Project Document

Project Life Watch

University of Central Florida

Department of Electrical Engineering and Computer Science

Dr. Lei Wei

Senior Design II

Group 9

Josue Ortiz – Electrical Engineering

William Toledo – Electrical Engineering

Carter Lankes – Electrical Engineering

John Alcala – Electrical Engineering

Table of Contents

Figure Index

Table List

2 Executive Summary

In the recent years, wearable technology such as smart watches have become increasingly popular. Things such as the Apple Watch have even been marketed for their health and fitness capabilities. Wearable devices are quickly becoming more advanced, the demand and need for these wearables to serve in medical applications is a huge growing field in the medical industry. Embedded system design serves as a means for implementing medical sensors and solutions to outdated means of patient information systems. Gathering data from medical sensors and sending it to devices used by medical staff through modern IT systems can become a solution to old systems and become continuous around the clock monitoring system which will solve many issues.

The Project Hospital Smart Watch is a small wrist-worn embedded system which combines Wi-Fi, Bluetooth, Microcontroller, and LCD to provide heart-rate measurements, localization abilities, and patient information. We are exploring the possibility of replacing the outdated plastic hospital wristband with a small embedded system that will become the future of biomedical type devices and hopefully last for many years to come.

Hospitals are large buildings with many patients moving in and out of it's doors. Having an outdated means of identifying patients and tracking where they go to has led to many medical errors that cause unintended harm to patients. Hospitals should be a place of comfort for the patient where they can rest and recover back to full strength, not a place of worry where errors can occur and something detrimental could happen to the patient because of a minor mix up of information. The smart watch aims to help eliminate even the smallest errors in the hospital environment by allowing easy verification of the patient's information and stay there.

The device is designed to be comfortably wrist worn rectangular case enclosing the device. The LCD displays the type of information you would see on a typical hospital wristband such as name and basic patient information. A NFC tag will communicate with a nearby device to associate the device with a specific person in the IT system serving as an instant means of pulling up all important information and also the ability to modify and write information onto the smart watch in case of a last minute change or something new coming to be known. The patient wearing the device will also have his/her local location inside the hospital communicated through the IT system using Wi-Fi on the device by interacting with the Wi-Fi routers throughout the hospital structure.

The cost of the device is ideally meant to be kept within $200, with the components planned for this project we do not expect to incur an amount exceeding that due to it only being a prototype. Although this project is for academic purposes, a real-world device of this type would need to remain well within the price range we have set to be a viable option on the market.

2.1 Project Description

Patients are occasionally misidentified within the hospital. This can result in small inconveniences -- being placed in the wrong room to more serious situations where a patient can be given the wrong medication or possibly taken to have major surgeries when not necessary. According to the World Health Organization, the failure to correctly identify patients results in medication errors, transfusion errors, wrong person procedures, and the discharge of infant families [3]. A hospital should be the last place where a person or patient should feel fearful of harm due to a mistake of his information since they are entrusting their lives into the hospital's hands. This hospital mistake actually occurred to the mother of one of our team members, and if he had not been there, his mother would have potentially undergone open heart surgery when she was not supposed to. With this in mind, our team decided to design a potential solution to this problem.

2.2 Motivation

The principle motivation for this project was to provide a secondary safety net that will help patients stay safe when inside hospitals such that they are not taken to a operating room mistakenly. This desire resulted from on of group member's mother almost being mistakenly taken into open heart surgery when she was on heavy sedatives. If the member had not been present when the mix-up occurred, the results of the hospital visit could have been catastrophic. The secondary pair of eyes of an automated machine would provide a helping hand to reduce the human error of hospitals. It would also provide a better net of emergency response with the panic system, and also include medical monitoring as a stretch goal.

2.3 Goals and Objectives

This system will use a programmable RFID or NFC tag to contain the medical and identification information of the patient that will then be scanned to verify the identity of patients in the hospital. This allows for all the information to be contained inside a small chip that can constantly be updated. This also eliminates the small delay that occurs when a traditional barcode is scanned, and the information must be retreated from a database. This also allows for improvement in scanning technology.

The main chip, which will contain the RFID/NFC tag, will communicate through the hospital's WIFI network to the main computer network of the hospital. This WIFI network will be used to communicate between the hospital's databases and servers and the patients chip. With the advance of technology, this allows for this communication to be very reliable.

As a stretch goal, using the WIFI network of the hospital, the chip will constantly communicate with the servers to track the location of the patient inside the patient. With the accuracy anreliability of WIFI, a virtual fence will be used to know not only where the patient currently is but also where the patient should be. The latter of which will be implemented by communicating with the patient schedule information on the hospital's database.

Another feature included on the chip will be a panic button. This panic button will be activated by the patient when urgent help is needed. The signal will be sent to the hospital network and will alert the staff of the emergency. Of course, safe guards will be put in place to minimize false alarms or accidental button pushes since this can decrease hospital efficiency.

Additional features as a stretch goal that will be incorporated into the chip will be medical sensors that can measure heart rate and temperature. This will allow for wireless monitoring of the patient when the patient is not inside the room or if the patient for some other reason is not connected to the already present medical devices.

Since this device will be used within a hospital, cosmetic considerations will be taken into account. The chip will be designed to fit into a housing that will be small as possible, comfortable, hygienic, disposable, and water resistant. Rubber like materials will be considered for attaching the chip to the patient's arm or wrist. It will be disposable so that the same chip can be used after a patient has left. It will be water resistant so that the chip will continue to function even if the patient showers or if bodily fluids come into contact with PCB board. This will allow and increase the reliability of the device.

This device is meant to be a low-cost device as to motivate the consumers (in this case hospitals) to switch over to buy this device. The goal of this device is not to be cheaper than the already in use system; it is meant to be more secure and "smarter". The main features of patient tracking/virtual fencing and patient identification will reduce the number of minor and major mistakes and in the long run reduce the number of lawsuits stemming from these mistakes.

2.4 Function

The main function of this device is to provide an easier way to scan a patient's information compared to the- barcode method used in the majority of hospitals. With the barcode method, hospitals must invest in a scanner and the appropriate technology to display and record the information: at times this additional hardware can seem outdated and easily replaced with mobile solutions. With the advance of technology, easier solutions (such as the one presented in this report) are possible. The advantage of using an RFID chip is that it can be scanned using a variety of solutions such as tablets, mobile phones, etc. eliminating the need for extra hardware. The main goal of this device is to scan an ID onto the RFID chip and have the WIFI module download the information and then display it to the LCD screen.

Besides the functions, we will also build an app as a stretch goal that can then scan the ID on the RFID chip and download the information and display it on the apps screen. We will also try to put some medical sensors in the actual device that can monitor patient health signals.

2.4.1 Related Work

Patient identification using RFID chips already exist in some hospitals but all attempts at monitoring patient location have been through the use of RFID and Bluetooth beacons and mostly for equipment. For example, at Florida Hospital's Celebration Health facility they implemented an indoor tracking system using technology from Stanley Healthcare which is a "…RFID-based staff tracking system…" according to the information of their website. This same provider also has a patient tracking system that seems to mainly use RFID with some help of WIFI to track patients. On the contrary, we seek a solution that would not only exclusively use WIFI but also provide that extra layer of protection explained in the section above.

2.5 Specifications

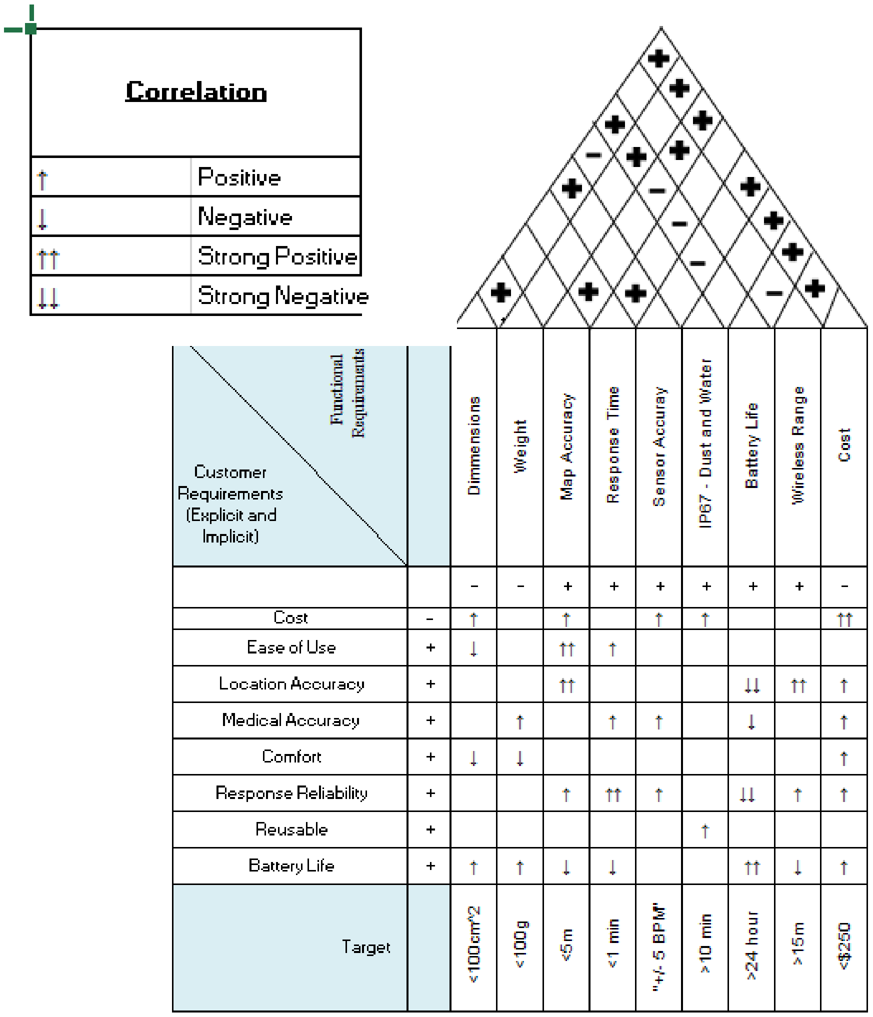
- Device will contain patient info necessary for medical staff (e.g. name, condition, medication, medication allergies, vital signs)
- Transmit this data using Wi-Fi to a tablet device or computer used by medical staff
- Device will be low power
- Low heat generation
- Battery powered (LI-ION) with a goal of at least 24 hours of continuous use out of a single charge
- Charging ability via USB to AC
- Compact and able to fit on the wrist of a patient without impacting movement of wrist or comfort
- Wi-Fi and NFC ability
- Meet HIPAA requirements
- Plastic case which encloses hardware that can be cleaned easily
- Disposable wrist strap
- Device shall not cost more than $250

Some of the stretch specifications are the following:

- Device will track patient wirelessly and provide map of loaciton
- Implement as many medical sensors as possible
- Abitlity to have emergency signal set off an alarm and communicate a warning wirelessly to hospital staff

