size is a main consideration. The lancing device, therefore, was planned to be as small as possible while the size of the casing is more dependent on the size of the boards that must fit inside of it. In the end, a lack of precision in the 3D printers prevented a lancing device of the desired size from being made, but the current design is capable of being scaled down. A wearable watch-like design for the casing was decided on in order to more easily fit the size requirements of our boards and to be more easily transported and kept track of by the user.

Similarly, there were three main subsystems for the application software component, listed and described below.

- Database and Skeleton Interface
 - An online database was created to store glucose readings and enable user access via mobile application. Each user has her own database, and a login system is implemented to password-protect database access.
- Bluetooth Integration
 - Bluetooth Low Energy APIs are used to send and receive data between the device and smartphone. The smartphone receives glucose readings and their associated timestamps and sends the current timestamp to the device in order to update the clock and maintain clock precision.
- Data Analytics and App Visuals

- Data analytics and visualization is implemented in order to improve user comprehension of his or her glucose reading data. The user can adjust the timescale of data that she is viewing, and also filter it based on activity performed prior to reading. Summary statistics are displayed in order to provide a quick summary of the data for the selected time range.

3.3 Meeting Expectations

While there is certainly room for improvement, which will be discussed in the 'To-Market Design Changes' section, we were successful in building an initial prototype for the device. All functional requirements were met, and the glucometer performed well in measuring glucose levels during tests with a solution of a known glucose level. One area where initial design goals were not met was the size factor, but for an initial prototype we believe we are in a good position to improve on this.

4. Detailed System Requirements

4.1 Overview of Final Glucometer System Requirements

4.1.1 General Features

- Glucose measurement range: 20 mg/dl to 600 mg/dl (1 mmol/l to 33 mmol/l)
 - Measure current from testing strip
 - Convert current to voltage
 - Amplify voltage
 - Analog to Digital Conversion interface on the microcontroller
 - Test result is displayed within five seconds
 - Convert digital voltage to associated glucose level
- Automatic storage of last 32 glucose readings with date and time stamp on internal EEPROM
- No test strip coding: Generic regression equation will be implemented and can be modified based on the test strip characteristics

4.1.2 Hardware Design

- Number of boards: Double Board
- PIC16LF1782-I/SS Device: 28-pin device
- Test strip connection: Terminals/connector provided
- RTCC (using internal Timer1): Date and timestamp for the glucose meter
- Internal EEPROM:
 - Record last 32 readings of the glucose meter
 - Store any parameters or calibration data related to the test strip
- Test strip sensing: To detect insertion of the test strip
- Temperature sensing: Provision given to consider temperature variation for the glucose calculations. If relation between the temperature and the glucose concentration is known, it can be incorporated in the regression equation to take care of any changes in the glucose concentration due to temperature variations

4.2 Overview of LCD System Requirements

- Display glucose level from glucometer
- Display user prompts to cycle through glucose reading procedure
- Serial connection to microcontroller

4.3 Overview of Bluetooth System Requirements

- Establish connection between smartphone and device
- Transfer glucose reading and associated timestamp from device to smartphone
- Receive timestamp from smartphone in order to update device clock

4.4 Overview of EEPROM System Requirements

- Store current timestamp
- Store each glucose reading and associated timestamp
- SPI connection to microcontroller

4.5 Overview of Software System Requirements

- Online database to store each user's glucometer readings

- Storage is located on Parse.com servers through their cloud platform. Parse specializes in data storage for mobile and web applications.
- Reading Bluetooth data into a mobile or computer device
 - Utilize Bluetooth APIs in order to acquire data from the Bluetooth device, which will send glucose readings from the glucometer into a mobile or standard computer platform.
 - Create an application which will be able to initiate a Bluetooth connection between the chip, receive data from the chip, and process the data
- Automatically storing glucose readings from Bluetooth into the database
 - Being able to automatically route all readings from the glucometer to the device mobile application and then into online database.
 - Readings will come with a timestamp and identify the user who is storing the current reading. It will also allow the user to store the activity they completed prior to and after the reading.
 - Upon data upload, the user will be notified. If for whatever reason (data corruption, lack of connection, and so on) the data cannot be uploaded into the Parse database, the user will also be notified
 - If no internet connection is available to upload the data, then the values will be stored in a buffer on the phone. All the values in be uploaded in bulk when internet access is available and the upload button is pressed. The upload button will clear the data buffer once it is uploaded.
- Web interface for users to interact with their data
 - A web interface conveniently available from both mobile and standard computer platforms for which users can view and analyze their glucometer readings
- Login Interface to identify unique users
 - To get into their individual web dashboard, each user will need to login into their service, so the database can identify what data to pull for the user
 - Login will be completed through a trusted and secure third-party service

- Parse's automated simple login service
- All passwords in Parse's users table is properly hashed and hidden
- Analysis features on the web interface
 - Line chart detailing glucose readings over time intervals
 - Ability to filter the chart based on day, week, month, and so on
 - Daily, weekly, and monthly average of readings
 - Warning indicators for abnormal glucose readings in the form of upper and lower limits of glucose readings as well as percent of readings within acceptable range.

4.6 Overview of Mechanical System Requirements

5. Detailed Project Description

5.1 System Theory of Operation

The Glucometer Smartwatch functions on many different levels: the digital part of the watch, the mechanical part of the watch, and the Smartphone app.

The mechanical design was made to supply the diabetic with a compact design to carry all the long-lasting parts of their testing system, which includes the lancing and glucometer (see note). As will be explained in section 5.8, the design of the mechanical system integrated these two parts by a simple attachment to the Smartwatch on one side of the case. When the diabetic needs access to the lancing, they simply remove it and use it to prick their finger for their test.

NOTE: For detailed instructions on how to use the device, refer to section 7.

Furthermore, for future considerations involving the theory of operation, refer to section 8.

The digital part of the Smartwatch includes a glucometer, a LCD, a Bluetooth Low Energy (BLE) chip, two buttons, and EEPROM memory. The following finite state machine flowcharts were created to integrate all the various electrical parts together. As shown, the LCD screen is used to display the commands necessary for the diabetic to take their blood sugar as well as displays the results. The BLE chip is used to integrate the blood sugar reading from the result of the glucometer circuit and send it to the Smartphone app. The Smartphone app then is used to display the reading as well as allow the user to log more information regarding the reading just taken, such as before exercise or after eating. The app can also be used to show various graphs so the diabetic can visualize their blood sugar readings (more information provided in section 5.7).

FINITE STATE MACHINE - DISPLAY PREVIOUS READINGS

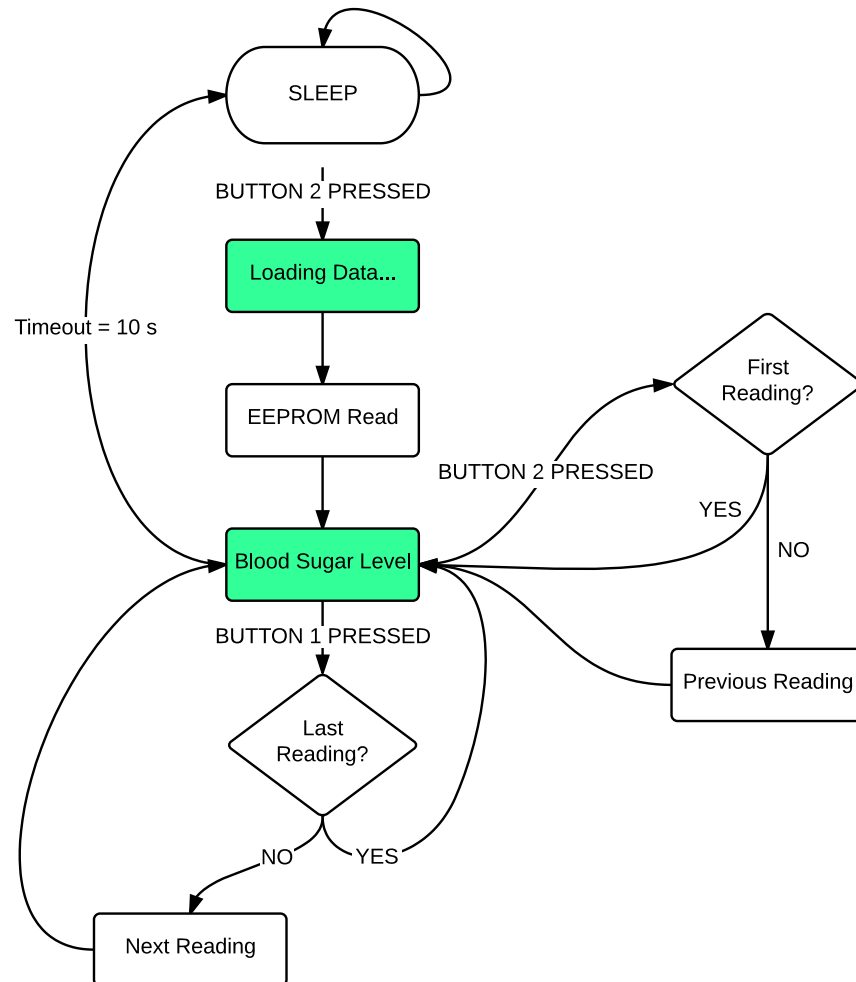
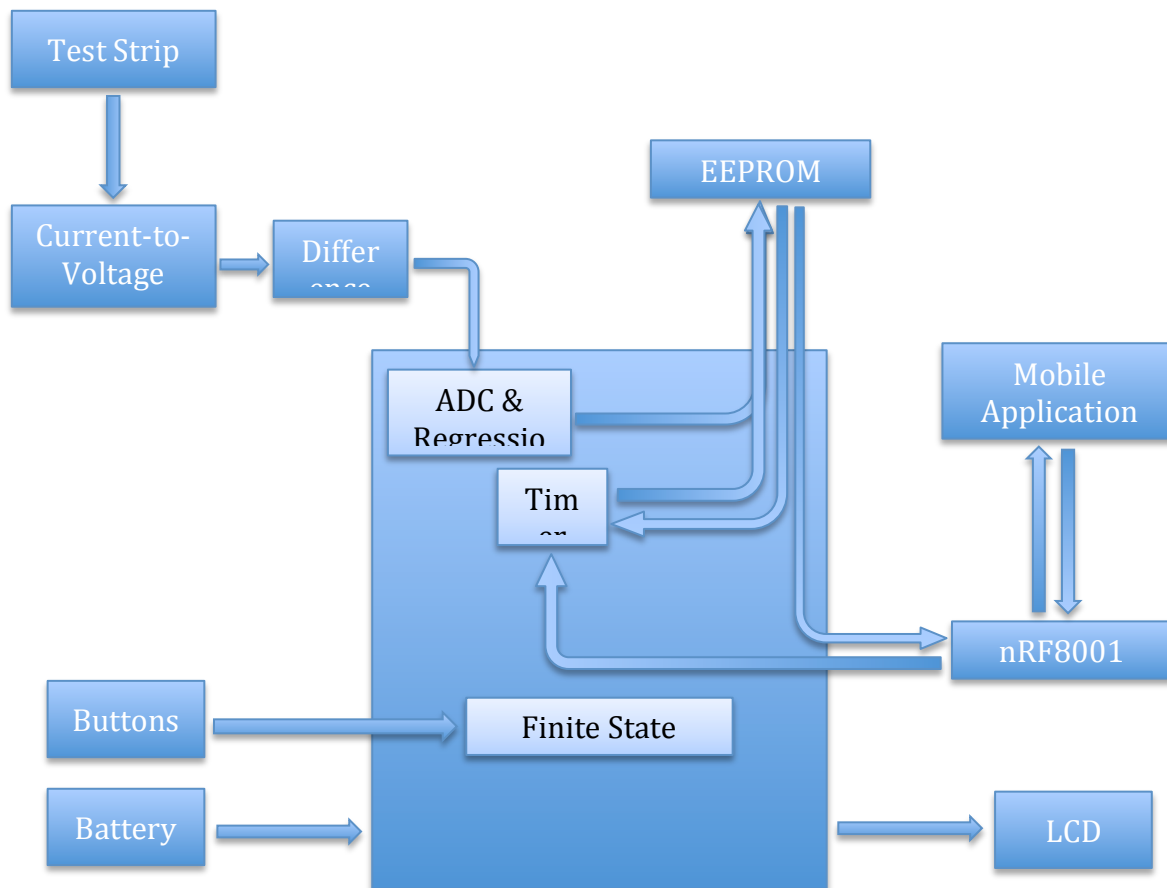


Figure 5.1.1. Display Previous Readings FSM

5.2 System Block Diagram



5.3 Design and Operation of Glucometer Subsystem

5.3.1 Reading the Test Strip

There are many different types of test strips available on the market. However, our requirement is for the glucometer to be low cost. Therefore, we have decided to proceed with Unistrip Generic Test Strips, which are some of the cheapest test strips available on the market to date. There are three electrodes associated with this test strip: working, counter, and reference. The generic general design of the test strip and the associated electrical circuit is shown below:

