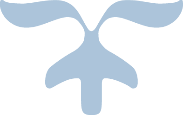


ECE 490: FINAL REPORT

Aperture Medical



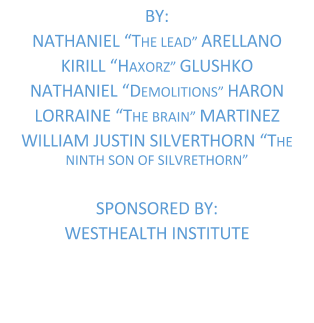


Table of Contents

[Abstract](#h.30j0zll)

[Introduction](#h.1fob9te)

Design Description and Results

[Conclusion](#h.3znysh7)

[References](#h.2et92p0)

[Appendices](#h.tyjcwt)

# Abstract

Heart disease is the leading cause of death in the United States. Heart disease is the buildup of plaques in arteries, which reduces blood flow to vital organs and increases the risk of a heart attack or stroke. If one was to suffer a heart attack or stroke and manage to survive, the life one would have to endure post attack would be very difficult, both financially and health-wise. What can one do to prevent this? By creating a self-powered, wearable, vital monitoring device, one would be able to take their own vitals into their own hands. With such a device, the user would be able to take a step in the right direction for their healthcare.

# Introduction

In the modern American medical system, a significant cost to the sick is time spent in the hospital just being observed. If that cost could be circumvented by moving that monitoring back to the patient's home, both patient and hospital could be saved considerable financial hardship, not to mention the healing power of both being in one's own bed and not hemorrhaging money. The healthy too can benefit from persistent vital monitoring like a runner who would benefit tremendously from being warned if they were close to dehydration. Imagine if every elder with a weak heart or child with a congenital illness knew as soon as they needed help, possibly even before. How much better would their lives be? How much better could all of our lives be?

The task at hand is to design a wearable device that can monitor the vital signs of its wearer and record them. The device must not interfere with the day-to-day lifestyle of the wearer, operate for up to 30 days without needing to change a battery or connect to an external power source, and communicate with an external means of viewing the data.

The design we have come up with is a light shoe, fitted with piezoelectric strips that can harvest energy from the footsteps of the wearer and removable solar panels mounted on top of the shoe that passively harvest energy from both indoor and outdoor light. An external leg-band is connected via wire to the shoe. Sensors for monitoring heart rate, temperature, hydration and blood pressure are fitted in the shoe and leg module. A near-field communication antenna is also fitted in the leg module, allowing the information to be recorded on the device and transmitted to an application on an Android smartphone.

# Body (think of better name?)

At first, the whole project seemed very challenging, especially with the aggressive timeline of three months. Although, with some organization, the project was completed. The project was broken down into a few manageable sections:

* Energy Harvesting
* Ultra Low Power Components
* Android Application
* Physical Assembly

The above mentioned items each had multiple sub sections, which will also be explored in detail later on, but first, like every engineering project, we had to adhere to some important constraints. The following list outlines those constraints:

Performance Requirements

* 24+ hour time logging
* Water resistant
* Fit in a wearable package
* 30+ days without external power source or charging
* Sensors
  + Temperature: [90℉ - 115℉ ± 0.5℉], 48 samples daily
  + Heart Rate: [10 - 300 ± 2 bpm], 96 samples daily
  + Blood Pressure:
    - Systolic: [60 - 300 ± 3 mmHg]
    - Diastolic: [10 - 150 ± 3 mmHg]
  + Hydration: normalized versus average
* Power estimates (Current total: **16 – 20 mAh per day**)
  + Sensors, Instant (<30㎲ ) :
    - Thermistor: 2.2V-3.6V @ 0.24mA ≈ 0.696mW, 2pAh per read; 96pAh total
    - Hydration: 1V @ 10mA ≈ 10mW ; 900 pAhtotal
  + Sensors, Slow (10s - 15s)
    - Heart Rate Sensor (LED + photodiode): 5V @ 4mA ≈ 25mW, 0.02mAh ; 2 mAh total
    - Blood pressure based on heart rate propagation time, added power: second heart rate sensor
  + Microprocessor (1 mAh worst case):
    - * Time on: ~25 minutes 2.2V @ 220 μA (1MHz) total
      * Time off: ~14-15minutes 2.2V @0.1 μA total

Talk about what constraints we met and didn’t meet

Energy Harvesting

One of the main focuses for this project was to harvest energy. After much research and testing we decided to go with solar and piezoelectric energy harvesting. From the research we did, we figured that these two items would allow the device to break even in its power consumption.