2.6 House of Quality

The following house of quality figure allows us the designers to see the relationship between the customer wants and needs and the engineering needs. With this house of quality in mind, the goal is to find a good median point: some compromises must be made in order to achieve a good equilibrium of features. Such a graph is useful in visualizing the relationships between the two aspects. The most important of these relationships are the ones that have negative and a strong negative correlation since this is where the compromises will be made. For example, if the customer desires to monitor the location of a patient continuously, this will consume more battery life as opposed to monitoring a patient's location at intervals. On the other hand, the latter of these designs will not be as accurate as the first, but it will be more energy conservative. Of course, the main aspect affected in any change is cost: better quality parts and designs go "hand-in-hand" with a higher cost. With all of this in mind, the following house of quality table was constructed:



Correlation	
↑	Positive
↓	Negative
↑↑	Strong Positive
↓↓	Strong Negative

Customer Requirements (Explicit and Implicit) \ Functional Requirements		Dimmensions	Weight	Map Accuracy	Response Time	Sensor Accuray	IP67 - Dust and Water	Battery Life	Wireless Range	Cost
		-	-	+	+	+	+	+	+	-
Cost	-	↑		↑		↑	↑			↑↑
Ease of Use	+	↓		↑↑	↑					
Location Accuracy	+			↑↑				↓↓	↑↑	↑
Medical Accuracy	+		↑		↑	↑		↓		↑
Comfort	+	↓	↓							↑
Response Reliability	+			↑	↑↑	↑		↓↓	↑	↑
Reusable	+						↑			
Battery Life	+	↑	↑	↓	↓			↑↑	↓	↑
Target		<100cm^2	<100g	<5m	<1 min	"+/- 5 BPM"	>10 min	>24 hour	>15m	<$250

Figure 1. House of quality chart

2.7 System Overall Block Diagram

The specifications detailed in section 2.4 will now be attributed to different subsystems which will implement those functions. In this manner, an overall system can be built. The following diagram gives a brief summary of the overall system including the subsystems.

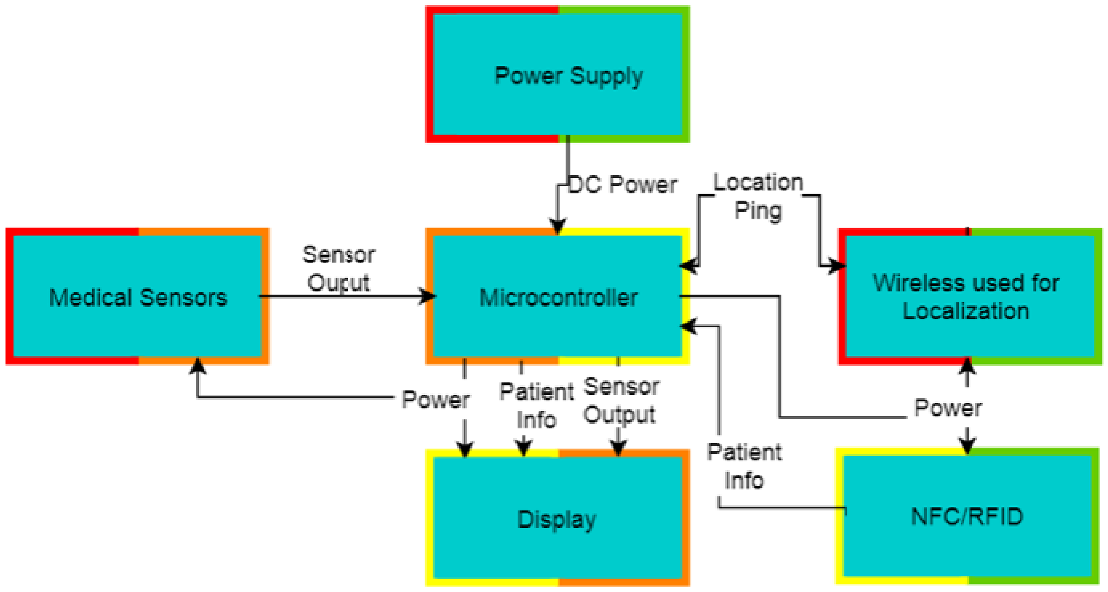


Figure 2. Overall system block diagram

The system showed above in the block diagram can be summarized into four key subsystems: 1) Microcontroller, 2) Power Supply, 3) Medical Sensors, and 4) Wireless Connectivity; and two "add-on" subsystems: 1) Display and 2) NFC/RFID. The reason the latter two have been determined by the group to be "add-on" or secondary is because their function is not vital to the overall performance of the system: they simply facilitate or enhance the system.

This overall block diagram provides the direction and orientation which the group needs to head: now that overall goals were specified and systems were made to satisfy these goals, designing these systems can now be implemented in a methodical manner.

3 Project Research

Now that the design requirements and specifications have been detailed, the first step of designing this product can begin: research. The main areas of research will be the following: medical sensors, micro-controllers, and wireless communication.

3.1 Existing and Similar Technologies

Some similar technologies and devices that we can observe, research, and learn from to help us in our project are as listed:

- Bluetooth enabled Heart-rate monitors
- Pedometers and fitness bracelets such as the fit bit
- Life-alert emergency communication/panic alert
- Indoor Localization with Wi-Fi
- The apple watch
- Bluetooth beacons
- NFC in mobile devices
- RFID tags and scanners

3.2 Relevant Technologies

In this portion we will see that there are many technologies to consider for our smart hospital watch. The technologies we will be researching is wireless location, RFID and NFC to read and write information, medical sensors, and more. The options are practically limitless to what we can choose, which is great because we will have the options to research and compare what will be best for us.

3.2.1 Wireless Communication

There are two main communication functions that the device will need to accomplish: 1) Communicate patient information to a scanner, 2) Communicate patient location and information to a computer network. RFID or NFC will be used to communicate patient information directly from the device to the nurse or staff member scanning the device, and WIFI will be used to update the information on the RFID or NFC chip. WIFI will also be used to implement the location tracking and to communicate any medical information from the sensors to staff electronic devices. By using WIFI, the cost of this medical system is greatly reduced since WIFI is already widely used in many areas of the hospital: this system will work on the already established WIFI network of the hospital. This does increase the risk of interference, but in the event that WIFI alone cannot satisfy the design specifications and requirements, Bluetooth and/or RFID technology will be considered in order to meet the consumer's expectations. Of course as in all things, this addition of technology will be done so sparingly because of the objective of also achieving an affordable product that is not expensive.

3.2.1.1 WIFI

Since WIFI is not a new technology, the inner workings and different protocols of WIFI communications will be ignored; only relevant information will be presented here. In large industrial or commercial areas (such as a hospital) different types of hardware are installed around the hospital to make sure that the wireless network connection covers the whole building. In the most usual and simplest case, this comes in the form of access points abbreviated as APs. Access points are “networking devices that allow wireless WIFI devices to connect to a wired network…. [It] acts as a central transmitter and receiver of wireless radio signals” [4]. These access points each have its own unique media access control address (MAC) and also assign an individual MAC address for each independent WIFI network it is used to transmit and receive. This MAC address is also known as the BSSID for a specific wireless network [5]. The figure below helps demonstrate this function.

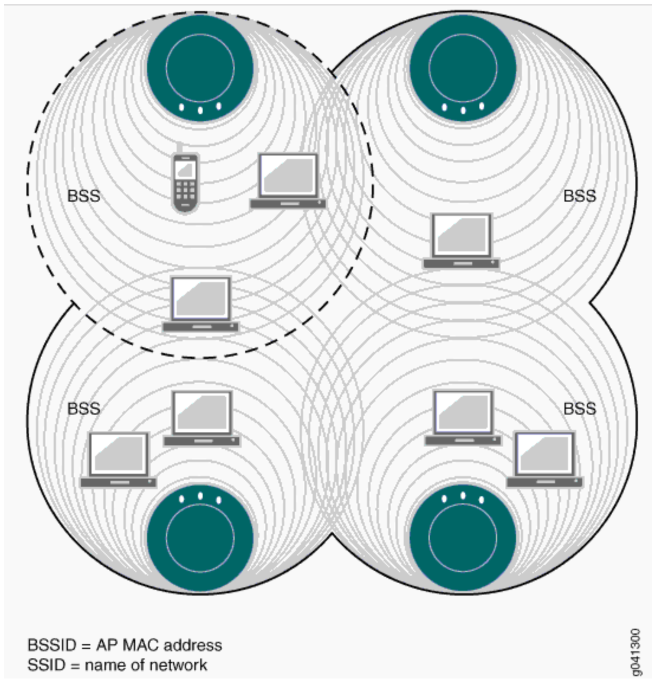
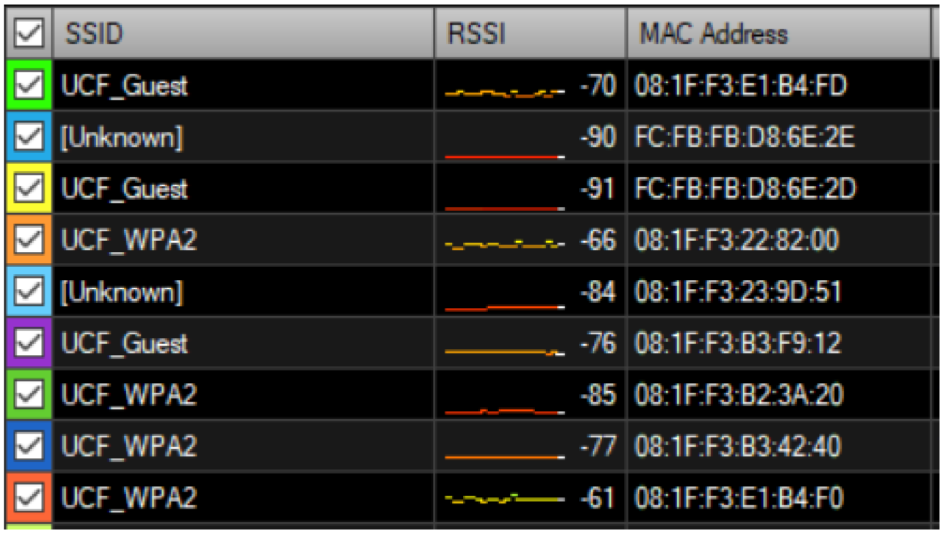


Figure 3. Access points depiction

In Figure 2, all of the devices can be considered connected to the same wireless network (they are all receiving the same signal), but depending on their location, they are connected to different access points shown by the green circles; and each green circle has a unique ID “tag” called the MAC address or BSSID.

This all leads up to the end goal of indoor localization. Indoor localization using WIFI is generally achieved by the use of two data “points”: 1) Received signal strength and 2) Access point MAC address. Much has already been investigated, tested, and researched on empirical equations and different methods that predict distance from the access point depending on the RSS (received signal strength), but that will not be presented here. The basic concept behind it is as follows: first, a WIFI enabled device must be able to detect the strength of the different WIFI signals it is receiving from various APs. Then, by comparing the received signal strength to the transmitted strength (this data is usually known by AP product specifications) for each AP, a relationship can be determined between signal attenuation and distance. Since the location of the AP is fixed and known, the location of the mobile device can also be known and attributed to a "real" place as well.