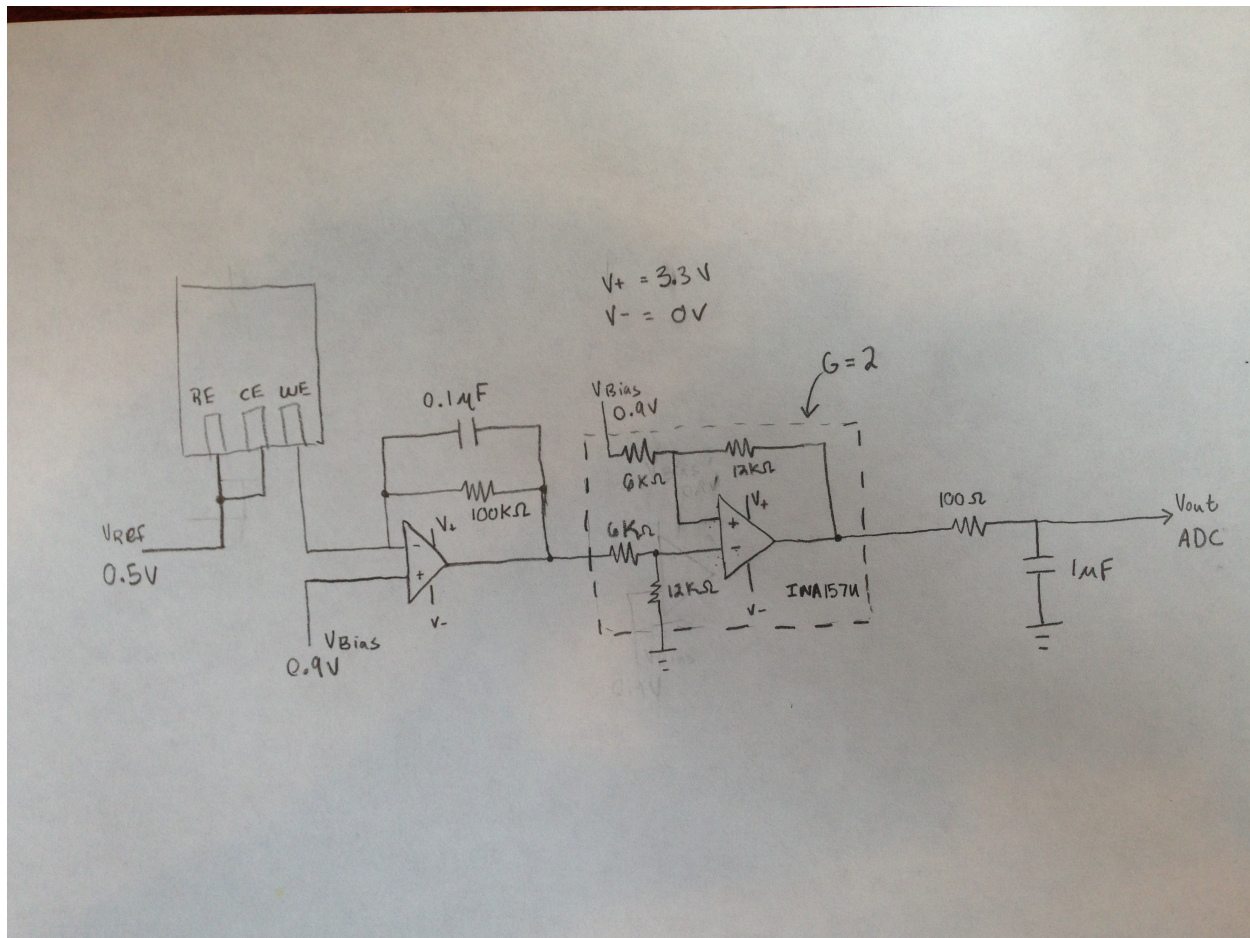


Figure 5.3.1.1. Glucometer Amplification Circuit

The electrodes can be described as:

- **Working electrode:** Electrons are produced here during the chemical reaction. This electrode is connected to the current-to-voltage amplifier.
- **Reference electrode:** Held at a constant voltage with respect to the working electrode to push the desired chemical reactions.
- **Counter electrode:** Supplies current to the working electrode.

The Reference Electrode is set at a specific reference voltage (V_{Ref}) while the Working Electrode is set at a specific bias voltage (V_{Bias}). This way there is a precise voltage drop across the working electrode and the reference electrode. The voltage drop between the Reference and Working electrodes drives the test strip's output current.

The magnitude of the output current correlates to the number of electrons produced by

the chemical reaction of oxidizing the blood. The flow of electrons will correspond to the flow of current through the working and the reference electrode and will change based on the glucose concentration in the blood. As shown in Figure 5.3.1.1, the current output from the test strip is connected to a transimpedance amplifier, which acts as a current-to-voltage converter. The output of this operational amplifier will be then connected to a difference amplifier circuit. The output of this operation amplifier measures the voltage above VBias as well as amplifies the difference by a gain of two. This is important as it allows the ADC readings to have a broader range and ultimately makes the glucometer more accurate. The final part of the circuit is a simple low pass filter to get rid of any noise associated from the test strip, electrical circuit, and operation amplifiers. This results in the final output of the circuit as a voltage level, which will be fed into the microcontroller's ADC. As will be described in part III, the regression formula will then correlate the ADC voltage level to a specific blood glucose level.



Figure 5.3.1.2. Oscilloscope Voltage Readings When Blood Sugar Was Applied to Test Strip

In order to ensure that the circuit setup works, the voltage output of the circuit was hooked up to an oscilloscope. With a test glucose solution providing a low blood sugar between 30 mg/dl to 60 mg/dl, the expected voltage outcome is high due to the fact that a low blood sugar correlates to less current being able to flow between the electrodes since less electrons will be produced. Thus, the oscilloscope output shown in Figure

5.3.1.2 above and on the left shows a high voltage difference of 2.040 V, which correlates to the lower range of expected current outputs and higher range of the associated resistance. Figure 5.3.1.2 above and on the right shows the reading after the glucose test solution was saturated with sugar. The oscilloscope voltage difference is very low, correlating well to a blood glucose level that is high. Therefore, we can conclude that the designed circuit serves its purpose well and will be used in the final design of the glucometer. Further explanation and oscilloscope printouts will be in the proceeding section on finding the regression formula.

NOTE: The time to start capturing the ADC values will be after 1.1 seconds to allow the reaction to take place and settle at its peak value. As seen in the oscilloscope readings, the reaction settles at a peak around this time for up to 5 seconds before it exponentially decays after the reaction has completed. Multiple ADC values will be taken and the max of the ADC readings will be used for the voltage level.

5.3.2 Temperature Sensing

The temperature sensor was included in the final design because it provides the ability to make the blood sugar readings more accurate over varied temperature ranges. We chose to use a low-power linear thermistor IC because it coupled low power while remaining compact, fitting well within our goal of a low power and compact glucometer. Shown below in Figure 5.3.2.1, the thermistor IC has three pins: V_{DD}, GND, and V_{OUT}.

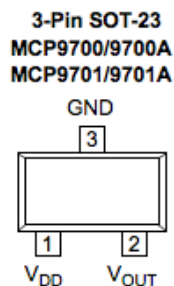


Figure 5.3.2.1. Thermistor IC Pinout Diagram

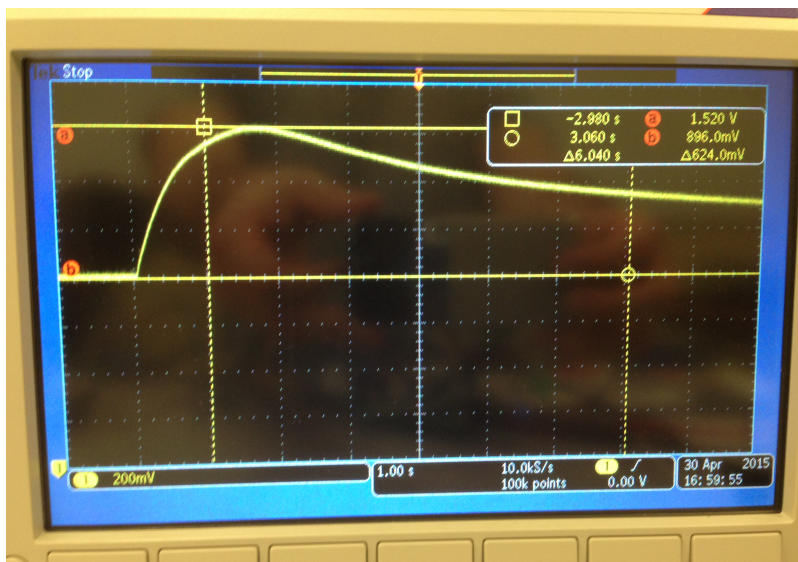
Pin 1 (Vdd) was connected to 3.3V. Pin 2 (Vout) was connected to an ADC input pin on the microcontroller. Pin 3 was connected to ground. The thermistor voltage value was read in as a voltage level and converted to a temperature in Celsius using the following formula and DC characteristics provided in the data sheet:

$$T_A = (V_{OUT} - 0.5V) / (.010 V/^{\circ}C) \quad \text{Equation 5.3.2.1}$$

A limitation of this first prototype was the inability to have the glucometer tested in a lab meant to handle accurate blood sugar tests. Therefore, the temperature sensor's data was not included in the regression formula, as the effect of temperature on test strips was not made available on the Internet. The sensor was still included in the final design since it would be needed for accuracy if this prototype was taken beyond Senior Design.

5.3.3 Determining the Regression Formula

Determining the regression formula was not an exact science because we were limited by our lack of access to a fully equipped testing laboratory. To perform the test for the regression formula we used Level 1 and Level 3 UniStrip Control Solution. A diabetic uses the control solution to calibrate the glucometer for every new cartridge of test strips they open. We used it to provide us a low blood sugar reading approximation (Level 1), a high blood sugar approximation (Level 3), and a medium blood sugar reading by combining the two solutions. The trials recorded the oscilloscope voltage responses after each solution was applied to the test strip, shown in the figures below.



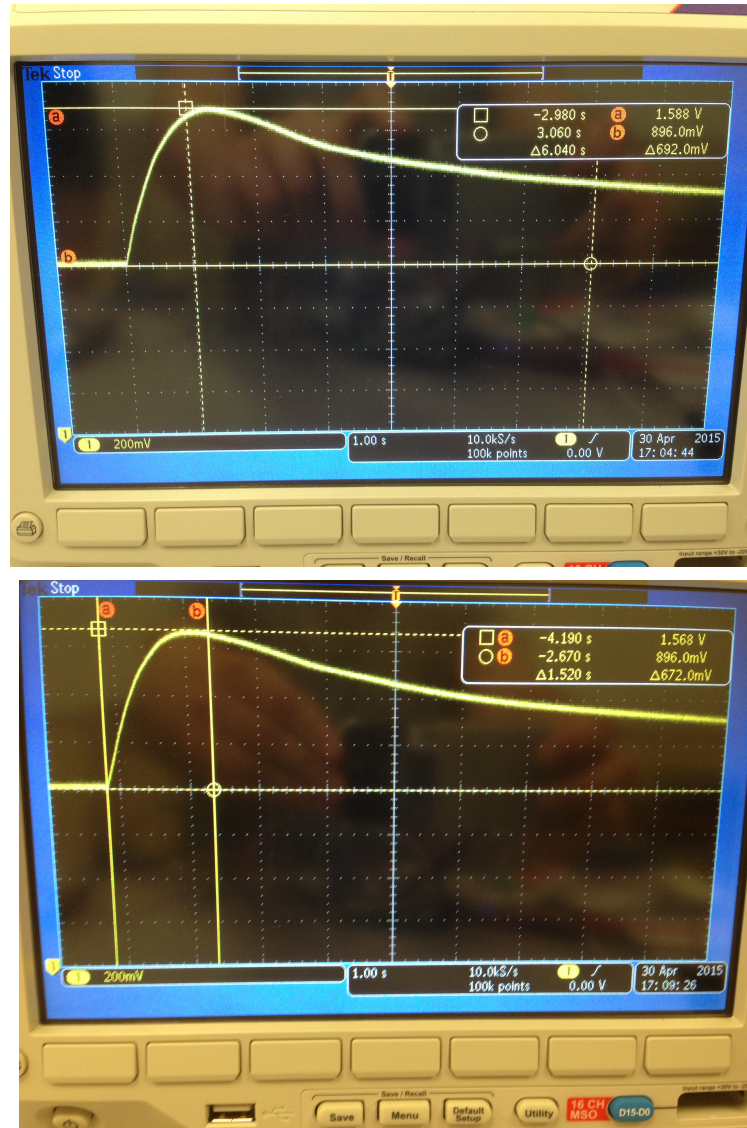


Figure 5.3.3.1 High Control Solution Oscilloscope Voltage Responses

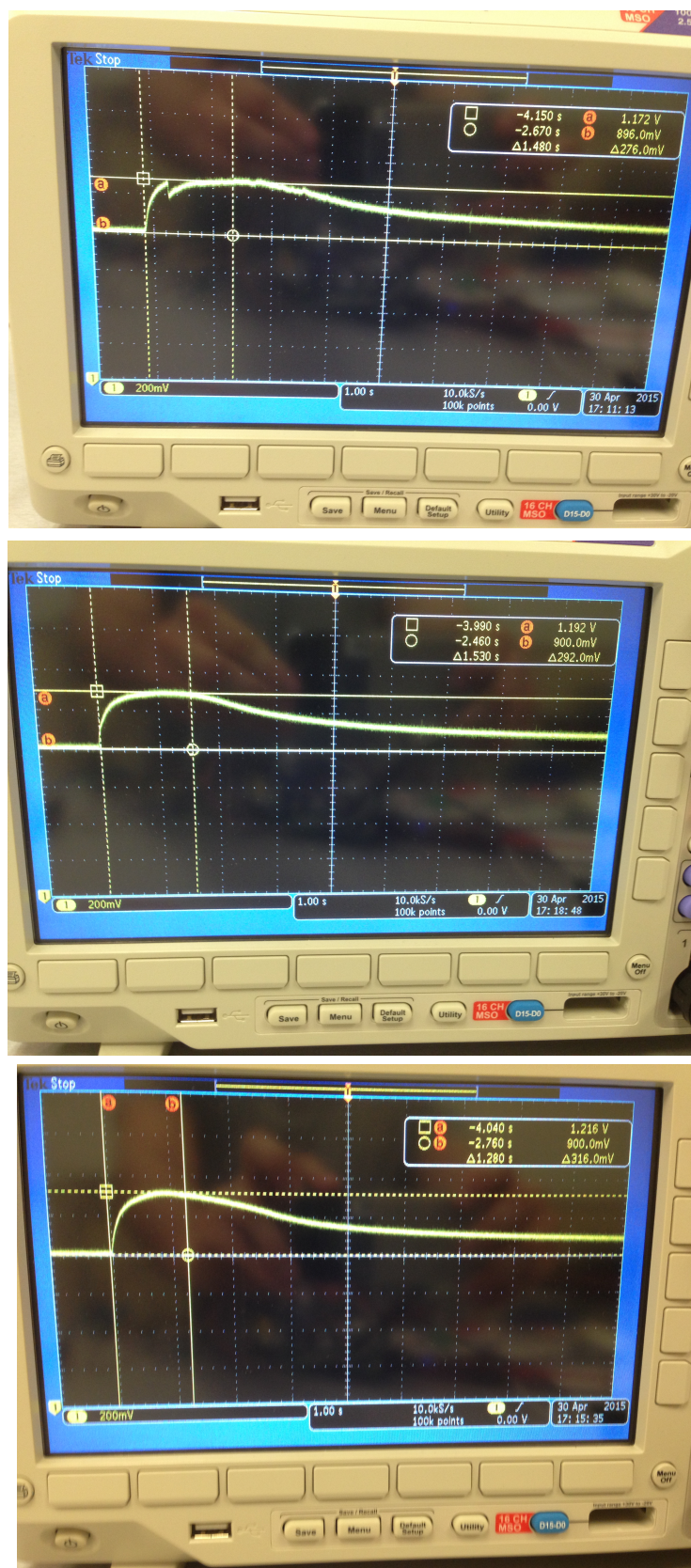
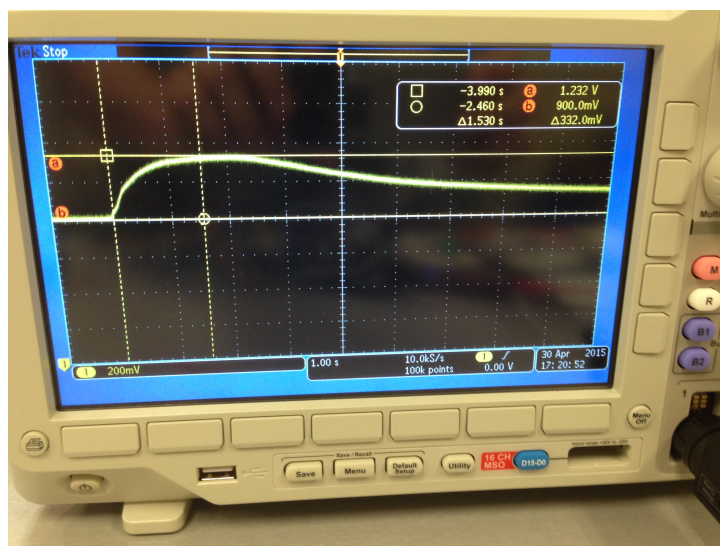