Solar

The photoelectrics used here are a trio of PowerFilm Solar SP3-37. These are flexible thin-film solar panels, 64mm by 36mm and only 0.22mm thick. They each typically produce about 18mA at 4v in direct sunlight, and together average about 200mW. In our design, they are mounted in parallel on a single piece of vinyl, then attached over the laces of the shoe module with conductive snaps, allowing for optimum angle of incidence for absorbing sunlight at all times of day, while also allowing simple removability without the need for wires or plugs. It should be noted that the panels also work indoors, with a measured output of about 20mW under our lab's fluorescent lighting.

Piezo

A portion of our energy budget came from mechanical energy harvesting in the heel of the foot. This was achieved by exploiting the Piezoelectric effect, a property of crystalline substances which causes electrons to be emitted when a mechanical stress is applied to the crystal.

This piezo-active elements chosen were Mide Volture Piezo Energy harvester. These were chosen for their compact design, as well as their resilience to damage and their specific energy harvesting oriented design.

Each Volture has 4 layers of Piezoactive film, when compressed can produce up to 100 Volts of instantaneous energy. This voltage, when properly rectified can be used to power devices directly, for our application, it is used to charge out power storage solution. The design uses two of these Volture elements mounted in the heel of the shoe, held in place by a custom designed 3D printed wedge.

Although Piezoelectrics produce high instantaneous voltage, they produce near zero current. After a number of attempts to design a custom piezo energy rectifier, we found a pre-manufactured IC from Linear Technologies which fit our design. The chip used was the Linear Tech LTC3588. This IC takes the raw piezo voltage on the input, with a shunt to prevent overvoltage, and outputs 3.3V, ideally, at 10mA.

The final piezo rectification solution was the LTC3588 IC mounted on a breakout board from Sparkfun. The board uses an inductor and a tantalum capacitor to keep output consistent in between steps. At an average human walking pace our power, although less than ideal, was a measurable 70µW of consistent delivery.

Power System

The power system of the G.L.A.D.O.S. Device is as follows. Energy harvesting is split between photoelectric and piezoelectric elements. The alternating current from the piezoelectrics are rectified to direct current using a specialized low-loss integrated circuit element. The D.C. Voltages from both are pushed to a supercapacitor for storage. The varying voltage across the capacitor are regulated to a fixed 3.3v rail for the rest of the system to use.

Supercapacitor

The energy generated from both sources is used to charge a Taiyo Yuden LIC1235R3R8406 Lithium Ion Supercapacitor. This part is rated at 40 farads, with a 3.8 volt maximum charge, 2 Amp maximum current and 0.15Ω terminal impedance. It comes in a 35mm long, ∅12.5mm can. A Lithium Ion Supercapacitor is a very recent development in energy storage technologies, a hybrid type package combining the strengths of two developments in capacitor theory. Electric Double-layer Capacitance is achieved much like the charge-separation in a classic electrolytic capacitor, but the electrodes are a material (in this case, active carbon for the cathode, lithium-doped graphite for the anode) that prevents the ions in the electrolyte from recombining or oxidizing on its surface, so when voltage is applied, it instead forms a Helmholtz double layer separated by a sub-nanoscopic layer of polarized solvent, storing energy electrostatically. Pseudocapacitance on the other hand is a faradaic charge that stores electrochemical energy without actually making chemical bonds or changing phase. Within the inner Helmholtz plane, desolvated ions can bind either redoxically or intercalatively (the latter with our lithium solution), to our electrodes. All supercapacitors exhibit both of these characteristics to some degree, but Lithium Ion Supercapacitor are engineered to maximize the effect of both.

We did not use a battery for a few reasons. First: batteries require extra support circuitry in order to keep them from getting overdrawn and to always charge them at the right energy levels during different parts of the charge curve. Supercapacitors are just capacitors and do not have this problem. Reason the second: any battery chemistry with the kind of energy density we needed would either have a high self-discharge rate, or low charge/discharge life cycle. Lithium supercapacitors have neither issue, with our particular model claiming less than five percent loss over three months, and a minimum hundred-thousand cycle lifespan. Third reason: power density. Capacitors have significantly higher power densities than batteries, and this holds true for supercapacitors. Our original power budget foresaw us drawing 20mA in ten second bursts, which would have burned out any single-cell battery we tried to use. Better sensors and aggressive power-saving programming techniques brought that draw to about 3mA, so this was less of an issue than it at first seemed.