As mentioned above, this can only happen since each AP will have a different MAC address than the others; therefore, each signal received by the WIFI device can be distinguished and attributed to the correct access point. A screenshot from a software program called inSSIDer demonstrates this perfectly [5]. This program is a free, online software program that provides useful information of the WIFI networks that your device is detecting. Here part of the screenshot taken.



✓	SSID	RSSI	MAC Address
✓	UCF_Guest	-70	08:1F:F3:E1:B4:FD
✓	[Unknown]	-90	FC:FB:FB:D8:6E:2E
✓	UCF_Guest	-91	FC:FB:FB:D8:6E:2D
✓	UCF_WPA2	-66	08:1F:F3:22:82:00
✓	[Unknown]	-84	08:1F:F3:23:9D:51
✓	UCF_Guest	-76	08:1F:F3:B3:F9:12
✓	UCF_WPA2	-85	08:1F:F3:B2:3A:20
✓	UCF_WPA2	-77	08:1F:F3:B3:42:40
✓	UCF_WPA2	-61	08:1F:F3:E1:B4:F0

Figure 4. inSSIDer screen capture

Figure 4 shows that the SSID UCF Guest (name of the network) repeats throughout the list but is associated with different MAC addresses which correspond to different APs and displayed is also the received signal strength in decibels. The software program designed for this project will need to accomplish a similar feat.

The other duty WIFI will be used to fulfill is communicating information from the device to other hospital devices where its stores patient information. This communication must be bi-directional: initial patient information is synced to the device over WIFI, and updated patient information (sensor information and location) must be sent from the device to the hospital's devices. The devices will all be connected to the same wireless-local-area network enabling them to send information to the AP they are connected to, and then the AP will send the data to the correct destination device.

3.2.1.2 WIFI Chip Options

To perform the function stated above or any WIFI connectivity, our design needs to include a WIFI module. As this project is about designing a product, the team will not be looking to make their own WIFI chip but instead look for one that is already made and includes the firmware to operate it. The most popular chips considered in this project will be the Texas Instrument WIFI connectivity modules and the ESP WIFI modules as well.

3.2.1.2.1 CC32xx Wireless MCUs

Based on familiarity, the WIFI modules from Texas Instrument were the first to be considered since some classes in the electrical engineering track involved using and getting to understand the MSP430 micro-controller and the programming code and syntax developed for TI systems. The CC32xx Wireless MCUs modules and ICs are the chip solutions TI has developed for projects or problems dealing with wireless connectivity. They are incredibly capable chips hosting two physically separate MCUs: one that is composed of an ARM Cortex application processor with 256KB of RAM and a second network processor MCU that runs the "WIFI and Internet logical layers" stated on the datasheet.

These families of chips include attracting features such as the following: advanced low power modes (1 µA, 5 µA), DC/DC converter, SW IP protection, 4-channel, 12-bit Analog-to-Digital Converters, and much more. This kind of chip is self-sufficient since it already includes a built in micro-controller unit (MCU) which can process other information from our design such as the sensors or RFID tag. All in all, this module could serve as an almost complete solution to our design, making the search for a micro-controller almost unnecessary. The cost of the most basic module (without any memory expansion) is listed as $17.08 [6]

3.2.1.2.2 ESP 8266 WIFI Modules

The other popular candidate found in many forums and project websites online was the ESP 8266 WIFI modules and the expansion modules that are based off this module. This module also includes general digital I/O pins, memory, and a low power 32-bit micro-CPU.

This device operates on 3.3 V and cannot step down the voltage from a higher, standard 5 V. To interface this with the microcontroller, this action will need to be handled by a separate part of the design. A feature to keep in mind is that in sleep mode, this module consumes less than 12 µA. The cost of this module before tax and shipping is listed as $6.95 [7].

The other expansion modules include the ESP8266 as part of a chip that includes an MCU much like the TI modules explained above such as the ESP8266-12E. This module contains a 80 MHz microcontroller, memory, and I/O ports.

3.2.1.2.3 ESP WROOOM 2

The ESP WRoom 2 WIFI module is another WIFI module under consideration. This WIFI module is 802.11b/g/n, but does not include IPv6 network protocol; this will have to be considered when making the final decision on WIFI modules. This WIFI module has an operating voltage range of 2.7V - 3.6V. Its average operating current is 80 mA. This WIFI module contains 16 GPIO (general purpose Input/Output) pins some which are reserved for controlling the UART.

This module also has 3 low power modes which can be implemented. Of these, the power mode that we would likely use is called “Light-Sleep” and it consumes an average of 0.9 mA. This low power mode is achieved by suspending the operation of the CPU until needed while maintaining WIFI connectivity. This is key for us since this implies that the chip will be constantly monitoring the WIFI signals and occasionally the CUP can be woken up to process the data and transmit it. Besides this it also includes a crystal oscillator, a PWM, and a built-in antenna.

3.2.1.3 Bluetooth

Like WIFI technology, Bluetooth is a wireless communication technology that uses radio waves and a standard protocol to communicate. Also just like WIFI, it is a fairly recent technology that was released just two decades ago (WIFI was released a little less than 3 decades ago). The advantage that Bluetooth has over WIFI is that Bluetooth connects or “pairs” to devices directly whereas WIFI simply provides the means of connecting a device wirelessly to a network (access point, router) but not directly to a device. Although Bluetooth and WIFI can operate in a similar frequency (2.4 GHz), when placed together, there is hardly any interference from one to the other.

This makes Bluetooth technology a viable option to implement on this project; adding Bluetooth to the project can further expand the possibilities and capabilities of this device since Bluetooth beacons can be placed around the hospital and provide countless applications of a more integrated environment. Besides this, Bluetooth can also provide a means by which our device can pair with another medical device if any Bluetooth medical device is ever made or implemented in the near future. An example of this would be the pairing of a nurse’s mobile platform (smart phone or tablet) with another Bluetooth enabled device inside the patient’s room that can let the nurse know the latest updates on her patient. The diagram below provides a good visual of this when compared to the WIFI figure (Figure 2).

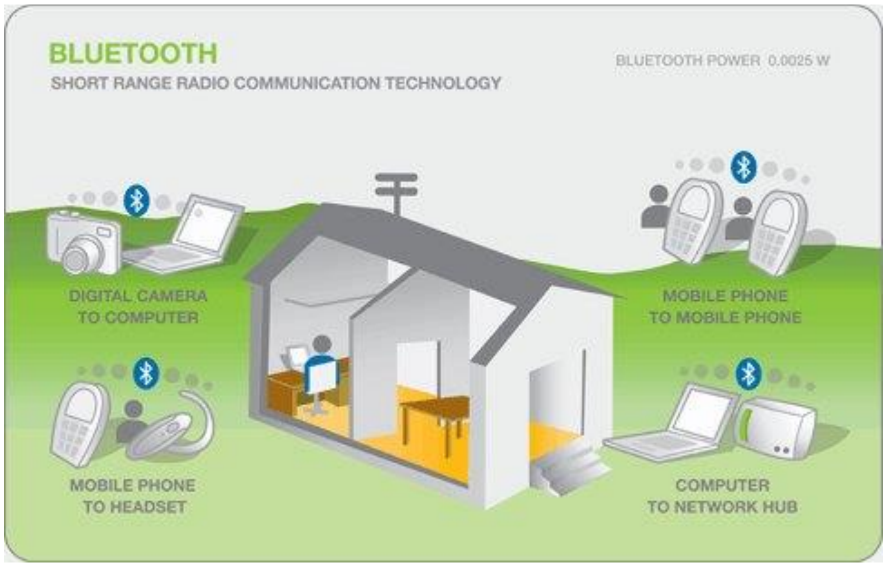

BLUETOOTH
SHORT RANGE RADIO COMMUNICATION TECHNOLOGY
BLUETOOTH POWER 0.0025 W
DIGITAL CAMERA TO COMPUTER
MOBILE PHONE TO MOBILE PHONE
MOBILE PHONE TO HEADSET
COMPUTER TO NETWORK HUB

Figure 5. Bluetooth technology [8]

This allows Bluetooth to be used for a unique kind of indoor localization than WIFI through the use of Bluetooth beacons. "Bluetooth beacons are no more than small Bluetooth devices that broadcast a radio signal…called an "advertisement" and follows standard Bluetooth protocols…" [9]. These beacons can be placed strategically throughout the hospital (depending on the quality, signal transmission strength, etc.). Once the beacons are placed, any Bluetooth device that pairs with the beacons will receive the transmission signals from the beacons and calculations of the received signal strength indicator (RSSI) can be used to determine the position, like the method using WIFI. The advantage that Bluetooth presents is that it is not very prone to interference by WIFI signals. The challenges of using Bluetooth are cost and its ability to only transmit radios waves for short ranges reliably (although there are specifications that say that certain Bluetooth technologies can transmit about 100 m, but this will need to be tested), therefore in a large setting like a hospital, a big quantity of Bluetooth beacons will need to be purchased to implement a reliable system.

3.2.1.4 Bluetooth Beacons

An initial search on the internet seem to show that Bluetooth technology would not be hard nor costly to implement: the consensus was that Bluetooth beacons were cheap, but the contrary has been the result of this project research. Most Bluetooth beacons from established technology companies sell at a price $20 to $30 each: this is for the basic and cheapest ones. Right off the bat, these Bluetooth beacons are "disqualified" from our parts choosing since a major design specification was that the product must be affordable, and the total cost could not exceed $250 dollars.

For an adequate demonstration, at least 3-4 of these beacons would need to be bought and this would take up about a third of the cost. This price for these beacons would also make it hard to scale for the target consumer which is the hospital. For this reason, the group is considering buying Bluetooth beacons from foreign suppliers which can be acquired for about $3 to $6 dollars. The disadvantage with trying to buy from foreign suppliers is that many times it is not properly documented (missing datasheet) and is sometimes not compliant with FCC standards and requirements. Despite all this, an attempt will be made since the other standard Bluetooth beacons are too expensive for this project.

3.2.1.4.1 Shenzhen Sato Intelligent Technology

The cost-efficient Bluetooth beacons were found on the global trade site name Alibaba whose market focus is based in China. On this website we found an attractive Bluetooth beacons produced by the company named above. Its range varies from 22-300 meters depending on the use; it uses the NRF51822/52832 chipset, it is a Bluetooth 4.0 low-energy iBeacon, and most importantly it is certified by CE and FCC. This Bluetooth module already comes ready to use in its own external casing and its broadcasting information (UUID, Major, Minor values) can all be modified. The cost of this device must be negotiated from $3.15-$13.50.