Figure 5.3.3.2. Low Control Solution Oscilloscope Voltage Responses



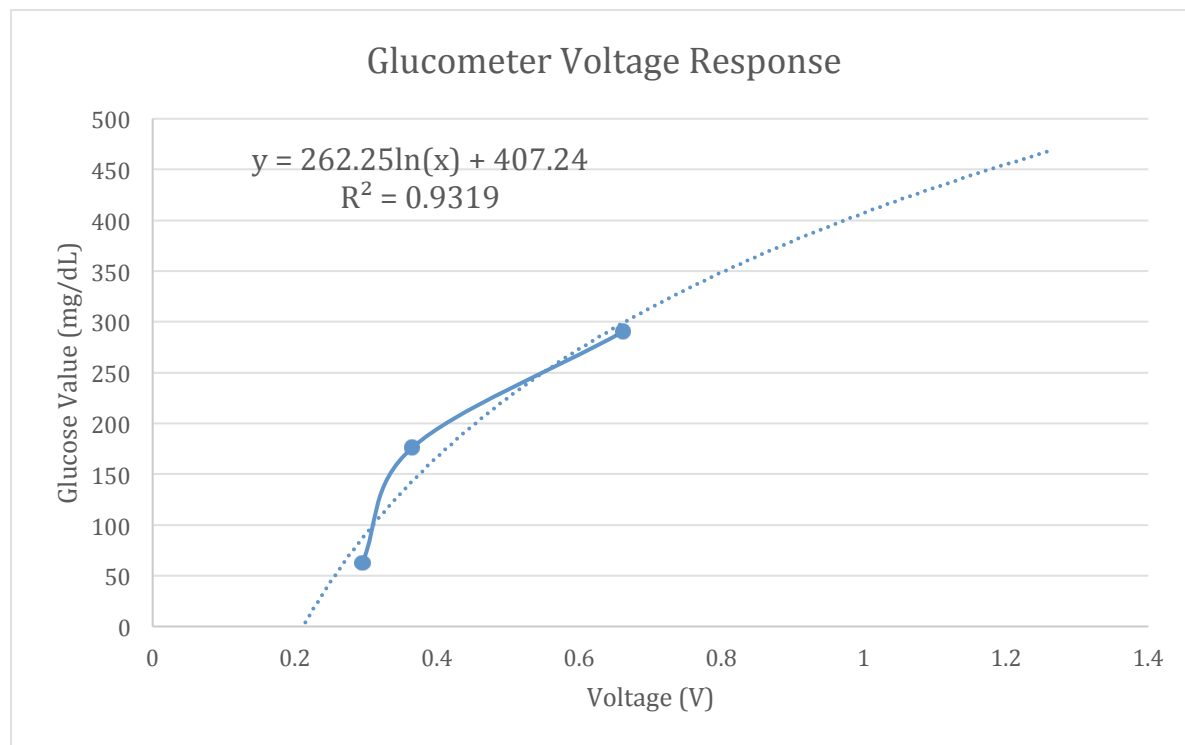
5.3.3.3. Medium Control Solution Oscilloscope Voltage Responses

The data was then recorded in Table 5.3.3.1 below.

Glucometer Voltage Response (V)			
Glucose Level	High (Level 3)	Med (Level 1 and 3)	Low (Level 1)
Trial 1	624	332	276
Trial 2	692	404	316
Trial 3	668	360	292
Average	0.661333333	0.365333333	0.294666667
Glucose Value	290 mg/dl	176.25 mg/dl	62.5 mg/dl

Table 5.3.3.1. Glucometer Voltage Response

The glucose value approximation was determined by taking the average value in the range supplied by the control solution. Level 1 was had the range 40-85 mg/dl. Level 3 had a range of 240-340. Adding one drop from each control solution and averaging the value between them determined the medium value. The following graph was created to better visualize the data collected and to determine a best-fit regression formula. The best-fit trend line was determined using a natural logarithm fit.



Therefore the following equation was used in the code to convert the ADC voltage readings to a blood sugar reading.

$$\text{Blood Glucose Level} \left(\frac{mg}{dl} \right) = 262.25 \left(\frac{mg}{dl} \right) * \ln(V_{in}) + 407.24 \left(\frac{mg}{dl} \right)$$

5.3.4 Timestamp

The timestamp is important so that the blood sugar readings can be recorded and stored in the EEPROM memory chip in case the user is unable to connect to their Smartphone (phone's battery is dead, forgot their phone, etc.). The team decided to use the RTC module within the microcontroller, however when the board design was done, it was not included. Therefore, a different solution needed to be worked out.

The best solution was to use the Smartphone's clock accuracy to set the clock on our Smartwatch. The procedure for this is described in detail in the Bluetooth Low Energy section of the subsystem documentation (section 5.5). Once a UNIX Timestamp was received from the Smartphone, it was stored in the EEPROM. The internal clock (Timer 3) of the microcontroller was used to count at a rate of one pulse every 60 seconds, which was done by raising the timer flag when the system clock had counted to the hexadecimal number 0x23C364 and then creating an interrupt. The count would then update the timestamp in the EEPROM. Since the diabetic is expected to use this system at least 5 times a day, the clock should remain accurate as it is set based on the phone's clock after each reading. This works in the idle low power mode since the system clock remains running during that mode. Ideally by using the RTC module, we would have been able to put the microcontroller into sleep mode, the lowest power mode available. This would significantly extend the battery life of our smartwatch.

5.4 Design and Operation of LCD Subsystem

5.4.1 Subsystem Overview

The main purpose of the LCD screen is to give the user an easy and comprehensive way to interface with the glucometer without the use of the smartphone app. The main functions of the LCD screen are as followed:

- Give the user basic instructions for controlling the glucometer. This includes basic commands to guide the user through the process of taking a glucose measurement.
- Offer real-time glucose results. The LCD will need to be able to display glucose reading shortly after taking a measurement without the aid of a smartphone
- Review past glucose readings. The glucometer will store several past glucose readings until they are able to properly upload to the user's smartphone, via Bluetooth. LCD screen will need to provide a way for the user to review these past readings.
- Give the user a unique, enjoyable experience. We don't want our LCD to just be a basic, boring screen. We want to give the user an experience that sets our product apart from the competition. This will be done with clear commands, easy-to-use interface, and animations.

5.4.2 Subsystem Design

Our glucometer's main selling point is that it is a small, compact, easy-to-carry measurement device with an all-in-one design to reduce the hassle of traveling with several components. Therefore, several different factors come into play when deciding what LCD is best for our glucometer:

- Small, compact size
 - The LCD screen must fit within our compact design.
- Quick interface with the microcontroller
 - The LCD must be able to quickly talk to the microcontroller with very few wires to conserve space.
- Readability

- The desired screen should be able to show several words on the display at once, while still being easily read from an arm's length away. This implies that with a small screen, the resolution will have to be fine enough to clearly fit many words on the screen.
- Provide a unique experience
 - The desired screen should be high quality and will enhance the user experience.

We originally chose the SSD1306 OLED screen because we thought it best suited our needs listed above. The LCD screen was 0.66 inches across with a small breakout of board retrofitted by Sparkfun that would have aided in development. The breakout board was planned to be replaced in the final design and put into the final PCB board in order to optimize space. That OLED screen had a 64x48 dot matrix, but for its small size, it provided a clear, crisp picture. This would have allowed us to fit several words onto the screen at once and even do animations if we desired. The board came with the versatility of communicating with the microcontroller via SPI or I2C.

We chose to use SPI for a few reasons. First, we are most familiar with SPI. We have worked with it more during our previous semester during our several assignments. Also, the read/write protocols are much simpler than I2C. Third, SPI is much faster, which we felt to be very important to the overall user experience. We wanted our LCD to update as quickly as possible. This could allow us to add animations and other design features to our board. In addition, it allows the code to proceed to the Bluetooth stage, which can benefit from any additional speed for quick connections and data transfers. While SPI does require an extra wire, we felt that the simplicity and speed of SPI outweighed the extra space, which was not detrimental to our board size in the end.

The OLED screen from Sparkfun did not work for us. The main problem was that the code that accompanied the breakout board was written from an Adafruit controller. Since we were using a PIC microcontroller, this would not work. I studied the datasheet for the SSD1306 controller and found out how to set up the initialization of the screen through a series of byte instruction sets, but I had no code for actually setting what I

wanted to show up. I explored writing a bitmap to the screen, but without proper functions to transfer text into bitmaps, it would be impractical to write my own bitmaps.

Next we explored other code already written for an SSD1306 controller. We found one that was suitable for us and fairly easy to understand, and began testing this code. However, we had trouble implementing the code to work properly for our screen. The biggest problem was that the bitmapping code was written for a screen 128x48, so it would not display properly on the screen, and there were few functions written for formatting the display. For this reason, we decided to search for other solutions for our LCD subsystem.

The LCD screen we settled on was a PmodOLED screen found on Digilent Inc.'s website. The screen is a 128x32 pixel 0.9" display with a 15 pin ribbon attaching to a breakout board containing a 12 pin, 6 by 2 header hookup. We chose this for a few reasons. First, it came from the suggestion of our professor who already had some available for us to start testing and implementing immediately. This was crucial because we were already behind schedule due to all of the obstacles of the previous screen. Second, the website provided code and breakout board schematic for easy implementation into our system. Functions were already written to properly format the text size and placement on the screen, as well as textual animations that enhance the overall user experience. In addition, the schematic for the breakout board made it easy to adapt the LCD screen onto our final PCB design. There was some initial concern about the ratio of the screen size. With its thin rectangular shape, it would not have worked well with our initial pen cap design. However, when we made the decision to change our design to a watch, it fit in perfectly. The slender screen now allows room for the lancing device to lie adjacent to it without bulging out of the user's wrist.

In the end, the code implementation was pretty straightforward. The difficulty with the code was that it tried to be more universal by using the pilb.h file. This file was written to help the designer adapt the screen for any pic controller. The problem with that was

that it had difficulty dealing with pic32 controller. Essentially what I had to do was dig into the code and remove all uses of `plib.h` and add my own code written specifically for our pic32 controller. This meant removing all calls of strange `#define` variables used in `plib.h` with my own that called for the specific bits used in our pic32. In addition I adapted the initialization and SPI send functions that we used previously for this code.

5.5 Design and Operation of BLE Subsystem

The nRF8001 from Nordic Semiconductors was chosen for the Bluetooth Low Energy subsystem. This chip was chosen due the availability of an Adafruit breakout board for testing and prototyping, and also because the nRF is relatively ubiquitous in BLE applications and therefore has a robust community following, which was helpful in development. A sample program, `ble_A_Hello_World_Program.c`, and the associated libraries were also provided.

As listed above in system requirements, the BLE subsystem had three main functions – connect device and smartphone, transfer readings, and receive an updated timestamp. To this end, three functions were written: `ble_connect()`, `ble_transfer()`, and `receive_timestamp()`.

After the BLE subsystem is powered on, it begins to advertise and waits for a device to connect. While this is occurring, `ble_connect()` returns false. After connection is established, the ACI event `'ACI_EVT_CONNECTED'` is received, `ble_connect()` returns true, and data transfer begins using `ble_transfer()`. `Ble_transfer()` loops until no ACI events occur, and then sends the data via UART. While looping, `ble_transfer()` returns false. Upon transfer, `ble_transfer()` returns true, and the system knows that the selected reading has been successfully transferred to the smartphone. `Ble_transfer()` and `ble_connect()` are located in the `ble2.h` header file.

`Receive_timestamp()`, found in the poorly named `send_timestamp.h`, returns a Unix timestamp sent from the smartphone. When called, the function waits for the ACI event `'ACI_EVT_DATA_RECEIVED'`, buffers the received value, and converts it from a char

string to an integer. Unix timestamps are always 10 digits long, so the received data is truncated at 10 chars, which are converted to a string of ints and then concatenated.