The final output of the power system is fixed to 3.3 volts via a Linear Technologies LTC3525-3.3 integrated Buck-Boost power regulator. It can convert voltages between 0.9 and 5 volts, with a maximum efficiency of 95%.

# Ultra Low Power Solution

In order to efficiently use the power generated, we made sure to use components that are ULP. The power consumption was greatly cut down in the heart rate sensors and in the optimized microcontroller code.

# ***Hardware:***

Hardware design was a large part of this project, as well as intelligently choosing pre-manufactured hardware solutions for both sensor integrity and ease of implementation.

Heart Rate sensor and support circuitry

The pulse sensor we chose was a pre-manufactured pulse sensor purchased from Adafruit, the pulse sensor has some pre-amplification on board as well as some low pass filtration, the main consideration in Pulse Sensor choice was the power constraints, and the pulse sensor we used has a consumption well below a standard LED, operating as low as 0.72 mW, but for the purpose of signal strength, we operated it at 1.32 mW (3.3v @ 4mA).

The LED shinning on the skin illuminates the sub-dermal capillaries, and the light is captured by the photodiode, as blood rushes through the capillaries, the light profile changes and the photodiode captures the change, which is output as an analog signal. This signal is modified by support circuitry, in an attempt to filter noise for our use case, and then passed to the microcontroller for interpretation. This circuit branch is powered by a GPIO pin of the micro controller, and read on a separate pin. The overall system incorporates two of these sensors, both with their own identical support circuit.

The Adafruit Heart Rate sensor we chose came with amplification on board, but as the targeted location was finger tips, we found the amplification insufficient. To increase our accuracy, we implemented a support circuit using a Texas Instruments INA333 Low Power Instrumentation Amplifier and a voltage divider. The INA and voltage divider are both powered off of the branch which controls the Heart Rate Sensor. The rails of the INA are VCC (ideally 3.3 Volts) and GND. The Heart Rate Sensor is attached to the negative voltage in, and a voltage divider which outputs 50% VCC is attached to the positive pin. The gain resistor brings the gain up to 150x. Such an extreme gain is needed to read a pulse from our two target areas, the foot and the knee.

The Net effect of this support circuit is filtration of low amplitude signals, as well as forcing a heartbeat to hit the rail voltage. This creates a signal which when analyzed on an oscilloscope appears visually digital, and makes for an easy to analyses semi-noise resistant signal passed to the microcontroller.

Pulse Transit Time

As previously mentioned, two pre manufactured heart rate sensors were used. One sensor is located behind the knee and the other is located on the foot. The reason for this was so that we could measure Pulse Transit Time (PTT). PTT is the time it takes a pulse to propagate through an arterial tree. This measurement is of importance to us because if the user’s PTT readings start to deviate from their average, it can indicate a serious medical condition that should be attended to. An example of a condition that could be detected is a blood clot in the leg, since that is the location of our sensors. If the PTT readings begin to get slower, that indicates that there is something blocking the way. The user can then take steps to improve their health or take the information into a medical professional to help them improve their health.

Another interesting thing that can be done with PTT is finding the user's blood pressure. From all the research papers we looked into, we found out that a strong linear correlation can be made from Pulse Wave Velocity and blood pressure. Pulse Wave Velocity (PWV) can easily be measured from PTT by using the following equation:

Since the sensors are at fixed locations, we have the variable D, distance, easily available. Once the value for PWV is found, it's just a matter of inserting the value into a linear equation. The equation would typically be derived from doing a study in which one gathers test subjects and calculates their PWV and blood pressure. Once all these measurements are taken, they are plotted a graph and a best fit line is used to approximate the relationship between blood pressure and PWV.

Temperature

The thermistor selected was a QTI Medical Grade thermistor, it was selected for its calibration to the human temperature range, as well as its small size and low power consumption.

Because the thermistor was tuned to the human core temperature range, some support circuitry was developed to tune its range to human foot surface temperatures.

The thermistor is NTC, meaning resistance decreases as temperature increases, with a room temperature resistance of 10 KΩ, and an average foot temperature resistance of 7 KΩ. With a datasheet tolerance of +/- 0.1°F, temperature is read by taking as average of 8 readings, and will be discusses further in the embedded software section. This circuit branch is powered by a GPIO pin of the micro controller, and read on a separate pin.

Hydration

***Software:***

The software implementation of the GLADOS device required both embedded and mobile solutions, for these purposes, an Android application was developed, as well as an embedded application.