3.2.1.4.2 DSD Tech HM-10 BLE Module

Another option is to use a Bluetooth module that can transmit radio signals and build our own Bluetooth modules. This HM-10 module is a four-pin module that will serve as a master device (the main device other devices connect to). It operates using Bluetooth 4.0. Its working voltage is from 3.6 V to 6 V; its power consumption in the active state is 8.5 mA which is a great feature since this is low power. This module is made by DSD Technology and the module is based on the third-party chips and the datasheet is not specific on which exact chip is uses. This module will first need to be configured using its serial ports to edit the information it broadcast. Its TX and RX pins will be used. After the information has been edited, the module will broadcast the information independent of a micro-controller, only a power supply is needed. Several projects have used this chip to perform similar projects involving Bluetooth beacons. Other modules like the HM-11 were considered but found not to be needed as the only advantage it offers is a chip “enable” pin and a chip “state” pin which are not needed in this project since the Bluetooth beacons will always be on. This module cost $10 before shipping and taxes.

3.2.1.5 Bluetooth Modules

Not only would Bluetooth modules need to be used to make beacons but also to give our project Bluetooth capabilities. The HM-10 module discussed above can both be configured to transmit and to receive. This module can also be considered as a Bluetooth module to place on the chip.

3.2.1.5.1 ESP-WROOM-32

This module is a combination chip of both Bluetooth and WIFI capabilities. Its operating frequency range is 2.4-2.5 GHz. Its Bluetooth protocol is v4.2 and BLE and WIFI protocol supports 802.11 b/g/n with WIFI network protocols including IPv4 and IPv6. Its operating voltage is between 2.7V and 3.6V with an average operating current of 80 mA, but this fluctuates depending on the programming mode that the chip is set. This chip contains 38 pins. What is most important about this device is the different power modes it has. Its power modes are like the ESP-WROOM-2. The challenge will be how often we wish to "listen" or receive Bluetooth and WIFI signals constantly to increase accuracy, the current consumption becomes 95 to 100 mA: this is a lot of power consumption.

3.2.1.5.2 RN4020

The RN4020 Bluetooth module is manufactured by Microchip Technology. Below is a block diagram of the chip.

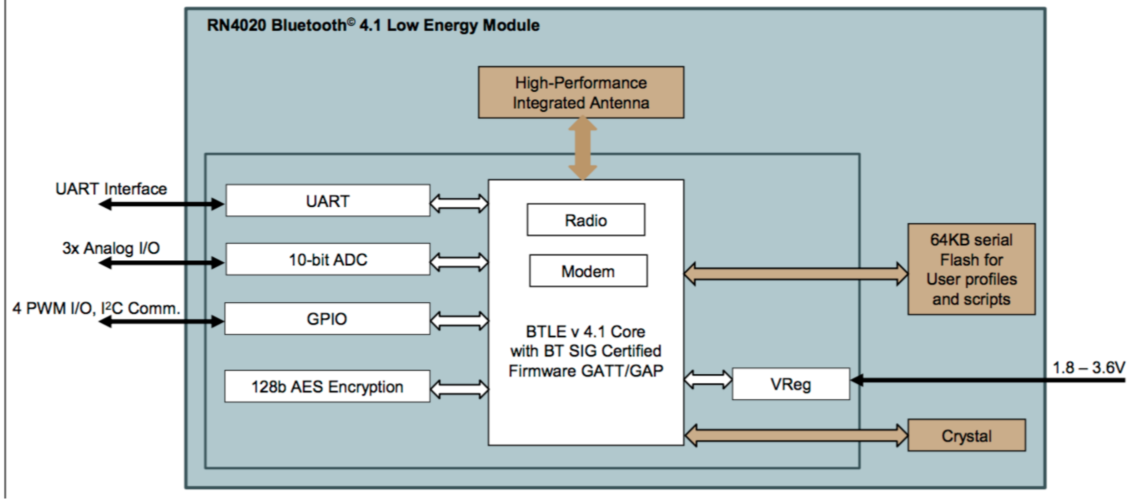

RN4020 Bluetooth© 4.1 Low Energy Module
High-Performance Integrated Antenna
UART Interface
3x Analog I/O
4 PWM I/O, I²C Comm.
UART
10-bit ADC
GPIO
128b AES Encryption
Radio
Modem
BTLE v 4.1 Core with BT SIG Certified Firmware GATT/GAP
64KB serial Flash for User profiles and scripts
VReg
1.8 – 3.6V
Crystal

Figure 6. RN4020 block diagram [10]

This is a Bluetooth 4.1 chip that is also backwards compatible with previous Bluetooth versions such as Bluetooth 4.0 (like the Bluetooth beacons mentioned above). It does have I^{2}C compatibility as well as the UART interface. Additional benefits of using this module is that it has its own voltage regulator, 64 kB flash memory, and antenna. Other chips found did not include antennas or included antennas that protruded from the module; this modules antenna is "on-board." According to the datasheet, the typical current at 3V while have the TX/RX active is 16 mA. The operation range is 100 m under ideal conditions. This chip also includes GPIO and an ADC converter, but these features have yet to prove under what circumstance they would be useful. The cost of this module before tax and shipping is $10.60.

3.2.1.6 Radio Frequency Identification (RFID)

The RFID chip or tag, as stated before, will be used to communicate the patient's information by scanning it and then updating the information over WIFI. RFID is not new technology, but there are a few types and variations to look at. For us we will most likely be deciding between the RFID being an active or passive unit. We will also need to decide either to use Low frequency or High frequency depending on our needed distance of scanning the chip. Encrypting the RFID will be necessary as well due to the standards of operating in a hospital setting.

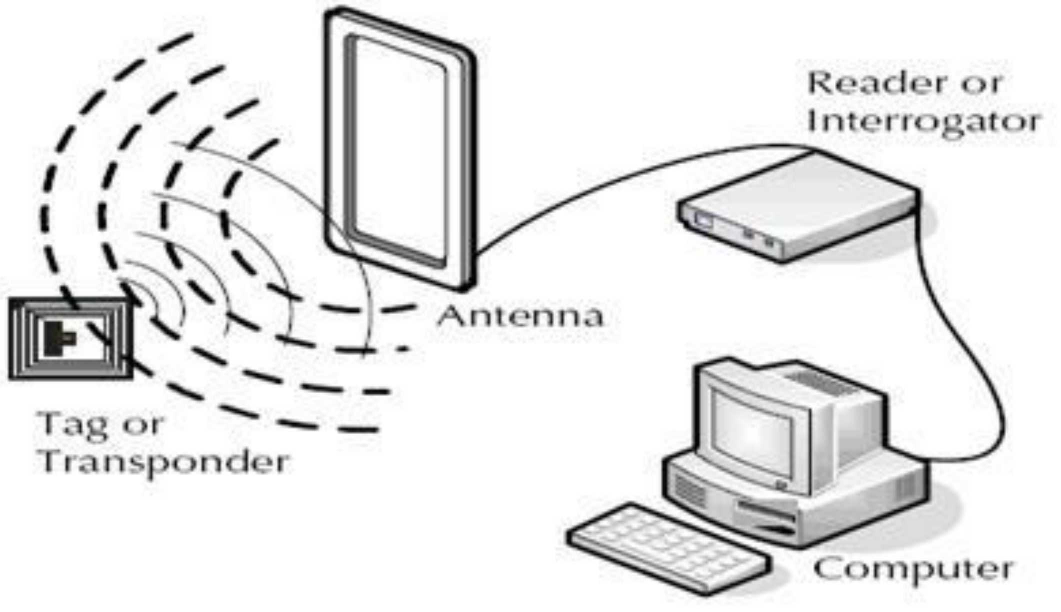


Figure 7. RFID technology

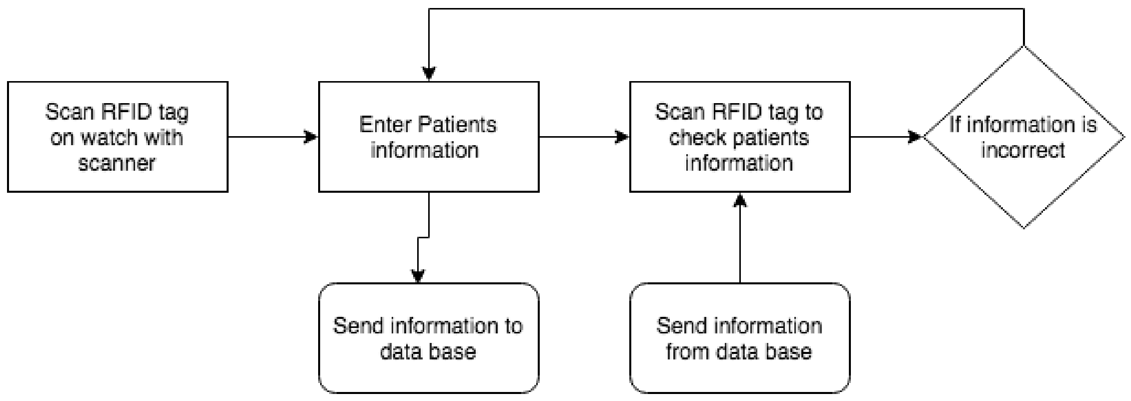

Scan RFID tag on watch with scanner
Enter Patients information
Scan RFID tag to check patients information
If information is incorrect
Send information to data base
Send information from data base

Figure 8. RFID Smart watch overview process

3.2.1.7 RFID chip options

The two main options for RFID tags are either passive or active. They both do the basic requirements of what we need, but active could be used along with our location service for precision on accuracy.

3.2.1.7.1 Passive RFID

The passive RFID seems to be the best suitable type for our project. It doesn't require an internal battery because it uses the RFID readers energy to power itself, very cheap, durable and flexible, small, and comes in multiple varieties of tag options. The passive RFID can also operate at either a Low Frequency 125-134 KHZ for a 1 to 10 centimeters range or High Frequency of 13.56 MHz for about 1 meter of range.

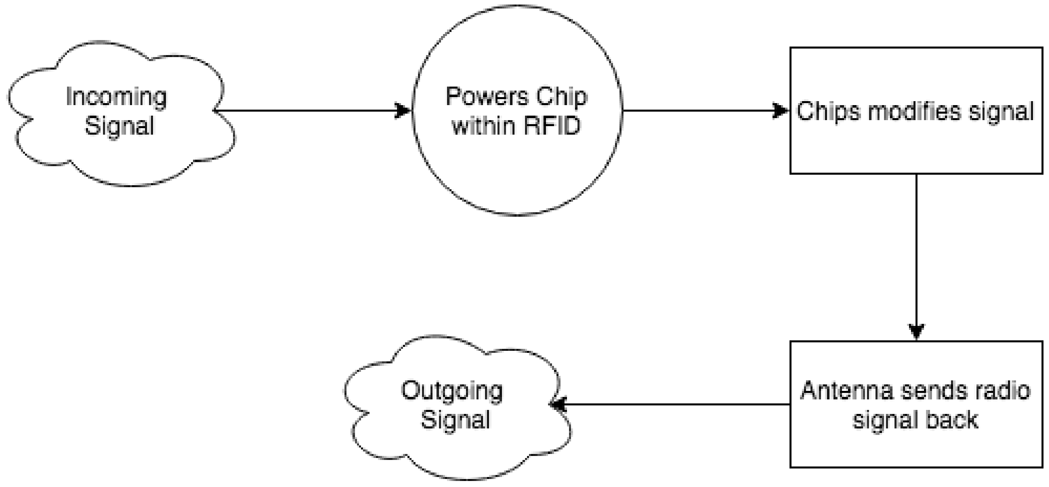


Figure 9. Passive RFID

3.2.1.7.2 Passive RFID Options

As for what Passive RFID transponder to use we will look at a few and decide which has the best durability, cost, size, frequency range, product life, and security.