The design challenges associated with this subsystem were predominantly related to understanding the extensive code provided and altering it to suit the needs of our system. The original `ble_A_Hello_World_Program.c` program operated in a continuous loop, so there was no need to return any values depending on which ACI events were received by the nRF. As a result, the `aci_loop()` function needed to be rewritten to return different indicators depending on ACI events. These indicators were also different for the different functions, so the `aci_loop()` function was duplicated and tailored to each function.

5.6 Design and Operation of EEPROM Subsystem

The EEPROM that we chose was a 25LC128, the same one we used in our class. We chose this memory device mainly because we had already worked with it and had SPI code already written for it, but also because it fit all of our needs. The EEPROM has a 16,384 x 8-bit organization. This means that it has 16,384 memory locations that hold 8 bits each, or one byte. 16,384 is 2^{14} meaning that the addresses for this EEPROM are 16-bit (2 bytes) address with the 2 most significant bits being “do not care” bits. The values that we store within the EEPROM are integers, and in a pic32 this means that all integers are made up of 32 bits, or 4 bytes. For that reason every memory location that we set up was a series of four locations. The 25LC128 works great for this because it allows up to a 64-byte page to be written at once. And with 2^{14} locations, we could technically hold over 4,000 readings at one time, but for practicality reasons we chose to limit that number. The overall speed of our system would be reduced if we chose to deal with every memory location.

The memory bank setup can be seen below. The memory “TIME” refers to the unix timestamp that we receive from the smart phone through the Bluetooth. This timestamp is the value that our counters use to give accurate timestamps on readings when not connected to Bluetooth. The next two spots are the current reading value and

timestamp. This will be the value that will be available for review from the LCD screen. These spots will be automatically updated every time a new blood glucose reading is taken. The following spots are a counting or memory values and timestamps. These spots are reserved for blood glucose readings that need to be saved because the Bluetooth was unable to connect to the smartphone app and transfer the data. There will be a limited number of spots, but they will be cleared after values are successfully transferred to the user's smartphone.

Table: EEPROM Memory Bank Setup

Memory Location	Memory Name
1	TIME
2	
3	
4	
5	Current Reading
6	
7	
8	
9	Timestamp of Current Reading
10	
11	
12	
13	First Memory Location
14	
15	
16	

17	First Memory Timestamp
18	
19	
20	
21	Second Memory Location
22	
23	
24	

Several functions were written for the EEPROM:

- EEPROM_init() = initializes the SPI on the pic controller and sets the appropriate bits, as well as sets the CS pin as an output.
- EEPROM() = basic function to send data through the SPI. It clears the interrupt flag, fills the SPI transfer buffer, waits for the buffer to clear, and returns the data received from the MISO.
- EEPROM_wait() = function to wait for the EEPROM to finish writing. It continually reads the status register and checks to see if the writing-in-progress bit (WIP) goes from 1 (still writing) to 0 (not writing).
- EEPROM_write() = function to write an 4-byte integer value to a given address. The function inputs an integer value and address. It breaks the integer up into 4 bytes and address into 2 bytes (since only the first 14 bits determine the address). Then goes through the write protocol of enabling write, sending the write instruction set, sending address, and sending the 4 bytes of the value as a page.
- EEPROM_read() = function to read the 4 bytes of a given address. The function inputs an integer address. It breaks the address up into the 2 necessary bytes. It goes through the read protocol of sending the read instruction set, sends address, and sends dummy bits in order to receive the 4 bytes through the MISO. The function then combines the four bytes into an integer and returns the value.

- EEPROM_empty_addr() = function finds the first available address with no value stored. The function starts at the first memory location and reads the value. It checks to see if there is any stored value. If there isn't, it moves on to the next memory location until it finds an empty value. The function returns the memory location value.
- EEPROM_clear_MEM() = function clears all of the values stored in the memory banks. Goes from the first to last memory location writing 0xFF to them. (active low)
- EEPROM_clear_TIME() = functions clears the TIME memory slot by writing 0xFF to it.
- EEPROM_clear_current() = function clears the current value and timestamp by writing 0xFF to it.
- EEPROM_clear_all() = clears all of the EEPROM by calling all three of the functions.
- EEPROM_new_value() = functions writes a value and timestamp to the current memory location and first available memory bank slot. Function writes to the current value and timestamp location, then uses EEPROM_empty_addr to find an open memory spot, and writes the value and timestamp to that spot
- EEPROM_read_all() = function populates an array with all of the values and timestamps in the memory bank. The function passes in a pointer and goes through the memory bank writing the values into the array. The idea is that this array will be used for data transfers through the Bluetooth.

Future improvements include rewriting the functions to write the value and timestamps at the same time. By writing a 8-byte page instead of two 4-byte pages, it eliminates a wait-cycle and would improve the speed of the device. However, since it is only a few bytes, it may not be needed because the function already works very quickly. The other improvement would be the EEPROM_read_all() function because it reads the entire memory bank even if it is not fully populated. Future improvement would be to create an array with its size being determined by how many values are actually saved in the memory

bank. This would improve the overall speed and efficiency of our device. However, this may not be needed either.

5.7 Design and Operation of Software Subsystem

5.7.1 Database and Skeleton Interface

The main objective of the software portion of subsystem 1 was to create a backend database in which to store the data and then allow users to access that data through an online web interface. This objective is detailed with the following goals:

1. Create a skeleton backend database available for both the web and mobile platform that will be able to be used to store basic glucometer data.
2. Create login interfaces for the web and mobile platforms and use that to access each user's individual database.
3. Have the ability to display tables for the data (sample test data will be used for the time being) on both the web and mobile platforms.

To achieve the first goal, database-hosting sites needed to be evaluated. After a meeting with the assistant director of Engineering & Science Computing (ESC), Ya'akov Sloman, it was decided that Amazon's Web Services (AWS), which includes a full suite of database, web, login, analytical, and other services would be used as the comprehensive service for hosting our entire application. The main advantage of AWS is its scalability. Building the application on AWS will allow this project to be completed for the purposes of this class as well as allow it to be fully operational on a larger, production scale. This is because adding more servers and storage is extremely simple on the AWS platform. Had this application been built out on a single server containing all the database and web components, it would be extremely difficult to transition the application from testing to running for production. Additionally, AWS' low to non-existent price point for the purposes of this project is also a tremendous plus.

The below will list, in detail, the components of Amazon's Web Services that have been used by our application and how they are used in our application. Following the list of

components will be an architectural diagram and explanation detailing how each component will fit together to build the glucometer database and application.

Amazon Web Services (AWS) Components

Elastic Compute Cloud (EC2): Allows for rapidly scaling the amount of servers. Initially the glucometer will require few servers—primarily a testing server for our team to develop—but having the flexibility to easily modify the number of servers will be extremely important when this product goes live.

Simple Storage Service (S3): The Simple Storage Service will be the backbone of the data storage of this project. Security features and backup. It claims to be able to easily store and receive data from not only EC2 and other Amazon services, but also various sources, which will be especially useful when working with Bluetooth. This is also scalable, which is critical due to the nature of our project.

MongoDB: NoSQL database service for the basis of this project's data storage. Data stored and returned are of JSON format. This database service was selected for its efficiency and scalability.

Cognito: This will be used for the login service of our application. It is compatible with authentication services such as Amazon, Facebook, and Google. Google with its OAuth 2.0 service has been selected for our authentication service. As a result users will login through their Google accounts. This ensures security as Google is a reputable login source.

Simple Queuing Service (SQS): Message queuing is absolutely crucial for our application since there can be many simultaneous readings and messages attempting to be stored or sent. Drastically simpler than having to create our own queuing service.

Elastic Block Store (EBS): Snapshots to backup EC2 instances and for launching new instances that you want to be similar to previous ones. Since this project is dealing with sensitive medical data that people will not want to lose access to, being able to backup instances is crucial for maximum uptime.

Lambda: Automatic management of compute resources and allows for rapid response to new information. Essentially, code in the form of lambda functions will respond to user events. This may be useful for alerting users based on their glucose readings and other advanced features.

SNS: Upon acquiring a user base, this can be developed for sending mass messages to users.

Express: A Model-View-Controller based web tool that assists in developing database driven web applications. Express is used to for developing the web interface and connecting the web interface to the database.

Architectural Diagram of how components will interact

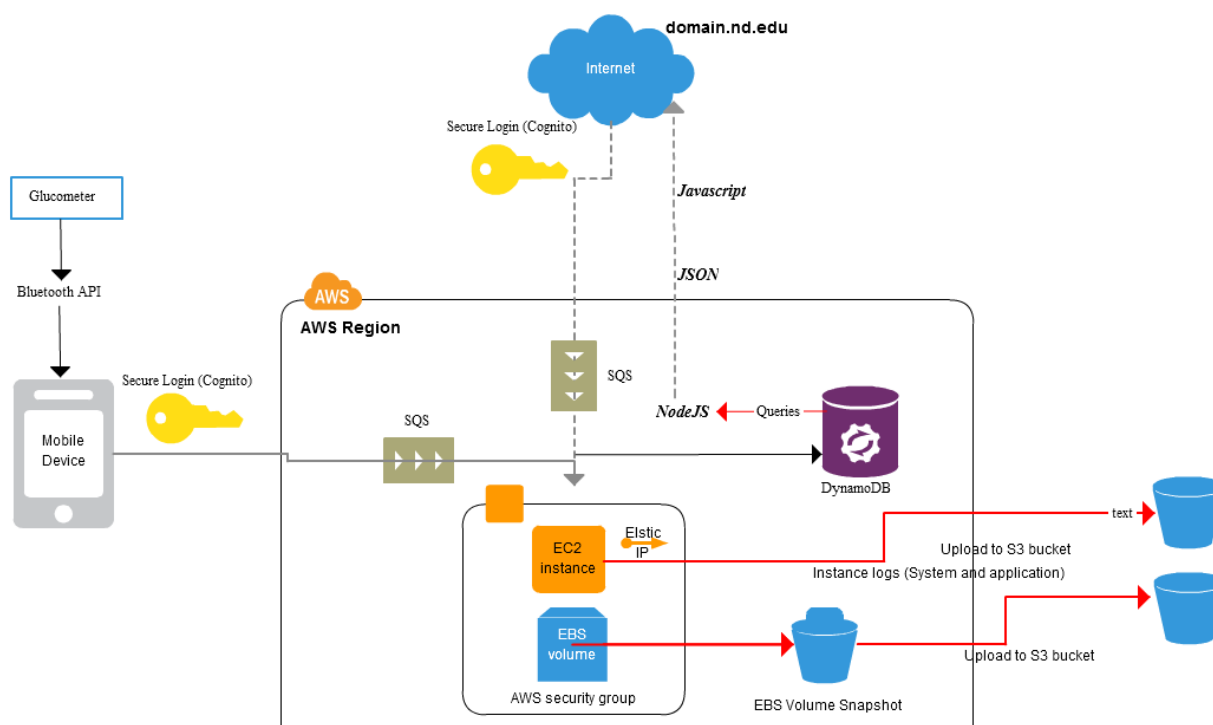


Figure 1 Architectural Diagram

Referring to the Architectural diagram in Figure 1, the essential process of the application will begin with data from the glucometer being passed to the phone through Bluetooth. A Bluetooth API will be selected and used in order to parse the data from the Bluetooth into the mobile or computer device. The device will utilize the Cognito login service in order to authenticate the user and the upload the data onto the Mongo Database. The Simple-Queueing-Service (SQS) will be utilized to queue the data upload in case the device does not have access to the internet or if excessive web traffic is congesting the upload. Once the data has been stored into the database, it can then be accessed by the web interface through NodeJS. The NodeJS development application being used, Express, is located on an Amazon EC2. Express is used to access the Mongo Database, query the data, and display it on the web interface. Note that the data being transferred is in a JSON format and will need to be parsed for usage. The data and servers stored is backed up by an EBS and stored onto Amazon's S3 service. In addition to being useful for backups, it is also useful for copying server instances, so transferring the application onto different servers is drastically simplified and expedited.

As previously mentioned, the web application is built out using Express, a web application framework. Express follows an Model-View-Controller (MVC) architecture in which the database (model) is connected to the HTML pages (views) through the JavaScript files defined with app.js and other JavaScript files in the routes directory (controller). The main advantage of using an MVC architecture is that the logic and the visuals of the application are separated and organized. This allows for much easier updating of the website if necessary.

5.7.2 Bluetooth Integration

This section of the subsystem featured a collaboration between the Bluetooth and software sides of the product. The main goal of this subsystem was to be able to send a value from the chip into the mobile device through a Bluetooth connection. In addition to being able to send a value, the components developed in the first subsystem were integrated so that once the value had been transferred into the mobile device, it can then be uploaded into the database. The upload into the database will include the read value, a timestamp, and the activity performed before and after the measurement.

To achieve the first goal of sending a value through a Bluetooth connection, several avenues of Bluetooth transmission into a phone were explored. This includes web application, iOS application, and Android Application. In the end, Android was selected as it provided consistent data transmission and is a widely used and readily available operating system.

The main advantage of selecting an Android application to perform the Bluetooth transmission is that since the application is native it will run significantly more smoothly and have easier access to the Bluetooth component of the mobile device. Development in Android can be performed through various means. For this project, Android Studio was utilized. Android applications are developed through a combination of primarily java and Extensible Markup Language (XML). Java is utilized primarily for the logic aspect of the application, although the visuals could also be designed using Java, while XML is utilized to easily create the layout of each activity (or page) of the application.

Being one of the most popular languages, Java is widely supported, well-documented, and highly compatible. Additionally, it is an extremely powerful language that includes many libraries and classes, including the ones necessary for the Bluetooth and database goals. Java includes a Bluetooth Library which supports scanning for Bluetooth devices, pairing between devices, and sending and receiving between devices. This results in a convenient method for interfacing between the board and the

application. Additionally, Java supports Parse cloud storage, which will be used to store the values into the database once the values are read into the phone.