The embedded solution manages sensor polling and power management, as well as some averaging of collected values, and finally, transmission of those values to the NFC transponders EEPROM memory. The Android application was written with a Nexus 5 as the target device, it utilizes the Nexus’s antenna to capture the data logged by the embedded solution. It does some intelligent parsing of the data, and then displays it graphically for user interaction.

For the sake of simplicity, these software solutions will be discussed separately, and, due to the sheer volume of code, methods may be referenced without reproduction.  
***Embedded Solution***

Overall Design Paradigm

The main constraint of the entire project was power, and to cater to that constraint all code was written with power in mind. Every un-used pin was deactivated, and used pins when not in use were set to Hi-Z. The overall software design aimed to keep the processor awake for as little time as possible, this design relied heavily on interrupts.

The main loop has 4 tasks:

* Access Battery level
* Activate and Read sensors
* Transmit Data to NFC Transponder
* Sleep the microcontroller

The logic is that the processer should only poll sensors if there is sufficient charge to complete a read, assuming this case is true, read the sensor values, transmit those values, and then allow the processor to sleep, setting up the watchdog timer to wake after a certain time period.

Battery Polling

The energy storage device is a super capacitor, which when attached to our power delivery solution had a relatively linear voltage depletion rate. Thus, capacitor remaining voltage directly correlates to the number sensor polls the processor can achieve. The microcontroller activates the ADC on a pin which is attached to the capacitor, and the voltage across the capacitor is read. Our low voltage threshold is 1.1 Volts remaining on the capacitor, as the power delivery solution tends to shut down at 0.9 Volts. This 0.2 Volt gap allows enough energy so that the microcontroller can still poll the remaining voltage for some time without completely depleting the capacitor by activating the (relatively) high power consuming sensors. Battery polling is performed between ever sensor poll, if it is found that inadequate power remains, the processer will sleep for 15 minutes, and then wake to poll the capacitor again.

Sensor Polling

Sensors are polled semi-sequentially. To determine Pulse Transit Time and Pulse Wave Velocity, the Heart Rate sensors must be polled simultaneously, and are by far the most time/energy consuming sensor(s). After they complete their cycle, temperature is polled, and then lastly, hydration.

At the worst case, heart rate polling will take 15 seconds, consuming 5.5µWh of power. Temperature Sensing takes approximately 8µS, and consumes nanowatts of power. Hydration Sensing averages 0.3 milliseconds, and also consumes nanowatts of power.

Total sensor polling power consumption at work consumes 5.6µWh of power.

Data Transmission

After sensor data is collected, the data is transmitted to the NFC transponder. For this purpose I2C is used. I2C is a communications protocol used by embedded systems, it is relatively low power, and works by driving a high pin to ground.

In our implementation, two pull-up resistors (12.7KΩ) are driven by a GPIO pins, and are attached to the SCL and SCA pins of the I2C channel. I2C is configured to operate at 100 Khz.

At this point, it is necessary to discuss the addressing and data schema of the NFC transponder’s memory.

NFC EEPROM Memory word is 32 bits long, divided into 4 byte segments. To communicate with the NFC transponder via I2C, the MSP430 powers the pull-up resistors, and then send a start condition byte, along with the address of the transponders memory. When the MSP receives a NACK, it transmits the target write address (2 bytes long), followed by a hold (approx. 50 µS), when a NACK is received, the 4 bytes of data are written byte at a time. After the word is written, a new target address is loaded, and the process starts over.

The code is omitted here, but included in the digital copy for reference.

Texas Instruments MSP430G2553 Mixed Signal Microcontroller

The selected microcontroller, the MSP430, was chosen predominately with power constraints in mind. This microcontroller sips powers, with a sleeping current in the nano amps and the ability to completely shut down its core clock during ADC conversions. Our variant of the MSP430 was the G2553 20 pin TSSOP package. This kept the circuit design compact as well as a manageable pin out, with exactly the correct amount of pins to support all of our sensors. The MSP430G2553 boasts a 10 and 12 bit ADC, as well as a clock that can be driven down to 32KHz.

As TI’s MSP430 is an industry standard for low power embedded solutions, a large code base and community support was also a consideration, and was of great assistance during application development on the embedded side.

Heart Rate and Pulse Transit Sensing

As mentioned previously, the signal as seen by the processor is digital inappearance, and relatively easy to interpret. Heart rate sensing in code is done using the 10 bit ADC and some control conditions. A GPIO pin powers the INA333 and the Heart Rate Sensor (Please note that this is occurring on two sensors simultaneously). The master sensor, in our case the foot sensor, counts deltas of its own peaks, the slave sensor, or the knee sensor, will count deltas of its peaks to the master sensors peaks.