3.2.1.7.2.1 Passive RFID Model AT88RF04C-MX1GA

A possible RFID transponder to use would be the AT88RF04C-MX1G. This transponder offers many useful features that will work great with the smart watch and in the hospital setting. It operates at a frequency of 13.56 MHz, it is encrypted to secure the patients information, compliant with industry standards, flexible for comfort and small in size. It is good that the signal does not go to far because there is less chance of patient information theft. This RFID transponder meets all of the requirements necessary for our device.

3.2.1.7.2.2 RFID Model LXMSJZNCMF-198

This RFID transponder is quite different from the previous option. It is a lot smaller (1.25 mm x 1.25 mm x 0.55 mm) and cheaper at $0.96 a piece. This transponder also sends 96 bits, which should be enough since the patient information isn't too much data and operates at 865 MHz to 920 MHz.

3.2.1.7.2.3 Passive HF MIFARE Classic EV1 1K RFID Tags

These passive RFID tags are another great option for all aspects of the smartwatch especially on the cost of being only $0.08 - 0.29. They are made of paper, which could be an issue for liquids being spilt, but we could probably build a water proof casing. It operates at the 13.56 MHz and has a range of about 1-10 centimeters. It is also small in size at 45 by 45 mm. It also has a data endurance of 10 years and write endurance of 100,000 times. [12]

3.2.1.7.2.4 3M Glue Waterproof NFC Tag RFID Sticker

This passive RFID is another top option to choose from. It is only $0.05-1 a piece, can operate at 125 Khz/13.56Mhz/860-960Mhz, range of 1-10 centimeters, data retention of 10 or more years, and is also small in size (26 x 42mm). This RFID also has the ability to be encrypted, but one of the best features is that it is waterproof

3.2.1.7.2.5 RFID transponder model: M24SR64-YDW6T/2

This passive RFID transponder is quite different from the previous ones because this one can directly connect to our board and send data to our microprocessor. It operates at 13.56 frequency and has a memory size of 64 kbit. The price is $1.86, which is not too bad. It also has the It also has the ability of I2C serial interface, requiring 2.7V to 5.5V, which will help depending on what microcontroller we choose. The data retention is very good as well being 200 years and up to 1 million writes depending on the temperature [12].

3.2.1.7.3 Active RFID

Active RFID is another possible option to use for storing the patient's information. Compared to passive it can store more information, assist with location services because it is always sending a signal out, and have a longer range of signal using 433MHz or 915MHz. Although, the downside of active RFID is that it is usually more expensive, and bulkier than passive.

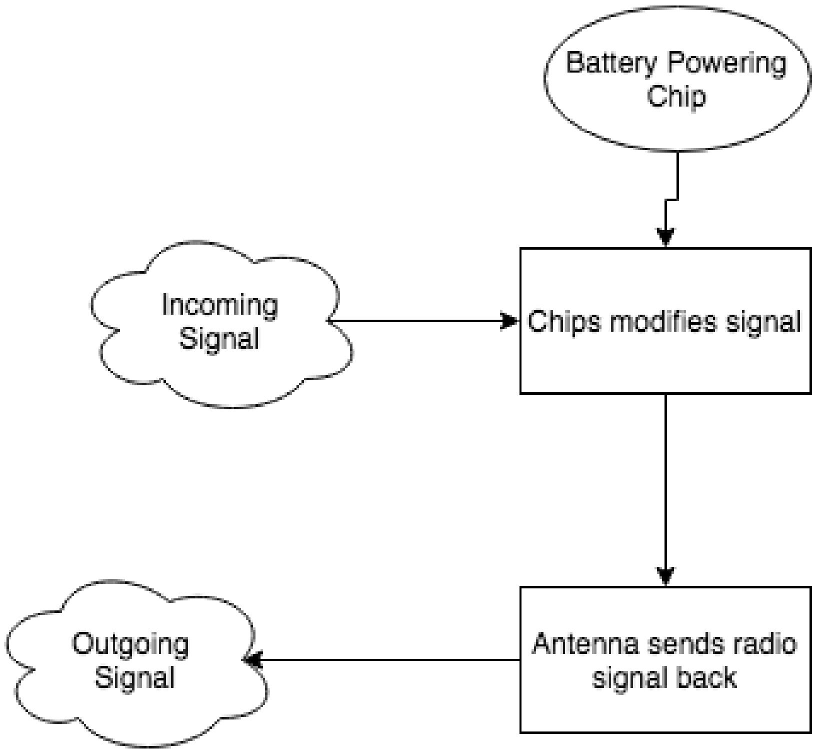


Figure 10. Active RFID

3.2.1.7.4 Active RFID Options

As for what Active RFID transponder to use we will look at a few and decide which has the best durability, cost, size, frequency range, product life, and security.

3.2.1.7.4.1 Active RFID Model: Model: RF430CL330H

A possible active RFID transponder we could use is the model RF430CL330H. This part also operates at 13.56 MHz and has an internal battery and can use SPI or I2C interface to connect the device, which is great for our use of microcontrollers. It only requires a 3.3V power supply, which our possible power supplies are easily capable of [13].

3.2.1.7.4.2 RFID Comparison

There are many RFID and NFC devices available in the market. Based on our research, the following table summarizes the different specifications and characteristics used to compore the various RFID or NFC known devices that were found as a result of our research:

Table 1. RFID Comparison

Type	**AT88RF04C-MX1GA**	**LXMSJZNCMF-198**	**Passive EV1 1K RFID tag**	**Waterproof NFC Tag RFID Sticker**	**M24SR64-YDW6T/2**	**RF430CL330H**
Active or Passive	Passive	Passive	Passive	Passive	Active	Active
Voltage	0V	0V	0V	0V	2.7-5.5V	3.3V
Frequency	13.56MHz	865-920MHz	13.56MHz	125kHz/13.56MHz/860-960Mhz	13.56MHz	13.56MHz
Range	1-8cm	1-7cm	1-10cm	1-10cm	1-8cm	1-12cm
Price	$1.56	$0.96	$0.08-0.29	$0.05-1	$1.86	$1.48

Figure 11. Different Types of RFID in order

3.2.1.7.4.3 Picking an RFID

The RFID/NFC component we chose to use for our overall project was the RF430CL330H because it has the most useful features for our device. It operates within range of our power supply at 3.3V, is small in size, uses the standard frequency of 13.56MHz. The extra special features of it too are that it can directly connect to our microcontroller and share data with I2C or SPI. It can also be written or read, has Bluetooth pairing capabilities which could be useful for the future if we decide to pair with a mobile device. It also has 3KB of SRAM for NDEF messages, which is plenty for our needs. The chip overall will be easy to test because the pins stick out for breadboard use. Overall this chip was definitely the best candidate for our project.

3.2.1.7.4.4 Breakout Board

When choosing the RF430CL330H it is important to also purchase a breakout board to be able to test the component out on a breadboard. Without a breakout board the component is too small to test and is hard to verify it's actual capabilities. For us we purchased the same package type of TSSOP 14 to be able to use the 14 pins on the chip. The breakout board we purchased was PA0033.

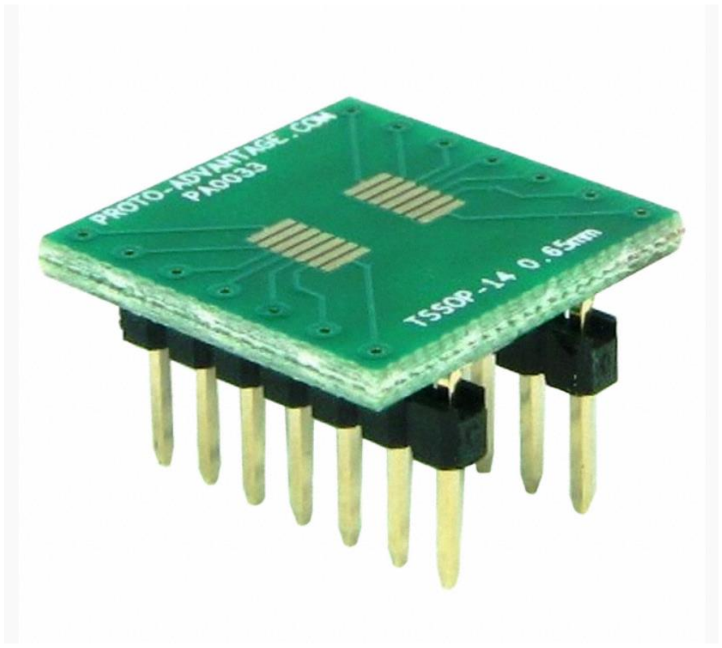


Figure 12. RFID Breakout Board

3.2.1.8 RFID Scanner

Along with the RFID chip/tag we will need a scanner device to intake and transmit the patient's information to the database. We can either buy or build this depending on our limitations and budget. There is a decently cheap scanner that works with the Arduino.

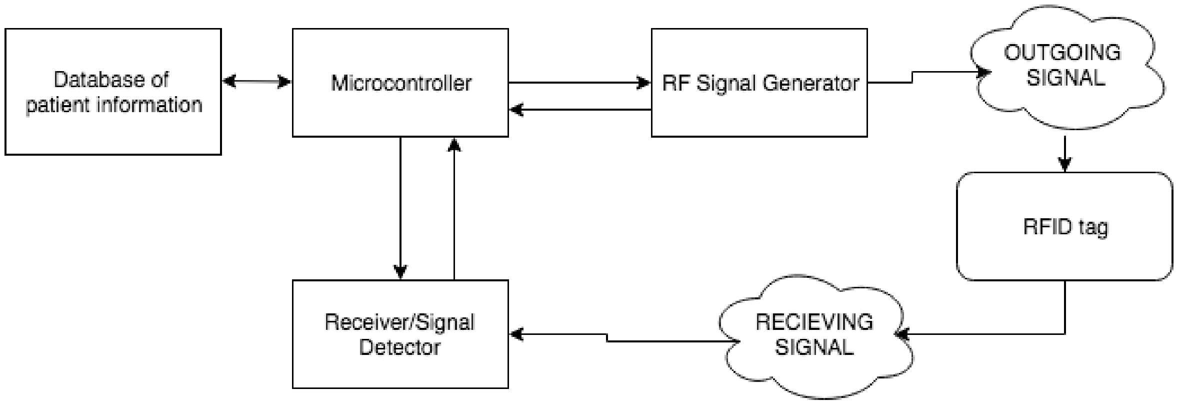


Figure 13. RFID Scanner/Reader Overview

3.2.2 Near Field Communication (NFC)

Another option for storing and transmitting the patient's information would be to use near field communication (NFC). NFC is very similar to RFID, but with this you can read and write data, where RFID is read only. They both operate at 13.56Mhz, but NFC still requires close up interaction. Compared to RFID though NFC is designed to be more secure for data exchange and capable of being a reader and tag. Allows peer-to-peer communication which is great for checking and entering patient information. NFC devices are also able to read passive HF RFID tags that are compliant with ISO 15693 [15].