Meanwhile, the main advantage of XML is its simplicity. The Android Studio development environment provides a visual environment for developing the application. This visual environment generates backend XML code which is readable and understandable even to those with little XML experience. The main goal of XML is to provide a language that is not only understandable by machine, but also intuitive for humans. XML in Android studio is used to generate the layout of the various activities. For example, to create a textbox with a button submission in XML, one would select a view type and then write the button and edit text tags. Each of these tags will have their settings, but beyond this, no other code is necessary. Everything is handled by Android Studio and the backend XML. Another advantage of Android Studio is the visual application designer. Android Studio offers an option to drag and drop widgets and layouts into a phone. Adding an item into the layout will automatically generate the XML code, allowing for rapid development. In order to generate Java logic for the items, it is a simple process of finding the view by ID and then setting a reference to it. In summary, Android Studio and XML allows for a convenient and simple development environment, which exemplifies rapid development.

With regards to the interface decisions, the application was kept visually simple for this subsystem. The beginning of the application prompts the user to log into Parse or create an account if they do not already have one. Once an account is created, the user will be directed to the main reading. The main reading page features a button to connect to the Bluetooth board and a text display that will change to the number once the value has been transferred. When the value has been transferred, the user can select the activity they were performing before and after the measurement and a button is provided to upload the data onto Parse.

5.7.3 Data Analytics and App Visuals

The goal of the final software subsystem is to provide data analysis on the data that was entered into the database. Additionally, this subsystem sought to visually improve the application by utilizing jQuery mobile for the charts section and performing visual design on the Android activity pages.

Data analysis on the readings involves generating a time-based line chart of the readings. The chart can be filtered based on a selected timeframe. The timeframe can be filtered by preset buttons or a slider for extra precision. In addition to the time filter, there is also an activity filter. The activity filter allows the user to filter the displayed readings based on the activity they performed before and/or after taking the reading. The chart also contains upper and lower limit lines to indicate what readings are in and out of acceptable range. These lines are signified with a red color. Below the chart are three crucial summary statistics. These statistics include the average blood glucose reading, the percent of readings in range, and the average number of readings per day. Note that all of these statistics are for the selected timeframe and activity filter. The importance of the time and activity filters is that they allow the user to see improvements in readings or problems. For example, performing readings after exercising may result in lower readings and thus the user can see the benefits of exercising. Meanwhile, the summary statistics provide useful and quick overviews of the data. For example, if the percent in range is abnormally low, then one can quickly realize they are having issues and may need to contact a healthcare provider. Also if the average number of readings is low, then the user can be reminded to take more readings. The average reading statistic is really useful for finding out the standard reading for a certain time period and filter selection.

As previously discussed, the Android Application itself is developed by Java and XML. And thus advancements to the application visuals were all built out by these languages with one plugin to help generate a connect status visual. Essentially, the login page was updated so that the edit text fields are centered and has more streamlined label texts.

Additionally, a background was added to each page in order to prevent the application from appearing barren. All of this was done by the default XML features available in Android Studio. Meanwhile, the reading page also had the same treatment where all of the labels were more streamlined and an appropriate background for the application was added. In addition, a circular label was created to more effectively display the connection status of the application. If the connection is active, then a green check within a circle will display to indicate an active connection. Otherwise, a red disconnect circle will appear. This allows for rapid visual identification of the connection status. This improvement of aesthetics is necessary to create an appealing application which creates a sentiment of welcoming to users.

On the charts side, the charts were built out using a combination of Google Charts and jQuery mobile. Google Charts provides the tremendous advantage of being widely supported and built out in HTML5. Google Charts is well documented with numerous examples, thus making development much more precise. Additionally, it provides numerous available charts options for modularity and future additions.

The main advantage of HTML5 is that it is widely supported by mobile devices. As a result, the chart will have no difficulty appearing on various platforms. HTML5 provides Cross-browser compatibility (adopting VML for older IE versions) and cross-platform portability to iOS and new Android releases. Additionally, no plugins are needed. The lack of plugins is particularly important because numerous non-HTML5 charts require flash, which is not default installed on phones. As a result, users may receive plugin errors if a non-HTML5 chart is used.

HTML5 also supports the use of jQuery mobile. jQuery Mobile is a HTML5-based user interface system designed to make responsive web sites and apps that are accessible on all smartphone, tablet and desktop devices. It follows the “write less, do more” mantra, where rather than writing unique applications for each mobile device or OS, jQuery mobile allows one to design a single web application that works on all popular

platforms. Another huge advantage of jQuery mobile is the ThemeRoller. The theme roller makes designing the components of a web app much simpler. The ThemeRoller is an online interface which allows you to select how your buttons, text, backgrounds, and so on will look like. After selecting these options, you can download a zip file containing css and javascript which generates the proper theme. The code it generates can then be further optimized since you are able to edit them. A huge advantage of jQuery mobile is that it applies dynamic web development. With dynamic web development, the appearance of the website will adjust based on the size of your screen. As a result, for any reasonable screen size, the website will still look great. To use jQuery mobile, all one would need to do is to include the proper script tag links and then utilize its functions. See below for code snippets demonstrating its use.

Utilizing Google Charts on a page is a similar process to using jQuery Mobile. To use Google Charts, merely include the necessary script tags in the HTML header, these are specified in the documentation, and then include the types of charts you want to use in the JavaScript code (an example of this can be seen in the code snippets). To set the data, the addColumn and addRows functions are used in a loop as the data is being acquired from Parse.

Parse has similar usage to Google Charts in that one needs to begin by adding the necessary Parse script tags. Once those tags are added, the currentUser function can be utilized to identify who is currently logged into Parse on this device. With that user identification, queries can be performed on that user's table information. As the data is being retrieved, we add it into the Chart rows which then allows the data to be displayed on the Charts. Parse was utilized for its simplistic but highly effective database cloud system. Parse comes with built-in user identification and storage functions as well as a rapid querying system.

Overall, the combination of Parse, Google Charts, and jQuery mobile makes the charts page extremely versatile. All of these tools support the use of HTML5 and are very well

established and secure. Note that the resulting web app was uploaded onto an Apache2 Notre Dame server located on Amazon Web Services. Apache was selected as the web server since it is one of the most popular, longstanding, and reliable web server software. This means there is a wealth of documentation and is compatible with essentially every operating system. There is also the benefit that it is well maintained and constantly updated, resulting in an overall secure and feature rich web server software.

5.8 Design and Operation of Mechanical Subsystem

5.8.1 Main Design

5.8.2 Lancing Device

Lancing devices are one of the key instruments in the successful management of diabetes. Lancets are small and sharp objects or needles used to prick the skin. This allows a small drop of blood to surface onto the pierced skin that can consequently be tested for blood glucose levels. Tools like blood glucose monitors and blood glucose test strips are used in tandem with the lancet in the testing of blood glucose levels.

While the option does exist for a diabetic to prick himself directly with a lancer, most diabetics prefer a lancing device. A lancing device, then, is a device that firmly, securely and safely grips the lancet in a way that allows for the lancing operation to be performed simply with the click of a button. Lancing devices allow the flexibility of setting the piercing force based on the thickness of the patient's skin, with thicker skin requiring a greater piercing force etc.

Owing to the fact that lancets are to be changed after a single use, lancing devices need to be designed in an intuitive and robust way, allowing for simple operation and dependability, especially because of the additional risk inherent in handling sharp objects.

Current State

Since lancers remain largely the same across different manufactures, while lancing devices do not, the discussion here will be centered more around the lancing devices that house lancers available on the market rather than the lancers themselves. The main point of difference that the various manufacturers stress in marketing their lancing devices is pain—with every manufacturer claiming that their lancing device is the least painful to use.

To address the issue of pain, many manufacturers have turned to allowing the end user the ability to adjust the “depth setting” on their device. Essentially, the depth setting is intended to accommodate various skin thicknesses, so that the piercing force is adjusted to match the texture of the user’s skin; a user with softer skin would use a shallower depth setting, for instance, than one with thicker skin.

The second challenge manufacturers grapple with is that of the initiation mechanism for the lancing device. As a point of clarification, most lancing devices on the market operate based on energy storage and release from a loaded spring, or spring-like material, with a button on the lancing device as the interface used by a patient to unleash the loaded spring and initiate lancing. In general, a softer button press is preferred, with the most successful lancing device manufacturers being ones that most responsibly manage the balance between ease of execution and safety; while a softer button press is easier on the patient, a button that is too sensitive increases the risk of injury.

Intended Design:

With an understanding of the market landscape and how that interfaces with consumer preference, the team’s design plan is to have a lancing device that is first intuitive to use, safe and easy to handle, and less painfully executes the lancing operation.

To make the device intuitive to use, its operation will be modeled after existing and familiar devices like the retractable-pen, a model that most of the thriving lancing

devices leverage. For the purpose of ease of execution, loading configurations of the spring will be investigated to determine which options best preserve the integrity of the spring while allowing for a soft and safe button press for the initiation of lancing. Finally, to address the challenge presented by the pain inherent in the lancing operation, the team will look into even further refinement of the “depth settings” present in models of lancing devices readily available in the market place and implement said refinements in the team’s lancing device. There will also be consideration given to unorthodox approaches like vibration, to be used as a numbing tool for the area to be pierced.

As a final note, while lancing devices existing currently in the marketplace do not appear to be optimized for compactness, the teams lancing device will be designed with portability as a key consideration, hence offering the team’s lancing device an edge in the marketplace.

5.8.3 Test Strip Cartridge

5.9 Design and Operation of the Lancing Device

5.9.1 Requirements:

- Portability
- Compatibility with glucometer subsystem
- Compatibility with existing generic lancets
- Different settings to accommodate different skin thicknesses

5.9.2 Approach:

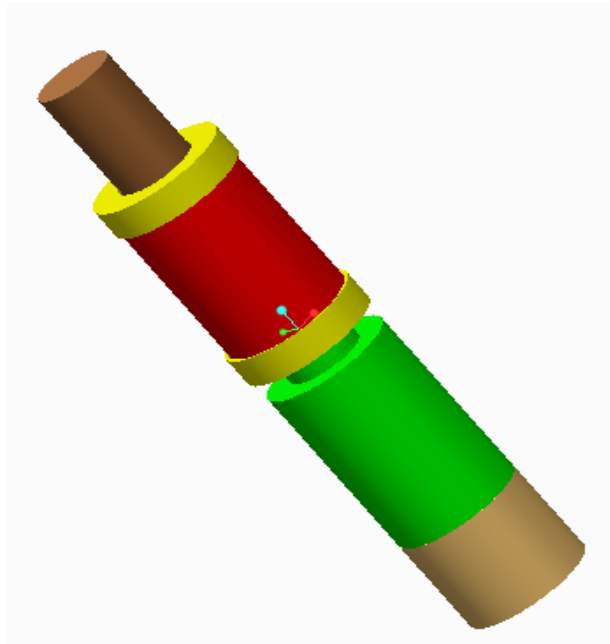
- The parts of the lancing device were modeled using Creo (commercially available CAD software)
- Modeled parts were printed using 3D printers
 - Makerbot Replicator 2X (Preliminary Prototyping)
 - Fortus 250mc (Final Prototyping)

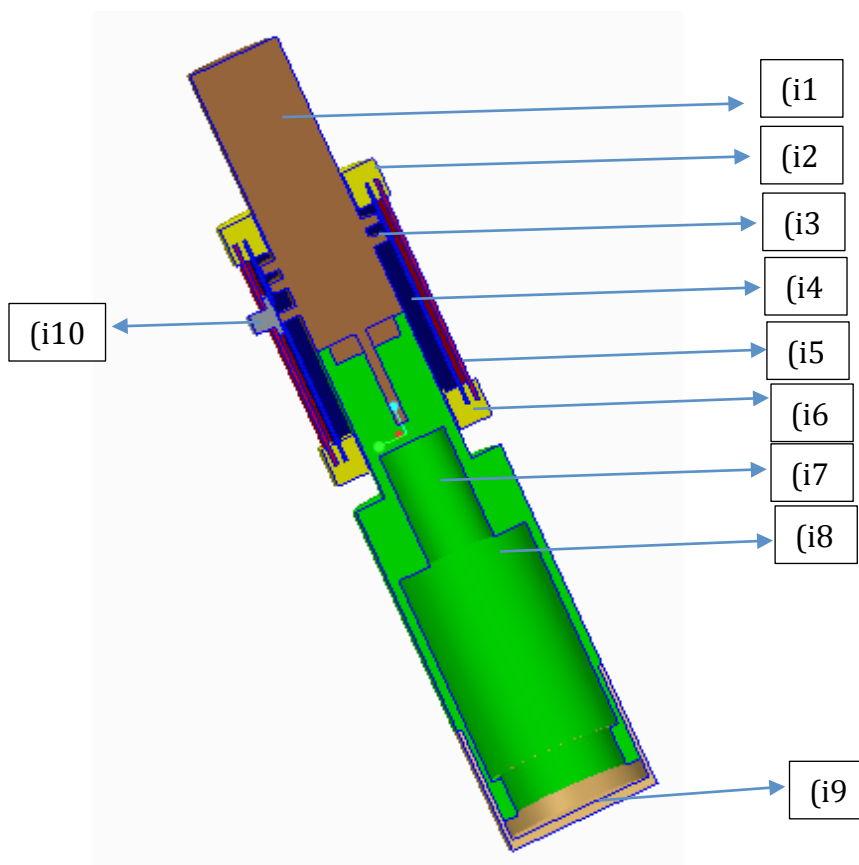
5.9.3 Challenges:

- Poor resolution on the best available 3D printer (Fortus 250mc)
 - Design scaled up to accommodate poor resolution on available 3D printer
- Portability requirement compromised with scaled up prototype
- No injection molding resources available for manufacturing parts
 - Part quality would see dramatic improvement with the use of the injection molding approach
- Time constraint significantly limited number of iterations of initial design
 - Limited time to react to poor resolution on 3D printer
 - Springs for ideally scaled model used for scaled up model (non-ideal)

Final Design I:

Lancing device at ideal scale (No physical model built):





Part Number	Description
i1	Puller
i2	Top Cover
i3	Spring holder
i4	Inner Wall
i5	Outer Wall
i6	Bottom Cover
i7	Lancet holder
i8	Bottom Case
i9	Cap
i10	Trigger