When both sensors are powered, a conditional keeps the ADC off until they have settled, and presales a timer to interrupt at 1 millisecond intervals. On each interrupt, two independent integers are incremented, one which times the master sensor, one which times the slave, and a conversion is begun on both sensors. Upon conversion completion, the ADC interrupt is triggered, and interprets the values read.

The schema of capturing “good” heartbeats is as such:

* After sensors have settled, begin scan after voltage goes low
* When a low is detected, at every scan, compare the value to a hard threshold
* When the hard threshold is reached, we know we will inflect soon, begin comparing every scan to previous scan.
* When scan N > N – 1, we have inflected. This is the maxima of the wave.
* Check timer count, if < 200 assume motion noise and sleep processor for 15 seconds
* Else, if time > 200, assume valid heartbeat, reset timer and begin cycle over again.

The above procedure is applied to each sensor independently. It is halted when one of two cases is met, either 8 “good” heart beats were found, or the sensor has operated for 15 seconds.

A for loop subtracts the values of the recorded slave beats from the master beats and returns the absolute value.

This loop also accumulates the transit deltas and the beats per minute deltas. If 8 valid beats occurred, the number is shifted 3 bits to left to determine an average. Otherwise, a cost division is performed to determine the same average. The transit time average and heartbeat time average are stored in a struct, and passed back to the main loop for storage until all data is collected and ready for transmission.

Pulse Transit Time

The pulse sensor we chose was a pre-manufactured pulse sensor purchased from Adafruit, the pulse sensor has some pre-amplification on board as well as some low pass filtration, the main consideration in Pulse Sensor choice was the power constraints, and the pulse sensor we used has a consumption well below a standard LED, operating as low as 0.72 mW, but for the purpose of signal strength, we operated it at 1.32 mW (3.3v @ 4mA).

The LED shinning on the skin illuminates the sub-dermal capillaries, and the light is captured by the photodiode, as blood rushes through the capillaries, the light profile changes and the photodiode captures the change, which is output as an analog signal. This signal is modified by support circuitry, in an attempt to filter noise for our use case, and then passed to the microcontroller for interpretation.

This circuit branch is powered by a GPIO pin of the micro controller, and read on a separate pin.

The overall system incorporates two of these sensors, both with their own identical support circuit.

Temperature

Temperature sensing is performed via a simple ADC conversion, the temperature is read by reading the voltage of the INA in the temperature support circuit by the ADC. 8 Consecutive readings are made of the voltage, 1µS apart, and then shift 3 bits to find the average. This raw ADC value is cast to a uint16\_t and passed to the main method for storage until all data is collected and ready for transmission.

Hydration

Although this section is titled hydration sensing, a more accurate description of the actual process is local sub dermal moisture sensing. The precision required to sense variance sub dermally is extremely high, on the order of picofarads. To achieve such precision with our small low power microcontroller, we relied heavily on hardware processes.

The MSP430 sports a multi-speed DCO, with a max frequency of 16MHz. Because our measurement, in terms of a purely engineering perspective, is capacitive, the algorithm applied is similar to a cap-sense solution, but iterative and much faster than any standard cap sense.

The setup cycle configures 2 timers, one to operate at 16 MHz, one to operate at 32 KHz. A pin is configured to act as a hardware comparator, and another is configured to act as an output of the 32 KHz. The comparator is configured to interrupt when voltage drops below 25% VCC (ideally 25% @ 3.3V = 0.825V).

The following process measures the RC of the capacitive system:

* First, the pin latched to the 32 KHz timer is activated, simultaneously, the 16 MHz timer is activated and begins accumulating in the timer accumulate register
* At 32 KHz, one period is 312.5microseconds, at 16 MHz, one period is 62.5 nanoseconds, with this in mind, one tick of the 32 KHz clock is 5000 ticks of the 16 MHz clock. We wait 5000 ticks of the 16 MHz clock, and then reset the 16 MHz timer accumulator
* At this point voltage is no longer applied to the capacitive system, and the comparator is activated
* The 16 MHz timer accumulates until the comparator threshold is reached, when the threshold is crossed, the comparator interrupt saves the value of the TAR (Timer Accumulator Register)
* This value is saved in a 16 bit integer

After 8 capacitive reads are accumulated, the number is shift 3 bits to average the values

This value is returned to the main method for storage and transmission to the NFC transponder

This relevant data to be transmitted in the number of 16 MHz ticks were taken while the capacitive system was discharging. Relatively high speed gives us a capacitance resolution of 200 femptofarads, which exceed our sensitivity requirement, allowing us to make accurate measurements.