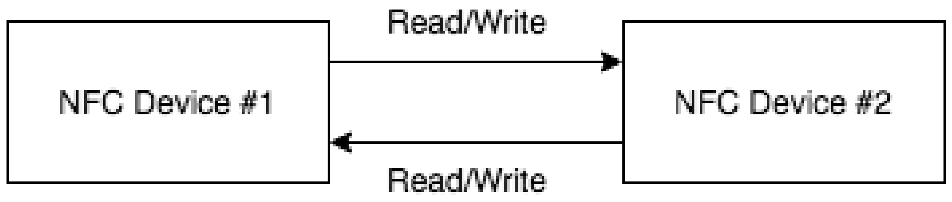


Figure 14. NFC General Overview

3.2.2.1 Using NFC with mobile device

In today's times practically all Mobile phones have NFC chips built into them for use of different types of applications, which is great for us. This will allow us to be able to use our mobile device as an NFC reader instead of having to go out and make or buy one. This will also greatly help in the aspect of making a simple app to use for our device because we can use a common style of coding and not start from scratch. We will have a head start for making the app that will display and allow the user to enter information of the patient over NFC capabilities. The NFC will work great with the RFID/NFC chip, RF430CL330H, we have chosen as well because it specifies that it is very compatible with smart phones for this purpose.

Figure 15. NFC Tag [16].

3.2.3 Power Supply, Regulation, and Recharging

For power supply we have a few options to choose from, but we are mainly looking for a battery that is good for low power products, long durations, rechargeable, and small in size. The best battery for this seems to be a lithium ion battery because it is mainly used in all small electronics today that do similar applications to what we are looking for. Depending on which sensors we choose overall will mainly determine which battery we want to use, but if it can power everything and last at least a day it will meet the criteria.

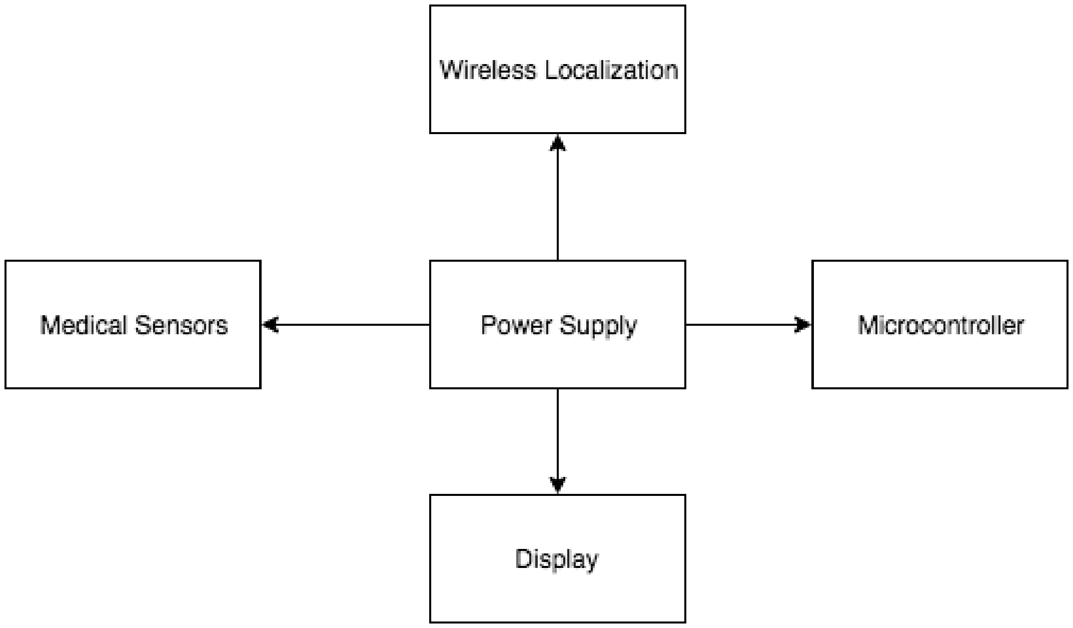


Figure 16. Power supply general diagram

3.2.3.1 Power Supply Tech

There are many power storage devices available for electronics. Some of the more common include batteries and capacitors, while non-storage solutions include solar, motion, and even thermal. In our case, the best option would be a battery technology, sor its long term storage, higher capacity, and nonvolatility.

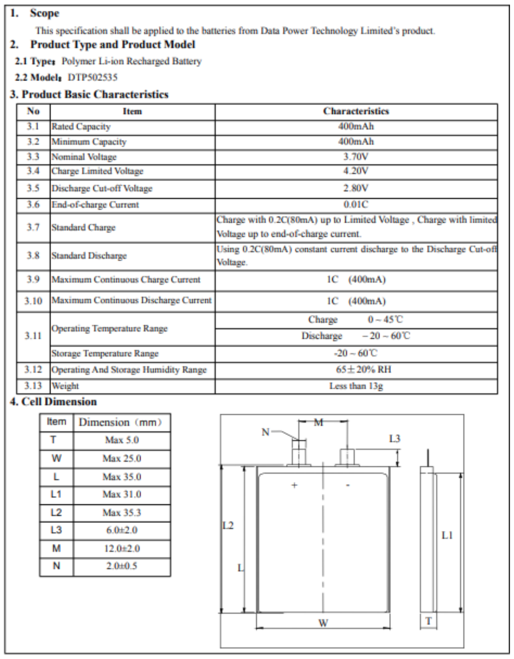
3.2.3.1.1 Power Supply Options

Now we will explore the possible options to be used as a power supply that will meet the expected requirements for our sensors, size, and duration of use.

3.2.3.1.1.1 Polymer Li-ion Model: DTP502535, 3.7V/400mAh

A possible lithium battery we could use is the DTP502535 model. It can operate up to 3.7 volts where most of the sensors require between 3.1V to 5.25V and the micro controllers operate at about 3V, although the MAX30101 could not work due to the LEDs requiring 5 volts. The size of this battery is also quite small, 1.04" L x 1.45" W x 0.20" H (26.5mm x 36.9mm x 5.0mm) [16] which meets the size requirement and can easily connect or disconnect with the jst-phr-2 connector. This will allow the consumer to recharge the battery at ease. It can also operate at 400mAH which should be good for the MSP430s operating at about 0.5mA or the ATmega328PB operating at about 1.4mA. This battery is very durable as well passing vibration, bump, drop, and other tests, which is good for the smart watch due to the potential of it being knocked around a bit. The price is only $4.95 and it is rechargeable, overall this battery is a good option.

Table 2. Electrical Characteristics of Battery DTP502535



1. Scope

This specification shall be applied to the batteries from Data Power Technology Limited's product.

2. Product Type and Product Model

2.1 Type: Polymer Li-ion Recharged Battery

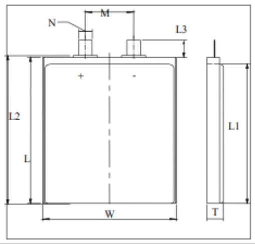
2.2 Model: DTP502535

3. Product Basic Characteristics

No	Item	Characteristics
3.1	Rated Capacity	400mAh
3.2	Minimum Capacity	400mAh
3.3	Nominal Voltage	3.70V
3.4	Charge Limited Voltage	4.20V
3.5	Discharge Cut-off Voltage	2.80V
3.6	End-of-charge Current	0.01C
3.7	Standard Charge	Charge with 0.2C(80mA) up to Limited Voltage , Charge with limited Voltage up to end-of-charge current.
3.8	Standard Discharge	Using 0.2C(80mA) constant current discharge to the Discharge Cut-off Voltage.
3.9	Maximum Continuous Charge Current	1C (400mA)
3.10	Maximum Continuous Discharge Current	1C (400mA)
3.11	Operating Temperature Range	Charge 0～45℃
		Discharge －20～60℃
	Storage Temperature Range	-20～60℃
3.12	Operating And Storage Humidity Range	65±20% RH
3.13	Weight	Less than 13g

4. Cell Dimension

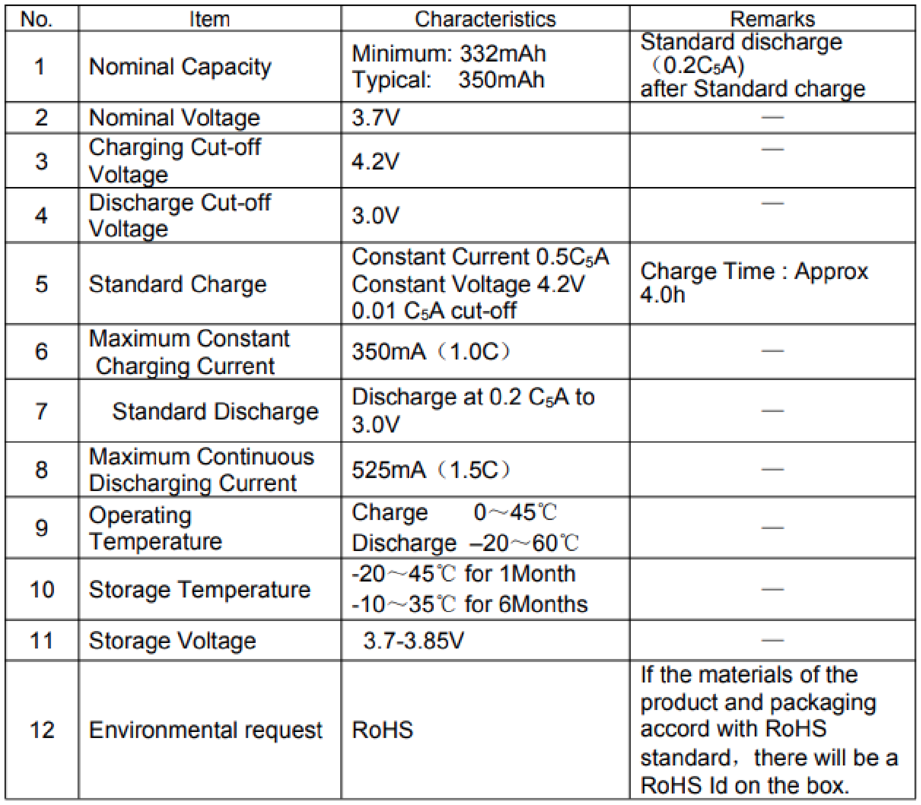
Item	Dimension（mm）
T	Max 5.0
W	Max 25.0
L	Max 35.0
L1	Max 31.0
L2	Max 35.3
L3	6.0±2.0
M	12.0±2.0
N	2.0±0.5



3.2.3.1.1.2 Li-Polymer Battery Model:552035 3.7V/350mAh

This battery is very similar to the previous one, DTP502535, although it is smaller in size,1.42" L x 0.79" W x 0.22" H (36.0mm x 20.0mm x 5.6mm), which is better for comfort and space on the smart watch. The cons are that it only operates at 350 mAh and is more expensive at $6.95.