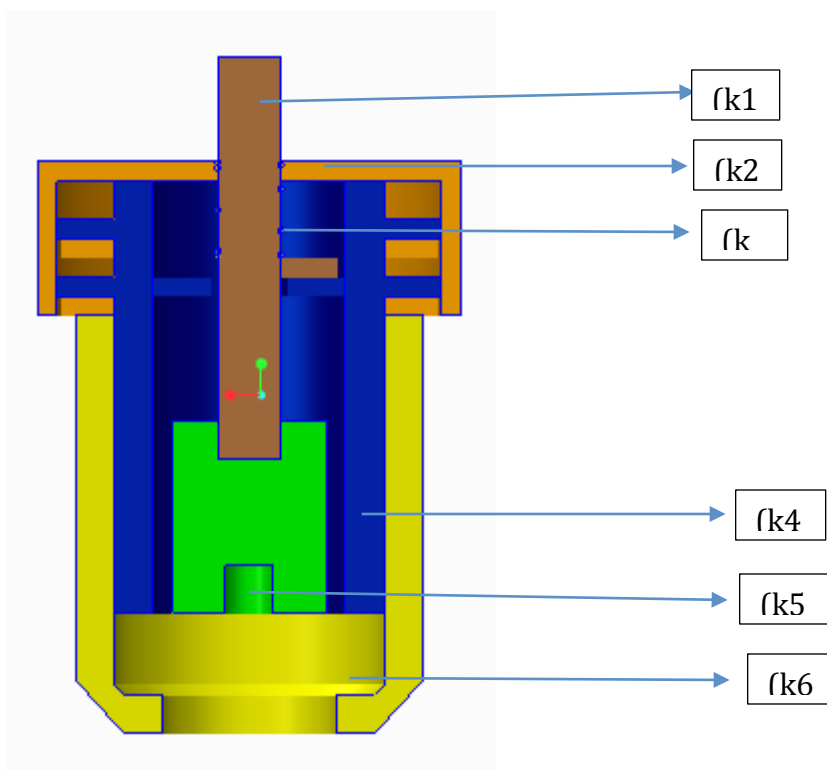
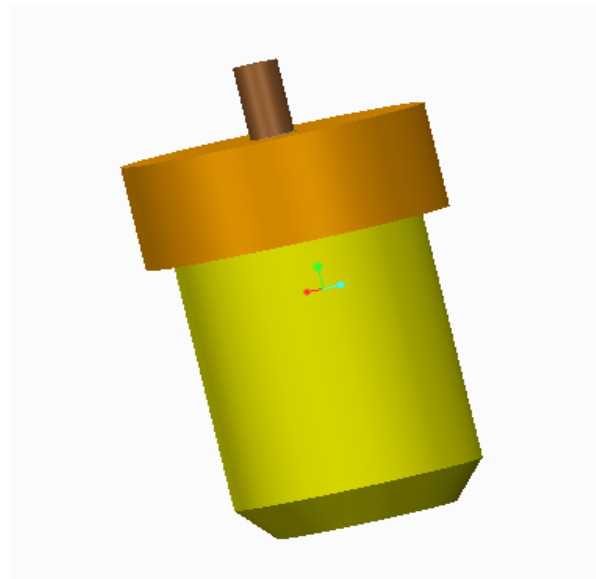
Operation:

1. Secure generic lancet in Lancet holder (i7).
2. Engage spring by pulling the Puller (i1) until a clicking sound is heard.
3. Put finger at the end of the Bottom Case (i8) with the Cap (i9) off.

4. Push the Trigger (i10) in to initiate lancing operation to break skin.
5. Take blood-glucose reading.
6. Remove lancet and dispose of it safely (i7).

Final Design II:

Lancing device at exaggerated scale for demonstration (Physical model printed using Fortus 250mc):



Part Number	Description
k1	Puller
k2	Top Cover
k3	Spring
k4	Inner Case
k5	Lancet Holder
k6	Outer Case

Operation:

1. Secure generic lancet in Lancet holder (k5).
2. Engage spring by pulling the Puller (k1) until Red marking is revealed.
3. Twist Puller (k1) in either clockwise or counterclockwise direction for a quarter of a circle.
4. Adjust Outer Case (k6) to desired depth to match skin thickness.
5. Put finger at the end of the Outer Case (k6).
6. Twist Puller in either clockwise or counterclockwise direction for a quarter of a circle. This initiates lancing operation to break skin.
7. Take blood-glucose reading
8. Remove lancet and dispose of it safely (k5).

Glucometer Case

Requirements:

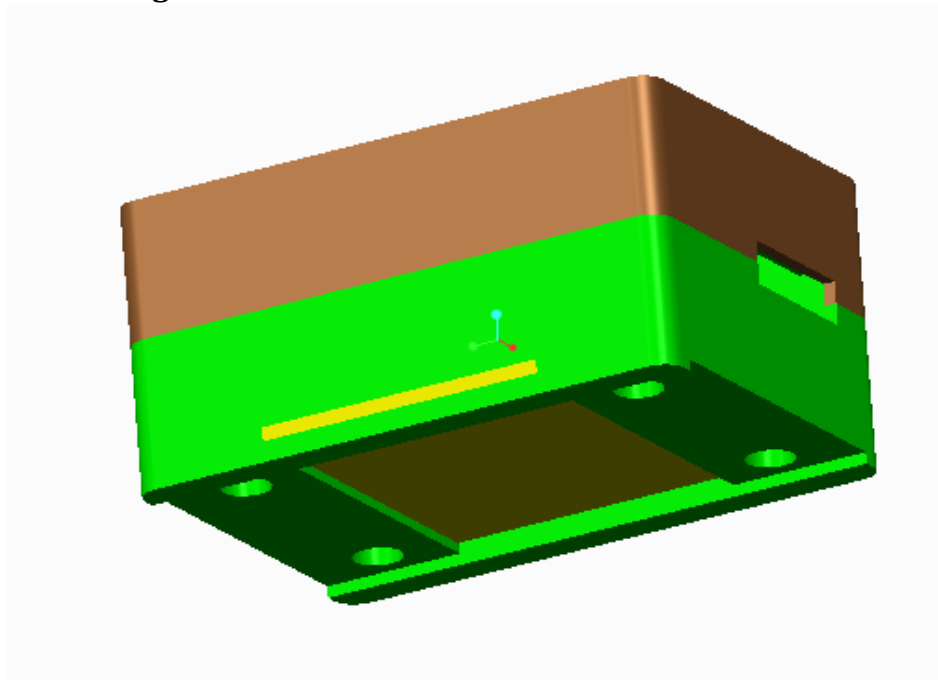
- Portability
- Compatibility with Lancing device subsystem
- Ability to hold all circuit components
- Provide intuitive access to control interfaces (buttons etc.)

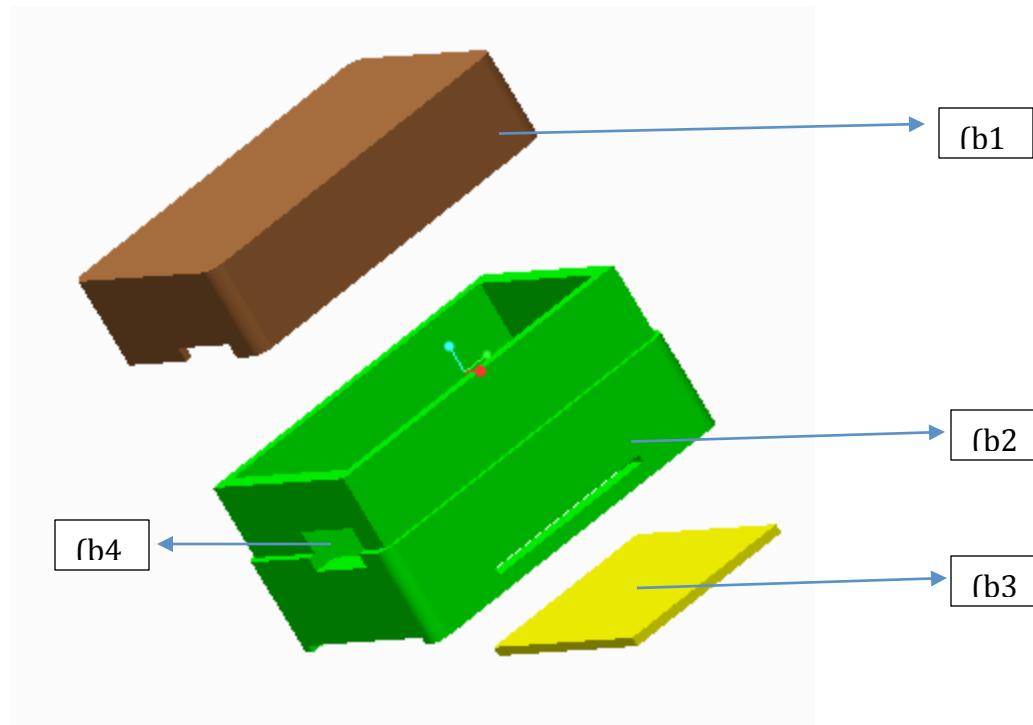
Approach:

- The parts of the lancing device were modeled using Creo (commercially available CAD software)
- Modeled parts were printed using 3D printers
 - Makerbot Replicator 2X (Preliminary Prototyping)
 - Fortus 250mc (Final Prototyping)

Challenges:

- Poor resolution on the best available 3D printer (Fortus 250mc)
- No injection molding resources available for manufacturing parts
 - Part quality would see dramatic improvement with the use of the injection molding approach
- Time constraint significantly limited number of iterations of initial design
 - Limited time to react to poor resolution on 3D printer

Final Design:



Part Number	Description
b1	Top Cover
b2	Circuit Housing
b3	Battery Cover
b4	Test Strip Holder

Operation:

Readings

1. Securely insert test strip into test Strip holder (b4).
2. Observe readings on LED screen.

Replacing Battery

1. Remove battery cover (b3).
2. Replace battery.
3. Reinstall Battery Cover (b3).

5.10 Power

The glucometer design requires that it be compact and mobile. Therefore, a small button battery would be needed in order to meet these requirements. We originally settled on CR2330 lithium button cell battery. It's 3 volts, 265 mAh with a small 23.3

mm diameter. During the board design process, we realized that the board size that we were targeting was not going to happen. The boards were going to be larger than we had originally planned. This presented an opportunity to rethink the CR2330 battery. The larger CR2450 has a much higher capacity, but still fits within the new board size. It's still 3 volts, but its capacity is 620 mAh, which will double the lifetime.

The battery lifetime of the glucometer was calculated by measuring the current during for a full cycle and finding the average. The average current while in use is 58 mA, and 1 mA while in sleep mode. The average time the device is on per use is 2 minutes. Device is expected to be used 5 to 10 times a day, and will be reviewed several times a time. We chose a conservative 15 times a day. That means that every day the device uses 29 mAh while in use and 23.5 mAh in sleep mode. The device will last around 12 days with a power use of 52.5 mAh per day.

6. System Integration Testing

6.1. Subsystem Integration Testing

6.1.1 Hardware Subsystems

The various subsystems were systematically tested to ensure proper operation. Each subsystem was tested independently, and then gradually integrated with one another.

The glucometer circuit was tested by using a full cartridge of test strips and the two levels of control solution purchased and described in detail in section 5.3. The blood sugar values displayed on the LCD after every use of a test strip was checked to ensure that it remained between the two ranges of the control solutions (40-85 mg/dl and 240-340 mg/dl). The values were consistently within range, varying slightly from test strip to test strip but remaining within the given range. We noticed that the values on the lower range were closer to the higher part of the range, with numbers consistently in the 70's and 80's. This is due to the fact that our best-fit line for our regression formula, explained in detail in section 5.3.3, was slightly above the data points taken for that

control solution. Thus, our glucometer system performed as expected and the errors were accounted for due to the lack of access to blood testing laboratory.

The BLE transfer system was tested using the Nordic Semiconductors application, available from the app store, before it was integrated with the Android application. This application allowed data to be both sent and received.

EEPROM was tested using the USBee Suite logic analyzer. All of the functions were individually tested by reading the MOSI and MISO of the SPI ports starting with the basic read/write functions. After that, the more complex functions could be tested by seeing that the correct integers were being written and read back from the memory bank.

6.1.2 Software Subsystems

In order to demonstrate the databases functionality, one must be able to perform the following tasks: successfully log into the web interface using their Google account, view a table of their glucometer reading data, and enter new readings into the database which will appear on the above table. For further verification, one could log into the database account and verify that the data is present and as the user specified.

It was quick and easy to test the functionality of the Bluetooth application. So with regards to testing of the application, the Bluetooth board attempts to connect with a device and then would send a value of 1 to any receiving and connected Bluetooth device through the Rx channel. Note that the Bluetooth board had an LED screen to determine the current state of the Bluetooth connection and transmission. At the start, the LED will prompt the application to attempt to connect with the board. Once connected, the LED will prompt a connected message. The application also has a connected feedback text. The disconnected text at the top will change to connected once the application connects with a device. Once connected, the LED will state that it is attempting to transfer a value onto the connected the device. If the transfer completes, then a completion message will appear and the value will appear in the

center text of the phone. Otherwise, the transfer attempt will continue until terminated. If connection or transferring failed, then there is likely an issue that needs to be resolved using the troubleshooting steps outlined in the troubleshooting section. Now that the value is on the phone, pressing the upload data button will upload the data onto the Parse database with a proper timestamp. A successful upload will be prompted with a data upload success message, whereas a failed upload will provide a data upload failed message. The data should have no difficulty updating unless an internet connection is not present.

In the case that measurements were completed without access to the internet, the glucometer device will store the measurements. Once internet access is obtained, the user can connect with the application and send all of the data stored on the glucometer at once. Once the upload data button, all of that data will be uploaded and the buffer of stored values will be uploaded. Note that all values are stored in a buffer on the phone until the upload button is pressed. This makes it so that multiple measurements are supported.