NFC Data alignment and Timestamp Management

Data is aggregated by the processor main loop and aligned for transmission to the NFC EEPROM memory. The basic data format is:

|  |  |  |  |
| --- | --- | --- | --- |
| BPM\_High | BPM\_Low | TRANSIT TIME\_High | TRANSIT TIME\_Low |
| TEMP\_High | TEMP\_Low | HYDRO\_High | HYDRO\_Low |
| TIME | DAY | PWR | 0xFF |

This is the data that is transmitted to the NFC card over I2C. All Sensor data is stored in 16 bit integers, and all time data is stored in two 8 bit integers. The battery information is stored in an 8 bit integer.

The main loop keeps track of the time stamp, day stamp, and transmission addressed. Our NFC transponder has 0x200 (512) addressable memory locations, each write consumes 3 memory locations, and a days’ worth of data (with 15 minute read intervals) consumes 276 (92 \* 3) memory locations.

The time stamp is incremented every 15 minutes, and the data stamp is incremented every 92 time stamp incriminations. This allows us to track when the measurements are taken, and parsing/synchronization will be discussed in detail in the sections pertaining to android code.

The entire time synchronization system depends on the processor accurately sleeping in 15 minute interval and in our initial test, this proved to be quite accurate.

***Android Application***

Overall Design Paradigm

The Android application is the user’s only way of interpreting the data collected by the GLADOS device. For brevity, I will here forth refer to the Android application as AMSync (Aperture Medical Sync). When designing AMSync, code optimization was deprioritized and focus was placed on a functional and visually appealing application. The overall is application flow is as follows:

* User launches application, assuming the device is NFC enabled and NFC is on, the user is prompted to scan their leg module. If NFC is not available, the user will be informed that their device cannot run AMSync, if it is available and deactivated, they will be prompted to activate it
* Alternatively, if they wish to view historic data, the can bypass the sync phase. If no historic data exists, they will be prompted
* On sync, the application reads the entire NFC Transponder memory and begins to parse it. When parsing is complete, the user is presented with some information and choices
* The “Overview” screen shows the user the most recent sensor polling results, as well as allows the user to press on an icon, taking them to that metrics detail graph view.
* On any given metrics graph view page, by default the application will attempt to render the last 24 hours’ worth of sensor data, if 24 hours exist
* The user can narrow and expand the date/time range, of the graph window
* If the user clicks on a point in the graph, a details popup is rendered, allowing them to see the exact time stamp of a reading, as well as the values and what this value means, (Very Good / Good / OK / Bad / Very Bad), with verbiage specific to each metric

The algorithms for parsing and rendering are kept as light weight as possible, while assuring that the user will not incur an error that would cause a fatal exception or unnecessary application lag.

NFC Algorithms - Read

*For purpose of length, all code is omitted, please refer to classes prefixed “NFC” in the Android source submitted digitally.*

The general NFC reading algorithm is a modification of the code supplied by STMicro for their M24LRXXE class transponders. When an NFC handshake is detected by the dalvik runtime, it passes the data to our application’s sync activity. This uses the STM Tag identification class, and if a M24LR16E is discovered, a full read is requested. If the transponder returns with a permission to read, the phone will begin requesting blocks from GLADOS. All data that is received by the phone is held in an ArrayList pre-parsing. The data is checked for validity before parsing. If our timestamp sectors are valid, as described in the embedded solutions section, the data is marked valid. If the timestamp is missing or corrupt in anyway, the user is notified that there was an error in read, and is asked to rescan the GLADOS transponder. Assuming the data is valid, the read cycle is complete and raw data is passed to the save algorithm. The entire read operation should be bound by O(n) time.

NFC Algorithms - Save

The save algorithm is relatively basic, a new file is created to contain the data, this file name is appended by the timestamp of writing (in miliseconds), as well as the data that it will contain (NfcV or NDEF). Currently our application only supports parsing of NfcV to data.