Table 3. Electrical Characteristics 552035



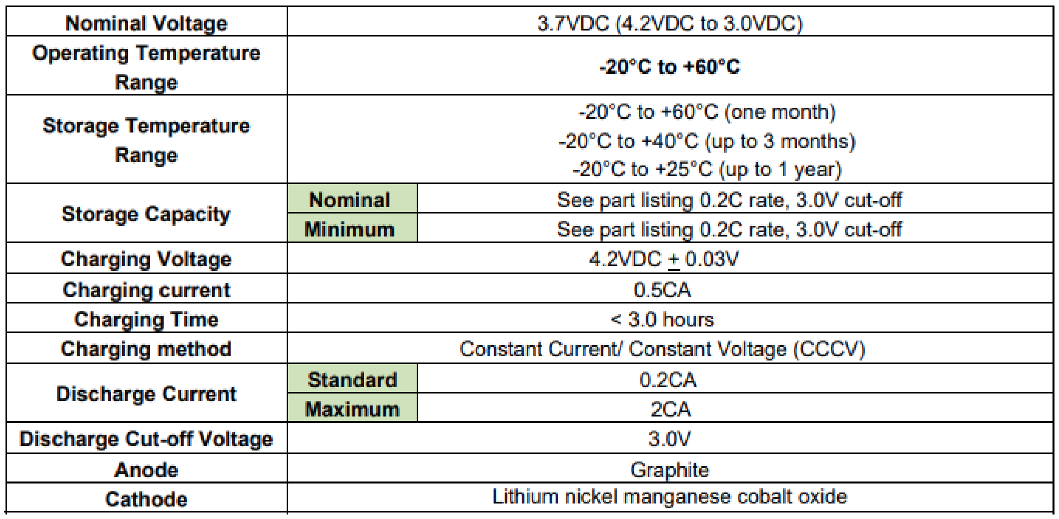
No.	Item	Characteristics	Remarks
1	Nominal Capacity	Minimum: 332mAh Typical: 350mAh	Standard discharge （0.2C_5A) after Standard charge
2	Nominal Voltage	3.7V	—
3	Charging Cut-off Voltage	4.2V	—
4	Discharge Cut-off Voltage	3.0V	—
5	Standard Charge	Constant Current 0.5C_5A Constant Voltage 4.2V 0.01 C_5A cut-off	Charge Time : Approx 4.0h
6	Maximum Constant Charging Current	350mA（1.0C）	—
7	Standard Discharge	Discharge at 0.2 C_5A to 3.0V	—
8	Maximum Continuous Discharging Current	525mA（1.5C）	—
9	Operating Temperature	Charge 0～45℃ Discharge –20～60℃	—
10	Storage Temperature	-20～45℃ for 1Month -10～35℃ for 6Months	—
11	Storage Voltage	3.7-3.85V	—
12	Environmental request	RoHS	If the materials of the product and packaging accord with RoHS standard，there will be a RoHS Id on the box.

Figure 17. PKCELL LIPO 552035 [17]

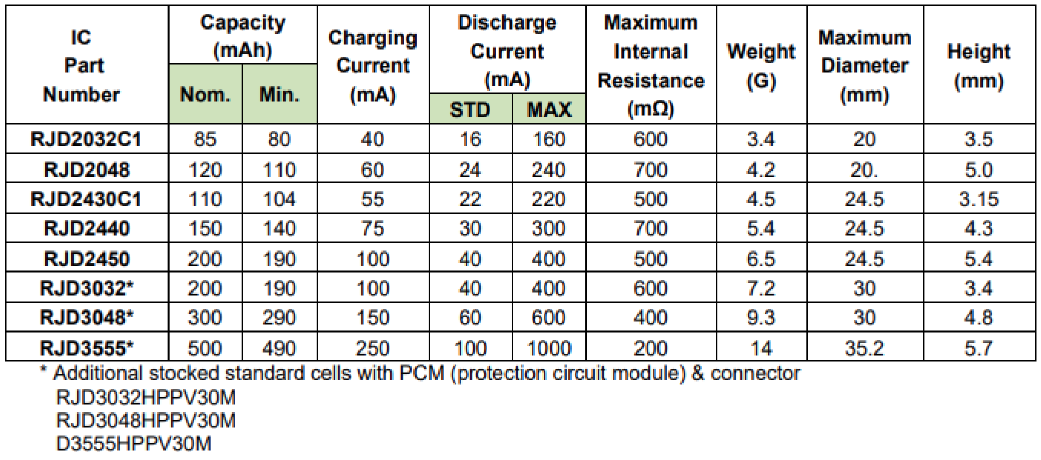
3.2.3.1.1.3 Model: RJD3555HPPV30M 3.7V/500mAh

Another battery type which would be very compatible with the overall smart watch design is the coin battery. This battery is very small, 1.39" Dia x 0.22" H (35.2mm x 5.7mm), very powerful, operating at 500mAh, and easy connectivity. Although the biggest con is the price which is $34.48. This battery definitely meets all the requirements best, but the cost may be too much.

Table 4. Electrical Characteristics RJD3555HPPV30M [18]



Nominal Voltage		3.7VDC (4.2VDC to 3.0VDC)
Operating Temperature Range		**-20°C to +60°C**
Storage Temperature Range		-20°C to +60°C (one month) -20°C to +40°C (up to 3 months) -20°C to +25°C (up to 1 year)
Storage Capacity	**Nominal**	See part listing 0.2C rate, 3.0V cut-off
	Minimum	See part listing 0.2C rate, 3.0V cut-off
Charging Voltage		4.2VDC + 0.03V
Charging current		0.5CA
Charging Time		< 3.0 hours
Charging method		Constant Current/ Constant Voltage (CCCV)
Discharge Current	**Standard**	0.2CA
	Maximum	2CA
Discharge Cut-off Voltage		3.0V
Anode		Graphite
Cathode		Lithium nickel manganese cobalt oxide



IC Part Number	**Capacity (mAh)**		**Charging Current (mA)**	**Discharge Current (mA)**		**Maximum Internal Resistance (mΩ)**	**Weight (G)**	**Maximum Diameter (mm)**	**Height (mm)**
	Nom.	**Min.**		**STD**	**MAX**				
RJD2032C1	85	80	40	16	160	600	3.4	20	3.5
RJD2048	120	110	60	24	240	700	4.2	20.	5.0
RJD2430C1	110	104	55	22	220	500	4.5	24.5	3.15
RJD2440	150	140	75	30	300	700	5.4	24.5	4.3
RJD2450	200	190	100	40	400	500	6.5	24.5	5.4
RJD3032*	200	190	100	40	400	600	7.2	30	3.4
RJD3048*	300	290	150	60	600	400	9.3	30	4.8
RJD3555*	500	490	250	100	1000	200	14	35.2	5.7

* Additional stocked standard cells with PCM (protection circuit module) & connector
RJD3032HPPV30M
RJD3048HPPV30M
D3555HPPV30M

3.2.3.1.1.4 Lithium Ion RJD3048HPPV30M 3.7V/300mAh

This battery is exactly like the model above. The pros to this one is that its smaller, 1.18" Dia x 0.19" H (30.0mm x 4.8mm) and less expensive at $17.13, although the con is that it only operates at 300 mAh. Refer to tables above.

3.2.3.1.1.5 18650 Battery Lithium-ion

Our last option for a power supply is quite different from the other options. This power supply is the cylindrical shaped standard battery and is by far the biggest, but our expected size of our prototype device is 100mm x 100mm and this meets those requirements coming out to be 18mm x 65mm. It is 3.7V and has the best supply at 3000mAh, which is fantastic for our device because this will allow it to last a very long time especially if we can manage it to be low power and efficient. This battery is also rechargeable and a good price for a bundle of 4 of them coming out to be $11.99. They can be recharged up to 1200 times.

3.2.3.1.1.6 Power Supply Comparison

Now we will compare our options of power supplies to see which will be best to choose from. There are many give and takes for each type, but this chart will help clarify in an overall view.

Table 5. Power Supply Comparison Chart

Model	**502535**	**552035**	**RJD3555H PPV30M**	**RJD3048H PPV30M**	**8650 Battery Lithium-ion**
Voltage	3.7V	3.7V	3.7V	3.7V	3.7V
Current	400mAh	350mAh	500mAh	300mAh	3000mAh
Size	26.5mm x 36.9mm x 5.0mm	36.0mm x 20.0mm x 5.6mm	35.2mm x 5.7mm	30.0mm x 4.8mm	18mm x 65mm
Cost	$4.95	$6.95	$34.48	$17.73	$11.99
Type	Block	Block	Coin	Coin	Cylinder

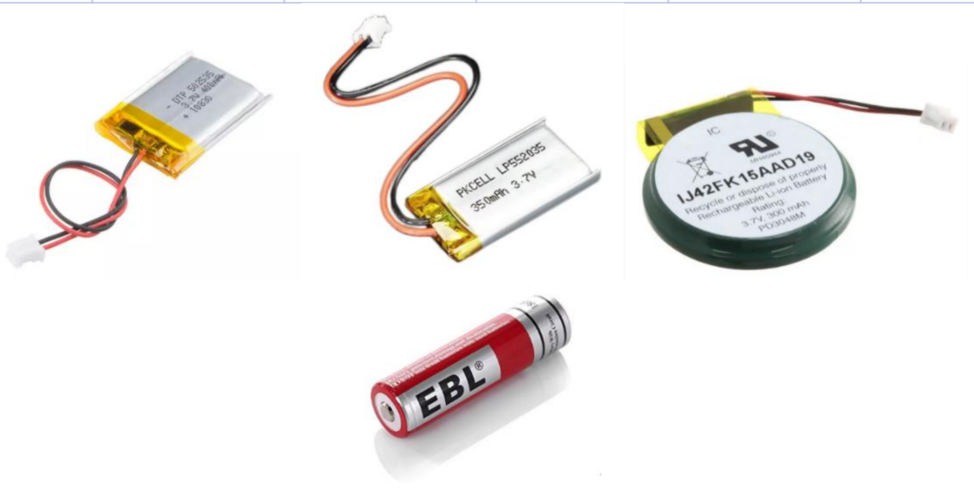


Figure 18. Different Types of Power Supplies

3.2.3.1.1.7 Picking a Power Supply

We have many options to choose from for a power supply, but the one that will be best for our prototype design will be the 18650 Lithium ion battery because it contains the most power. We are not fully sure what our power consumption will exactly come out to be, so it is best to go above what we need then exactly what we need at the moment. The main con to this power supply is definitely the size, but if need be we can probably find a smaller cylindrical battery easily. The price for this battery is not bad either considering it is a four pack, and if there are issues with batteries or we are not satisfied there is a warranty on them. As for now though this is our power supply and it will be nice having 3000mAh to be able to use for our overall device.