In order to verify and test the functionality of the data analytics and visualization of the app, the app and charts page was run on varying devices. Through this we were able to verify that the integrity of the application and the website does not vary device to device. So essentially, we loaded the .apk on multiple Android phones with varying screen sizes and made sure that the pages appeared fine on all phones. Had we had access to more devices, further testing could be performed. To know that the charts were functioning, merely interact with them. Attempt to change the filters around, use the slider, and so on. If the data being visualized matches the data being entered, then the charts are clearly working. Manual calculations may also be used to ensure the summary statistics are valid calculations.

6.2. Design Requirement Demonstration

The finite state machines described in section 5.1 were tested by running through the entire sequence and allowing for all the various paths to be taken (timeouts, bad BLE

connection, etc.). The only path that was not fully tested was the EEPROM storage and retrieval working with the Bluetooth system since we ran out of time before we were able to integrate the two together. Testing the finite state machine in this manner required that each subsystem was working properly, and demonstrates that the design requirements were met.

7. User Manual

7.1 Hardware

1. Before interacting with the glucometer, the device should be sleep mode. In sleep mode, the OLED screen will be off to preserve the battery.
2. To wake the glucometer, press one of the two buttons. Each button will perform a different instruction. Button 1 will activate the device's review mode, and Button 2 will begin to sequence to take a blood reading.
3. When Button 1 is pushed, the device goes to review mode. In review mode, the most recent blood reading will be displayed, and Buttons 1 and 2 become cursors. Button 2 will move to the second most recent reading, then third most recent, and so on. Button 1 will go in the opposite direction. It will go to the least recent reading, then the second least recent, and so on. If only one reading is present in the database, the value will not change at all. Wait for sleep mode to return to other modes.
4. When Button 2 is pushed and the blood reading sequence has begun, the OLED screen will read "Insert test strip and press button when ready." The user should follow these instructions and place a test strip in the slot on the side of the device. Press Button 2 when the test strip is securely in the slot.
5. Next, the device will prompt the user to apply blood to the test strip. The user should use the lancing device attached to the side of the device. Instructions for the lancing device are below.

6. The blood drop should be applied by slowly moving it to come in contact with the thin edge of the test strip. The blood should NOT be placed on top of the test strip because this will often cause a bad reading.
7. In the case of a bad reading, the device will display “Bad Reading” and go back to the beginning of blood reading sequence. Refer back to #4.
8. If the reading is successful, the blood glucose number will be displayed on the screen for 5 seconds. Afterwards, the device will prompt the user to open up the smart phone app and connect via Bluetooth.
9. Once the device successfully connects to the app, a transfer of the data will begin automatically. The OLED screen will keep the user updated to the status of the transfer. When the transfer is completed, the device will show the blood glucose value again for 5 seconds, and then go into sleep mode.
10. If the device is left stagnant for more than 30 seconds, the OLED screen will show a 3 second count before returning to sleep mode. To prevent this, simply push any button.

7.2 Lancing Device

1. Secure generic lancet in Lancet holder (k5).
2. Engage spring by pulling the Puller (k1) until Red marking is revealed.
3. Twist Puller (k1) in either clockwise or counterclockwise direction for a quarter of a circle.
4. Adjust Outer Case (k6) to desired depth to match skin thickness.
5. Put finger at the end of the Outer Case (k6).
6. Twist Puller in either clockwise or counterclockwise direction for a quarter of a circle. This initiates lancing operation to break skin.
7. Take blood-glucose reading
8. Remove lancet and dispose of it safely (k5).

7.3 Software

Installation of the glucometer reading and charts application requires a few simple steps. First the apk file must be transferred onto the phone. This can be done by either downloading and acquiring the apk file from the phone or transferring the apk file from a computer into the phone using a mobile. Once the apk file is on the phone, merely open the apk file in order to install the application on your phone. The installation process is automated and will result in a Glucometer application appearing in your list of applications. The application can now be opened and used.

An alternative installation method would be to compile the source code in Android Studio and have Android Studio target the desired phone as the testing device. In order to do this, connect the phone to your laptop with a USB-to-micro-USB cable and click the run and compile button on Android Studio. This will install the application on the device as well as open it.

To set up for a measurement, have the glucometer and the application reasonably close to one another. Setting up the software side of the product requires one to merely open and run the application. If the application is not available on the phone, follow the directions in 7.1 to install the application. Installation of any other software requirements are not necessary as long as a minimum of SDK 14 is installed (Ice_Cream_Sandwich). This SDK was released in 2011, so any Android phone which has been updated before that time will have no issues. To perform the reading, log in or create an account on the startup activity. This will lead to the reading screen. Now perform the measurement on the glucometer and attempt to send the data to the application. Make sure that both devices return a connected state. The phone will now receive any reading streamed data.

The first troubleshooting step to perform in the event the application is having difficulty is to restart the phone. There is the potential that another Bluetooth-based application is

interfering with the glucometer application or some running process on the phone is causing issues. The second measure would be to uninstall the application from the phone and reinstall it. It may be possible that the apk install failed and a fresh install will remove any issues.

If issues still persist, not that a few conditions on the phone have to be met for the application to be properly running. Firstly, ensure that Bluetooth is enabled on the phone. In order to perform a Bluetooth transmission, it is important that Bluetooth is actually enabled on the phone. Generally, to enable Bluetooth on a smartphone, drag the top bar down and make sure the Bluetooth button or tab appears active. If it is inactive, tap the Bluetooth icon to activate it. Secondly, ensure that the internet is active. While the internet is not necessary for receiving the reading, if you are receiving a “data upload failed” message, then this means the app was unable to connect to the parse servers via the internet. As a result, follow a similar procedure as the Bluetooth enabling procedure and turn on the internet on the phone. A third potential issue may be that you refused to allow the application access to Bluetooth or internet. To get around this issue, reinstall the application and make sure to accept when prompted whether the application should be allowed to access Bluetooth or internet.

Now if the issue is related to the reading not being read, there may be a couple potential problems in addition to the Bluetooth enabled and allowed status. To send a proper reading, the glucometer measurement device must be reasonably close to the mobile device. Try closing the distance between the glucometer and the phone and retrying the measurement. Another potential issue may be interference or another device accidentally pairing with the glucometer. If this issue arises, attempt to take the measurement in a more private area or try to turn off other Bluetooth devices such as another phone or laptops.

More advanced troubleshooting of the application will require interaction with this source code. This method will require one to perform the alternative installation method, which

involves running Android Studio, connecting the phone to the computer, and compiling the application onto the phone. Doing this will allow the user to run the application while seeing the status of each action in the logcat. The logcat will record any errors and alert them to the user. This method is not recommended unless the user is advanced and is well aware of the inner workings of the application.

8. To-Market Design Changes

8.1 Software

Improvements to the product would include increasing the number of mobile platforms compatible with the product. Currently, the Bluetooth reading portion of the product only works on Android-based mobile products. Making it compatible with iOS and Windows would drastically increase the accessibility and thus popularity of the product. In order to do this, either the Bluetooth reading portion of the application must be ported to become a web application or native versions for iOS and Windows must be made.

Other improvements may be polishing of the application. For example, the login screen could include a forget password/username feature. E-mail verification could also be a feature to prevent an excessive amount of fake usernames from being created. The application could also have reminder features. For example, you could schedule the regular time you would normally take a blood sugar measurement and then the app would the phone vibrate to remind you to take the measurement.

Additionally, the application could go into more detail about what the user is eating rather than just consider eating as a general activity. As a result the application would simultaneously double up as a diabetes dieting application. The feature to search for and connect to nearby healthcare providers may also be effective for diabetes care.

8.2 Hardware

We have identified three areas for improvement of the device – form factor, user interface, and battery life.

As this is an early prototype, there is much room for improvement in form factor. Ideally, we would like to make the device both smaller and sleeker before releasing it to market. To do this, we would need to progress from the current 3D printers to printers with more precise capabilities. We would also like to incorporate a variety of options for attaching the device beyond a simple wristband, including a waistband attachment and an armband.

The mechanical system has a lot of room for improvement. For the scope of this year, we focused on designing a compact lancing design. With the proper injection molding process, our lancer would be close to its final design. For the main casing, we would include a compartment for a day supply of test strips. This way, a diabetic would not need to carry a separate test strip cartridge. This would be able to be designed with the reduction in size for the board layout as described above.

The user interface, consisting of the LCD screen and the buttons, also has room for improvement. Before releasing the product to market, we would like to improve this interface by increasing the size of the display in order to display more information and increasing the functionality of the buttons, possibly to cycle through previous glucose readings and view the same summary statistics available in the application. This would also require increased storage on the EEPROM.

Due to time constraints, we were unable to implement a system that monitors battery life. This feature is a necessity before releasing the product to market. Additionally, there are some improvements to be had in the power consumption of the device and these changes would be made in order to improve battery life. An additional consideration would be to utilize a rechargeable battery instead of the watch battery.

The glucometer would need to be tested extensively to ensure accuracy. In order to do this, a fully equipped blood-testing laboratory would be required. This would allow for a more accurate regression formula to be created and help integrate the temperature into the blood sugar readings. Significant US FDA testing would be required before the glucometer would be allowed on the market.

One oversight in our project was the power. We used a 3 Volt button battery, but we prototyped with 3.3 Volts. Also, the voltage slowly declines as the energy drains. Our bias voltage for our ADC glucometer circuit needs to be at 3.3 volts to be accurate. In the future, we would add a 5 Volt battery with a voltage regulator to deliver a constant 3.3 volts.