NFC Algorithms – Parse

The parse algorithm has some complexity as the data is read in 8 bit segments, and our data alignment in the embedded code is a mixture of 16 bit and 8 bit values, shifted to optimize space. Therefore, the parse algorithm must clean and divide the data into its relevant segments. Here is a reprint of the data alignment for ease of reference:

|  |  |  |  |
| --- | --- | --- | --- |
| BPM\_High | BPM\_Low | TRANSIT TIME\_High | TRANSIT TIME\_Low |
| TEMP\_High | TEMP\_Low | HYDRO\_High | HYDRO\_Low |
| TIME | DAY | PWR | 0xFF |

When data is saved, it is stored as a flat array or 8 bit integers. The first step of parsing is sorting. Container arrays for BPM, Transit Time, Temperature, Hydration, Time, Day and Power are created. A loop strides the flat array per entire block and divides the data into these purposed arrays. After all of the data has been divided, a merge algorithm then strides each of those flat arrays individually and merges the data based on its bit width pre-division in the microcontroller, expressing all of the data as integer values.

Each data type has its own corollary equation to covert that data sent from to microcontroller into something useful. After the data is merged, as described above, all data is ran through their respective corollary equations and stored into post parsed arrays.

The equations are:

* Heart rate:
  + - 60/ (d / 1000); Where d is the milliseconds on average between peaks.
* Pulse Transit:
  + - Pulse transit has no data fixing equation, it is presented in milliseconds
* Hydration:
  + - (d > 1384 ? 100 :
    - (d > 900 ? d/96.8 + 85.7 :
    - (d > 658 ? d/48.4 + 76.4 :
    - (d > 581 ? d/15.4 + 47.3 :
    - (d > 503 ? d/3.9 - 64.0 :
    - (d > 455 ? d/2.4 - 144.6 :
    - (d > 319 ? d/6.8 - 21.9 :
    - (d > 0 ? d/12.76 : 0
    - Where d is the ticks counted by the microcontroller, this returns a values between 0% to 100% hydration
* Temperature:
  + - (0.0505 \* d) + 64.7, where d is the raw value of the ADC, returns Fahrenheit

Timestamp parsing is done based off of the initial scan, the “timesync” scan. We use the timestamp in milliseconds of that original scan to offset all time stamps, timestamp time 0 day 0 in the microcontroller is set to the first scan timestamp in milliseconds, then, all time increments are added by adding 15 minutes in milliseconds to the root time stamp. This requires scanning to occur at least once per day, or data loss will occur. To prevent double counting of data, data blocks are compared, and if they are identical, are removed from the final data returned to the graphing algorithm. After all data is prepared, it is passed in its final form to the graphing algorithm for user display and interactivity.

Graphing Algorithms – Generate and Draw

Before the graph is displayed, it is fully generated by a separate thread. The data given to the graph’s draw methods is interpreted to construct the visual. The graphing class is relatively intuitive and generating points and lines is relatively simple. A loop iterates through the data provided to the thread and finds the min and max data value, then the standard deviation. The minimum draw boundary is one standard deviation lower than the min, and maximum draw boundary one standard deviation about max; this will give the graph a smooth and compact appearance. Point objects are constructed and added to line objects, after a line object is complete, and the draw thread calculates where to draw points and lines, using Android’s canvas graphics standards. After the draw thread is complete, it pauses itself, and the canvas is passed to the main GUI thread to be displayed on screen for the user.

Graphing Algorithms – User interactivity

The user is presented with two buttons on the bottom of the graph screen to set start and end date, if these values are not set, they default to 24 hours and the current calendar date. When the user selects a start and/or end date/time, the generate and draw algorithm is re-run on the new scope of points. This can have the appearance of lag if a large amount of points are being requested, but operating in a separate thread, the phone OS remains stable. After all points are rendered and displayed, the user can also interact with the graph by touching a point.

An onTouchListener overload tracks the on screen x/y coordinate of the users touch, and every touch iterates the points displayed to determine if there is an overlap, with some acceptable margin of error, a detailed data view pop-up is displayed.

The popup has a button to dismiss it, as well as detailed information about the data point touched by the user. This information includes exact point values, exact point time and date, and a general remark about if the value is good, bad, or normal.

Physical Assembly

Even though our entire design worked on breadboards and development programmers, we would eventually need to integrate everything into a more user friendly package. In order to accomplish this, we had to prepare three things:

* 3D Printed Housing
* Fitting energy harvesting solution to the shoe
* Printed Circuit Boards (PCB)

3D Printing

All parts were designed in Solidworks with rough estimates of measurements. Each package has two parts – the back and the cover. The back is where every part is mounted. The cover is for protection and was designed in a way that would make it easy to remove and get to the parts. The package mounted to the back of the shoe only houses two boards and a SATA connector so it was designed to be much smaller than the leg package. To fit the boards, the back part had to be at least 1.5 inches high, with the diameter of the inner semicircle being 2.5 inches. The top also had to have enough space for the SATA connector to go through so we made the package 0.5 inches wide.