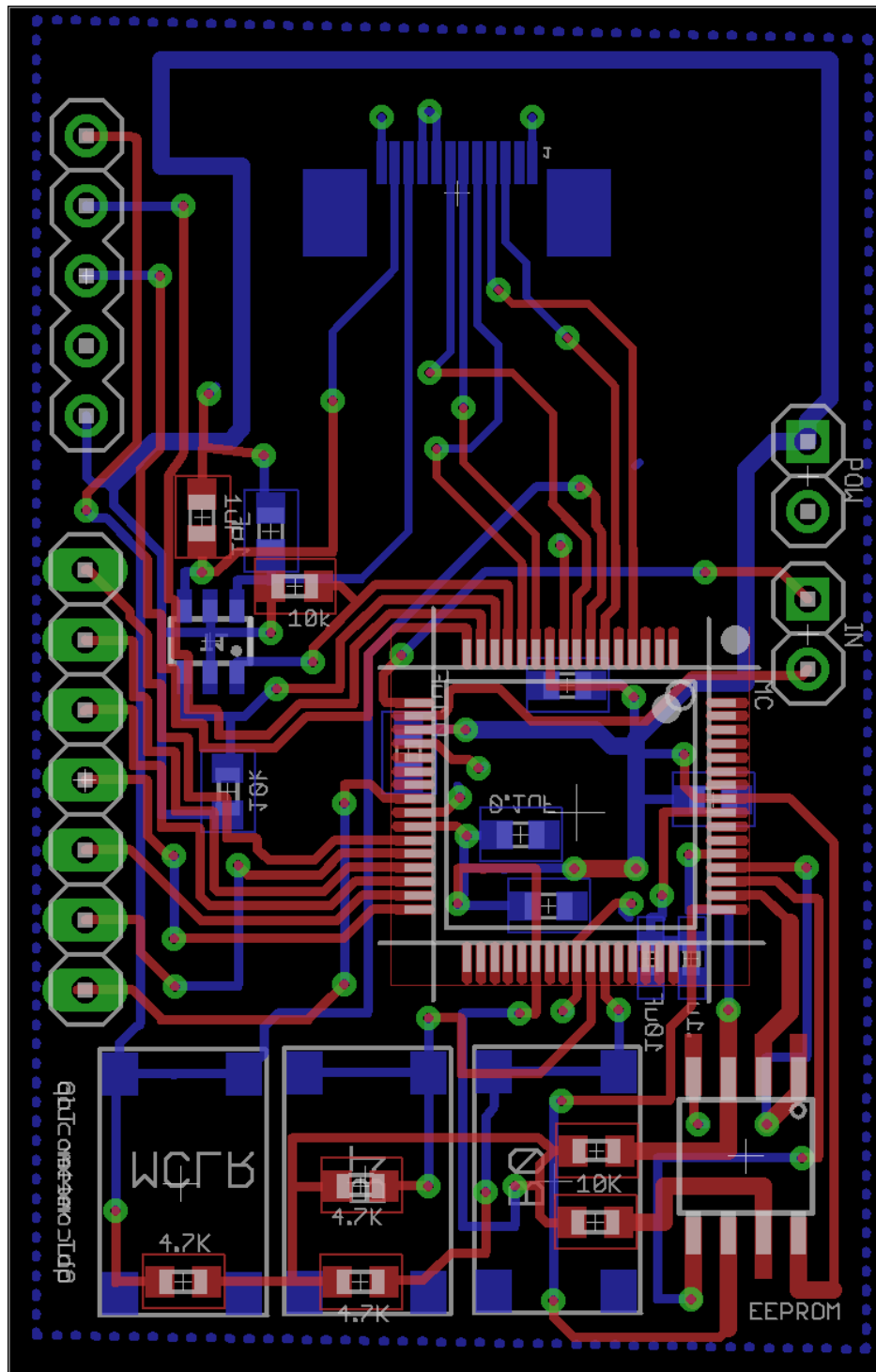
9. Conclusions

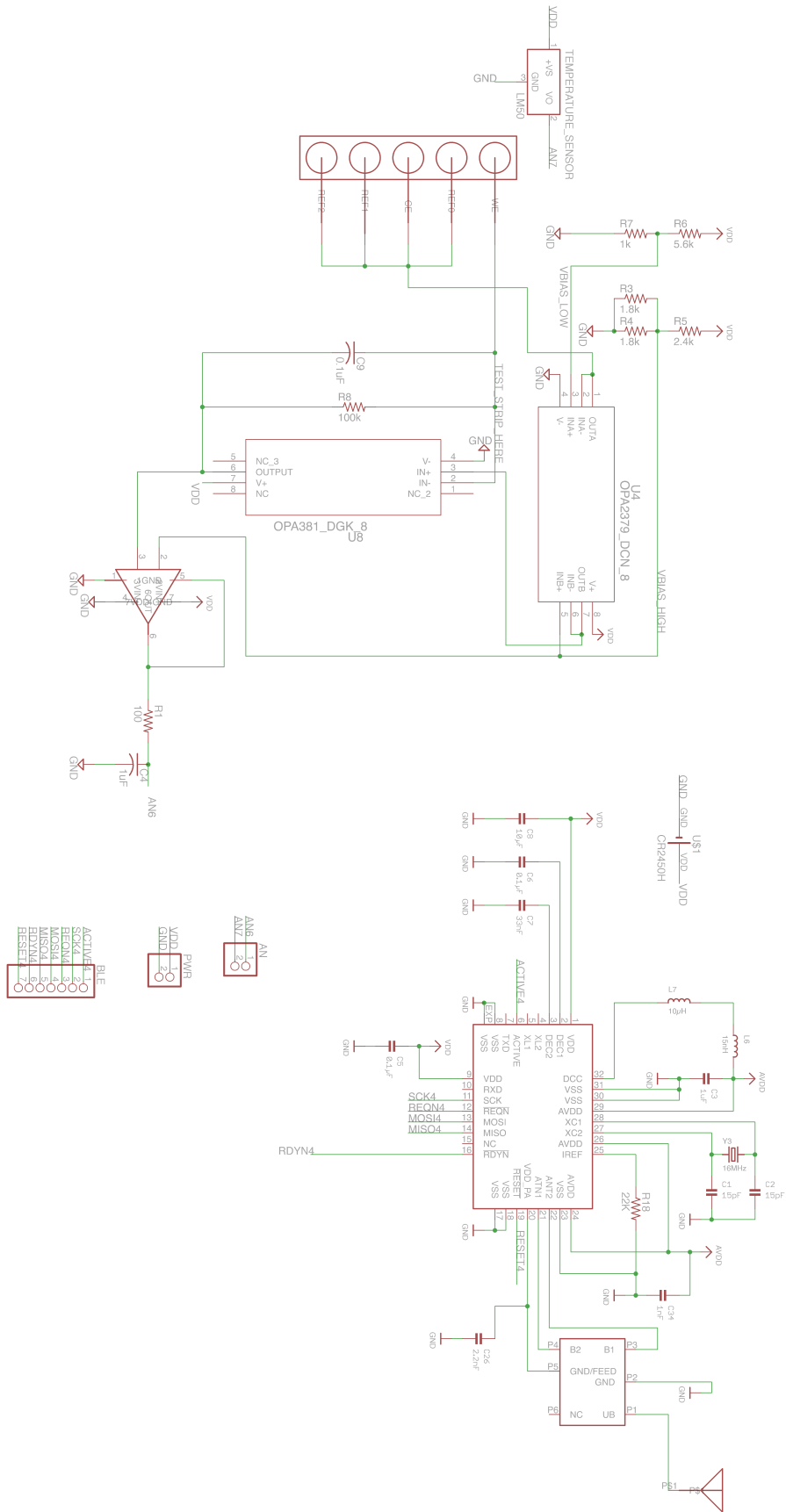
Our prototype design met initial requirements and is a promising first step towards bringing this device to market. While there were some shortcomings in this iteration, namely one of the PCBs not working and limited testing of the EEPROM, the prototype operated in much the same manner as a final version of the product would be expected to operate. In addition, the software and hardware components of the project seamlessly interfaced with one another. There are certainly some changes to be made, which are previously addressed in this report, but this iteration of the Little Vampire is a successful first prototype and meets the initial outlined requirements.

10. Appendices

Appendix A. Hardware Schematics and Board Layouts

A1. Top Board





Appendix B. Microcontroller Software Reference

Overall Main File: main.c

Header Files (some include .c files to coincide with the header files but for sake of understanding the header files explain the functionality of the underlying .c file)

General Code:

ADHEADER.h -----ADC code
 Button.h -----Button Interrupt and Polling Code
 EEPROM.h -----EEPROM Initialization and Implementation Code
 sleepMode.h -----Sleep Mode (IDLE) Initialization and Implementation
 Timer.h -----Timer initialization for 1 s timer2 and 60 s timer3

OLED CODE:

lcd_main.h-----Write to screen as well as Write Blood Sugar functions which creates bigger numbers to be used for displaying the blood sugar
 PmodOled.h-----OLED initialization and Pinouts

Bluetooth Code:

Nrf8001 (folder) main file used and modified: ble4.h

Note: The entire source code can be found on our website and the list above is by no means exhaustive. Our source code is thousands of lines of code and there are many files. The files highlighted above are the most important files in terms of understanding the high level application of each part of the code.

Appendix C. App Software Reference

C1. Programming Languages

- For Android Development
 - Java
 - XML
- For web development
 - HTML
 - CSS
 - JavaScript

C2. Integrated Development Environments (IDEs)

- Android Studio: Utilized to develop Android Applications. Includes a graphical and XML-based UI for creating each activity.
- Notepad++: Web environments were created in Notepad++, which is a smart text editor that features highlighting and recognition of common HTML, CSS, and JavaScript keywords.

C3. Miscellaneous Software

- Apache2 on Amazon Web Services to support the analytics website
- Android OS to run the application
- Application primarily tested on Samsung Galaxy S3+ and LG LTE devices

Appendix D. Electrical Parts Data Sheets

D1. Glucometer & General Parts

Microcontroller Data Sheet:

<http://ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf>

Operational Amplifiers:

For buffer circuit: <http://www.ti.com/lit/ds/symlink/opa379.pdf>

For current to voltage circuit: <http://www.farnell.com/datasheets/1834039.pdf>

For differential op-amp circuit: <http://www.ti.com/lit/ds/symlink/ina157.pdf>

Temperature Sensor:

MCP9700AT-E/TT: <http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf>

Design Reference Document:

<http://ww1.microchip.com/downloads/en/DeviceDoc/00001560A.pdf>

EEPROM:

<http://ww1.microchip.com/downloads/en/DeviceDoc/21191s.pdf>

LCD Controller:

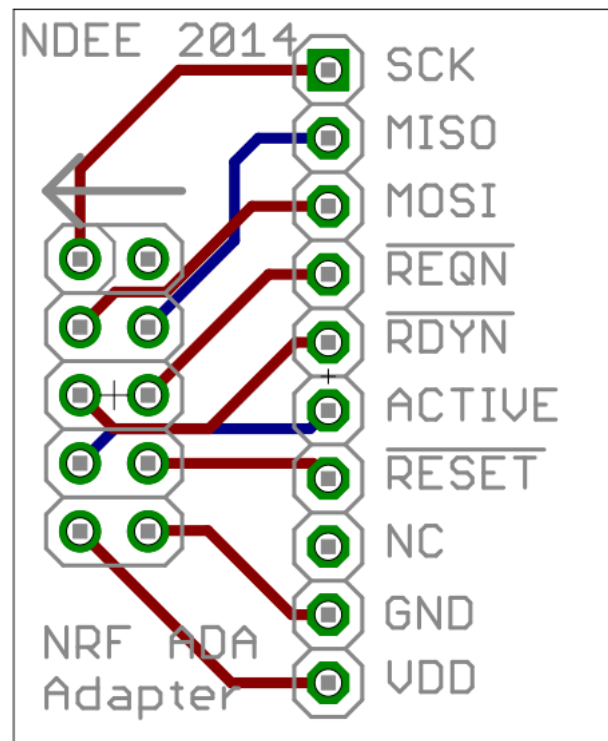
<https://www.adafruit.com/datasheets/SSD1306.pdf>

Buttons:

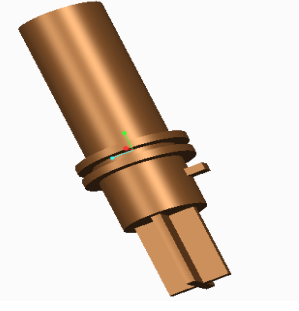
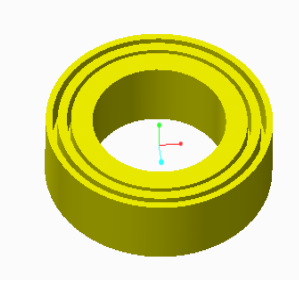
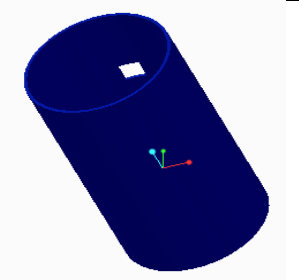
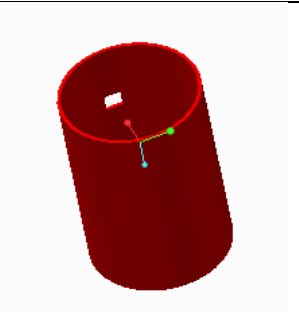
<https://cdn.sparkfun.com/datasheets/Components/Switches/ADTS6-ADTSM-KTSC6.pdf>

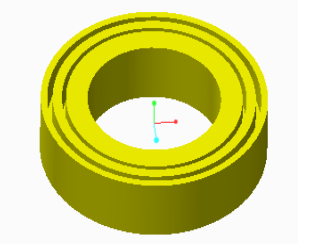
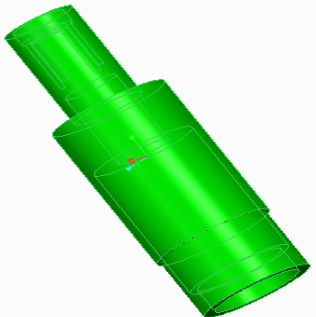
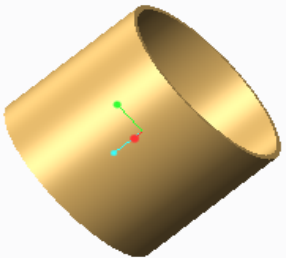
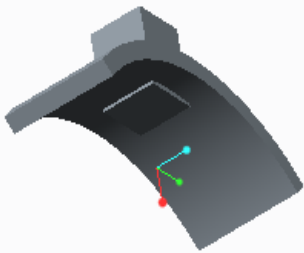
D2. Bluetooth Low Energy

Data sheets supplied to us via Sakia thus no online reference but included in our folder on our website.

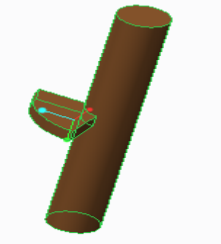
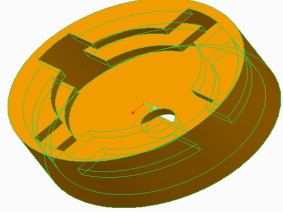
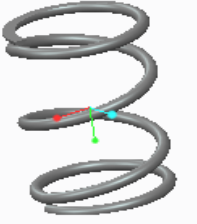
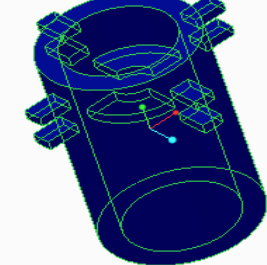
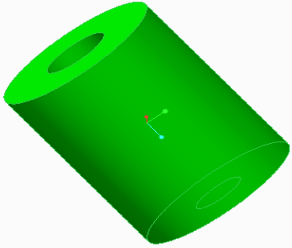


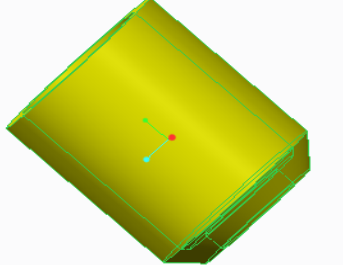
Appendix E. Mechanical Parts Reference and Data Sheets**Final Design I:**

Part Number	Schematic
i1 Puller	 A 3D schematic of a copper-colored cylindrical puller. It features a central shaft with a wider section at the bottom, a narrow groove around the middle, and a small protrusion on the side. A coordinate system with red, green, and blue axes is visible inside the part.
i2 Top Cover	 A 3D schematic of a yellow cylindrical top cover. It has a flange-like top edge and a central opening. A coordinate system with red, green, and blue axes is visible inside the part.
i4 Inner Wall	 A 3D schematic of a blue cylindrical inner wall. It is a simple hollow cylinder with a small rectangular notch on the top edge. A coordinate system with red, green, and blue axes is visible inside the part.
i5 Outer Wall	 A 3D schematic of a red cylindrical outer wall. It is a simple hollow cylinder with a small rectangular notch on the top edge. A coordinate system with red, green, and blue axes is visible inside the part.

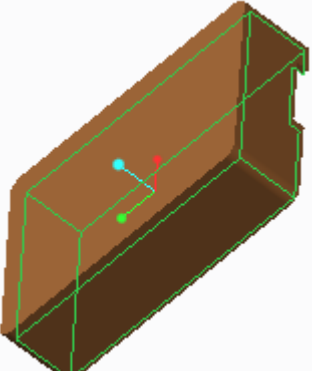
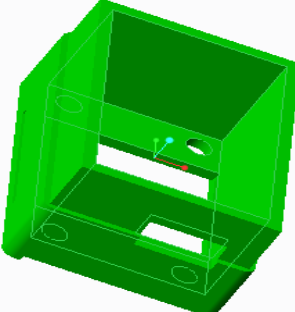
i6 Bottom Cover	
i8 Bottom Case	
i9 Cap	
i10 Trigger	

Final Design I:

Part Number	Schematic
k1 Puller	
k2 Top Cover	
k3 Spring	 http://www.mcmaster.com/#9434k76/=x1rzlk
k4 Inner Case	
k5 Lancet Holder	

k6 Outer Case	 A 3D CAD model of a yellow, rectangular outer case. The model is shown from an isometric perspective, highlighting its thickness and the top surface. A small red dot and a small blue dot are visible on the top surface, likely representing mounting points or alignment markers.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Glucometer Case

Part Number	Description
b1 Top Cover	 A 3D CAD model of a brown, rectangular top cover. The model is shown from an isometric perspective, highlighting its thickness and the top surface. A small red dot and a small blue dot are visible on the top surface, likely representing mounting points or alignment markers.
b2 Circuit Housing	 A 3D CAD model of a green, rectangular circuit housing. The model is shown from an isometric perspective, highlighting its depth and the internal structure. It features a central opening and several circular features on the sides, likely for mounting components or providing ventilation.

b3 Battery Cover