The leg package was designed to fit the main board, the supercapacitor, and the NFC antenna, but also had a bit more space to move components around. The back is 2.5 inches high, 0.75 inches wide, with the inner radius being 2.19 inches. A shell was designed on the back of the leg unit to slip a strap through and a window / opening was designed in the middle to help guide it. Some parts were a tight fit, but we tried to make the unit as slim as possible.

A box for a right angle connector board was also made. The connector takes a SATA connector and routes it at a 90 degree angle to right angle headers that go straight to the leg package.

Printed Circuit Boards

Shoe Assembly

The shoe selected was a Minimus HI-REZ, manufactured by New Balance. The reason for using this particular model was for the lack of sole. With an almost non-existent amount of sole, the full impact of a heel strike can be transferred to the piezoelectric strips, maximizing the power we get from the piezos. In order to get the two piezos into the shoe, we cut two holes at the rear of the shoe, wedged small grommets into the holes, and glued the two together to create a stronger bond. Once the hole was made, the piezos were able to be inserted through the hole freely. Upon placing the 3D printed wedge and finding the optimal position for the piezos in the shoe, we applied silicon adhesive to bind the two together. Lastly, we place some fabric on top of the wedge and piezo combo in order to mask the presence of the silicon adhesive. The next item to attach onto the shoe were the solar panels. The flexible solar panels were glued onto a piece of fabric, which had snap-on buttons sowed to it. The leads to the solar panels were soldered onto snap buttons. The opposite end of the snap buttons were then sown into the shoe, with two of the snap buttons having wires soldered to them to transfer the power from the solar panels. This allowed the panels to be easily removable and fully functional.

The housing for the rectifying circuitry was mounted in the rear of the shoe, where the holes were made for the piezos. The housing was held on by two screws, two washers, and two nuts. All the wires from the solar and the piezo were placed into that box.

Testing and Validation

Budget

Milestones

Creating a Gantt chart at the beginning of the project was a very challenging task, for we didn’t really know every aspect that would be involved with the project. Without knowing this information, how can we assign an importance and amount of precious time to a certain task? Well, we tried our best at breaking up the project into smaller manageable portions and giving each task an appropriate amount of time. The full Gantt chart is located in the appendix but we will first mention a few key portions of the timeline.

The ultimate goal was design day on the fifth of December, so there was a mountain of work to do before then. After our initial research work, we tried to start working on the main portions, energy harvesting, microcontroller code, sensors, in parallel. If our work went well, we were slated to have our first prototype by the end of October. Unfortunately, we had a couple of issues with software and hardware failures. This pushed back our prototype to the middle of November. The good news was that our first prototype was very close to what our final prototype was going to be. After the 3D printing was done and the PCBs were printed, we were able to start assembling the production device.

Unfortunately, the only milestones we were able to fully adhere to were the first few that involved selection components and doing research. After that, everything started going in different directions than what we imagined. During one of our meetings with Westhealth, we received some very useful advice about creating Gantt charts. We were told that when doing a project like this, it should be planned backwards. That is to say, one assumes the project is at the end stage and one has to think of the steps involved in getting to that final stage. Unfortunately, we received that advice too late and we were not able to change our Gantt chart.

# Conclusion

Through the use of new, exciting, and emerging technologies, we successfully accomplished our mission. The most amazing thing about this project is that it can actually become a household medical product one day. This is just the first step for devices such as this one. There are many improvements we would like to make if we were given the budget and the time to have another try.

One of the most important things we would need is a dedicated medical consultant to validate our assumptions and correlations. Some of us consulted medical professionals while working on the project throughout the semester but it’s not the same thing as having someone that’s as integrated to the project as we were.

Another important improvement we would make in order to make this product more marketable and user friendly would be to slim down the design. The first thing we could do to slim down the design is to try to manufacture a flexible PCB. Flexible PCBs became more available at the turn of the century. The only issue for us would be the price of the PCBs we would need printed, but thanks to their increase in popularity, prices are becoming more affordable. The next reduction would come from the supercapacitor. If we could find a supercapacitor that fits our needs and is either curved or coin sized, we would be able to fit it into our design much easier.

The next improvement would be to make a shoe conversion kit. Like we previously mentioned, we had to make many modifications to our pair of Minimus shoes. If this was to become an actual product, this would not be very user friendly. For the piezos, we could make a pocket that just slides into the heel area of the shoe. This package would then connect to the leg module via a discrete cable.

# References

asdfjkasdfj

# Appendices

askdlfksadlf