3.2.3.2 Power Distribution (Regulation)

One of the most important aspects of this project will center around power: power distribution and power efficiency. These topics are a common theme in all electronics: the more efficient the device is the better. Due to all the different modules and sensors involved in this project, careful consideration will be taken. The following sub-sections will detail the devices or methods use to ensure efficiency and adequate power distribution.

3.2.3.2.1 Voltage Regulators

Due to the sensors requiring different voltages and currents many voltage regulators will be needed to meet these requirements. We will evaluate the possible voltage regulators that could be used for the smart watch.

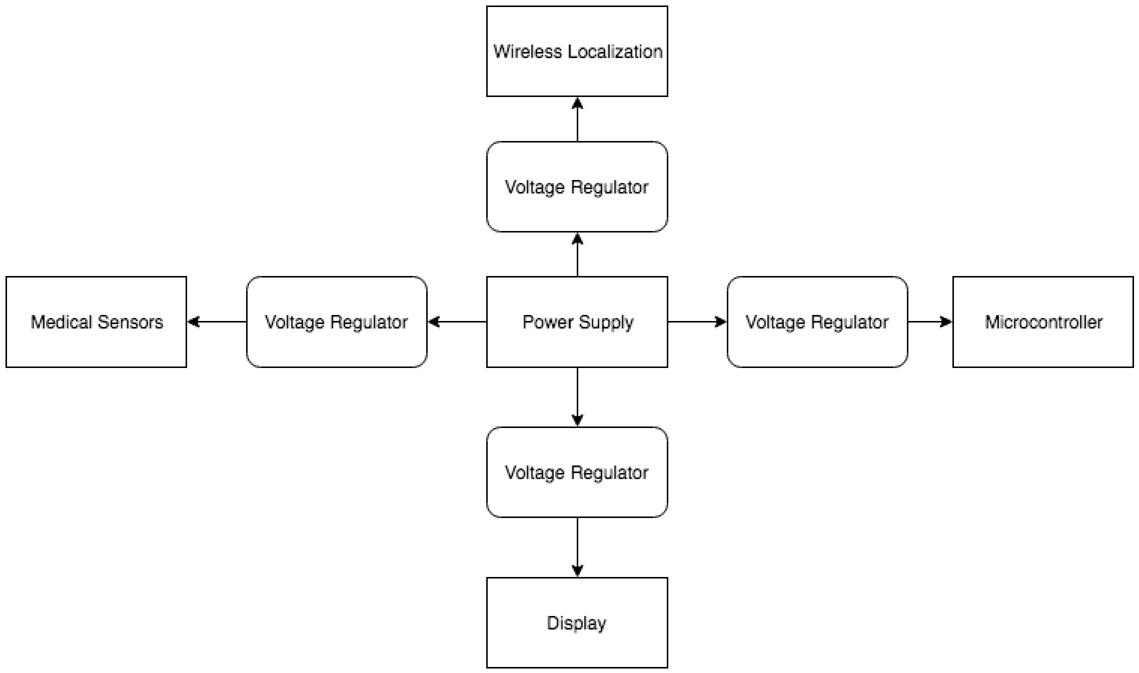


Figure 19. Overall Diagram of Power Distribution with Voltage Regulators

3.2.3.2.2 LM1084IT-ADJ/NOPB

This voltage regulator has the ability to output voltage from 1.2V to 15V which is in our range of use. Most of the sensors operate at about 1V to 3V and the microcontroller operates at around 3 volts as well. Its input voltage is minimum 2.6V and is a through hole mounting style. It is also decently small and priced for $2.59. [20]

3.2.3.2.3 TPS63036

Another option for a voltage regulator is the TPS63036 High Efficient Single Inductor Buck-Boost Converter with 1-A Switches. We were able to find this highly efficient DC to DC converter from a similar device that used a lot of the same functions we are using. Input voltage range is 1.8V to 5.5V and can output 1.2V to 5.5V which meets all of our requirements. It is also very compact in size being 1.854 mm x 1.076 mm and efficiency up to 94% which is great. Another pro is that the data sheet even suggests using it for personal medical products and LEDs. It even has a power save mode and overtemperature protection all for just $1.71. This will most likely be the chosen voltage regulator because the amount of pros are enormous.

3.2.3.2.4 Comparing Voltage Regulators

Model	Input (Volts)	Output (Volts)	Size (mm)	Cost
LM1084IT-ADJ/NOPB	2.6V - 29V	1.2V - 15V	10.18 x 8.41	$2.59
TPS63036	1.8-5.5V	1.2V - 5.5V	1.854 x 1.076	$1.71

Figure 20. Types of Voltage Regulators in order

3.2.3.2.5 Choosing a Voltage Regulator

As predicted we are going with the model TPS63036 because it was highly recommended by Texas Instruments for a device they made that used sensors and similar qualities to what we are trying to do. The efficiency is great for low power since we are trying to make our device last as long as possible. It also outputs a voltage necessary for everything we have listed and should be able to power everything well. Plus it helps when the data sheet suggests that is recommended for personal medical devices and powering LEDs. This voltage regulator will be a great addition to our overall device and provide us with exactly what we need.

3.2.3.3 Power Supply Charging

Mobile devices have a constant cycle of surviving on battery and being recharged. In order to keep the longevity of the battery, we must assure that the battery is being charged correctly. Some of the important characteristics of correct charging are the rate at which the device is charged, the current used to charge the battery, and the temperature result of the battery.

There are a few methods at which the rate of charging the battery affects. There is the slow charge and the fast charge. The key difference is that the slow charge uses “trickle” method of charging that allows the battery to safely charge without damaging the battery, and is very dependent on the chemistry of the battery. A slow charge for a Ni-Cd is defined as having a charging current equal to 10% of the Ah rating of the battery. However some fast charge Ni-Cd batteries can use up to 33% of the Ah rating. Ni-MH batteries on the other hand cannot handle continuous charging as much so the slow charging rate can range from 2.5% and the 10% of the Ah value, which would take much longer [20].

Fast charging tends to be used more often that the slow charge, since the slow charge rate of 12 hours is often impractical. The fast charge is defined as 1 hour recharge time, which corresponds to 120% of the Ah rating. This is very damaging to the battery and therefore most devices do not use these values.

The circuit below shows an option for a charging circuit that allows for slow charge and fast charge. The slow charging is implemented when the LM2576 Buck Converter is turned off. This allows the resistor R_{TR} to determine the trickle current into the battery. When the Buck converter is on, it acts like a constant current source of 2.6A. A circuit like this allows for variable battery charging profiles.

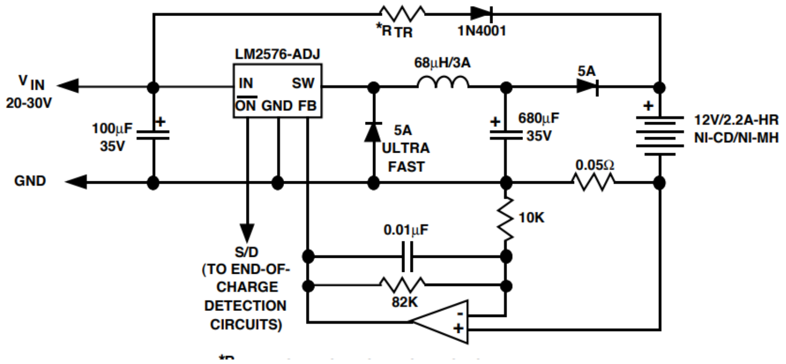


Figure 21. Battery Charger using an LM2576-ADJ [20]

Another critical point when dealing with battery charging is the ability to keep the temperature stable. Charging a battery too quickly can result in an increase in temperature, which can lower the resistance of the battery, increasing the current, and therefore increasing the charging rate. This positive feedback loop can result in batteries that combust, like with Samsung Note 7, which could explode after reaching 80% charged. A temperature sensing circuit can help stop this positive feedback loop and keep the product safe. The circuit below shows an example of a temperature sensing circuit that stop the charging.

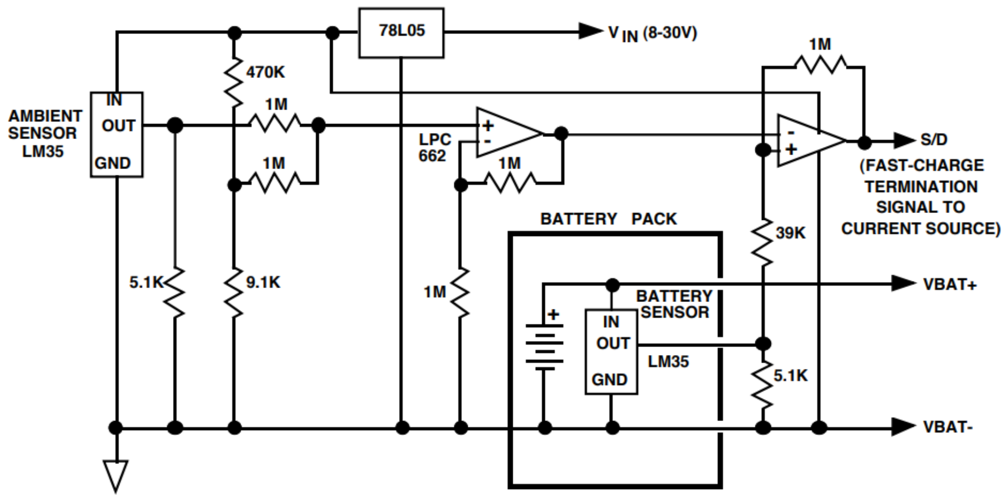


Figure 22. Using the LM35 to signal the Temperature [20]

This circuit uses the LM35 ambient temperature sensor to determine the temperature near the battery. When the temperature is above 10 degrees Celsius, the voltage of the LM35 passes through the unity gain of the LPC662 and is compared to the reference voltage at the rightmost op-amp. If the voltage is higher than at the bottom pin coming from the LM35 near the battery, then the battery is 10 degrees C higher than the ambient, resulting in an output from the op amp. This can be used as a signal to stop the fast charging.

3.2.3.4 Medical Sensors

The need for medical sensors is fundamental to our design requirements, in particular the ability to measure heart rate and blood oxygen levels. These two measurements can be achieved by integrated circuit chips which there are several of available on the market. The technology from which these ICs achieve the measurements is relatively all the same, through two methods: transmissive and reflectance oximetry. This non-invasive method and has a good accuracy (error is within +/- 2%), therefore is a reliable means of measuring oxygen saturation (SaO_2) through reading the peripheral oxygen saturation (SpO_2). This approach is clinically accepted for monitoring oxygen saturation [21].

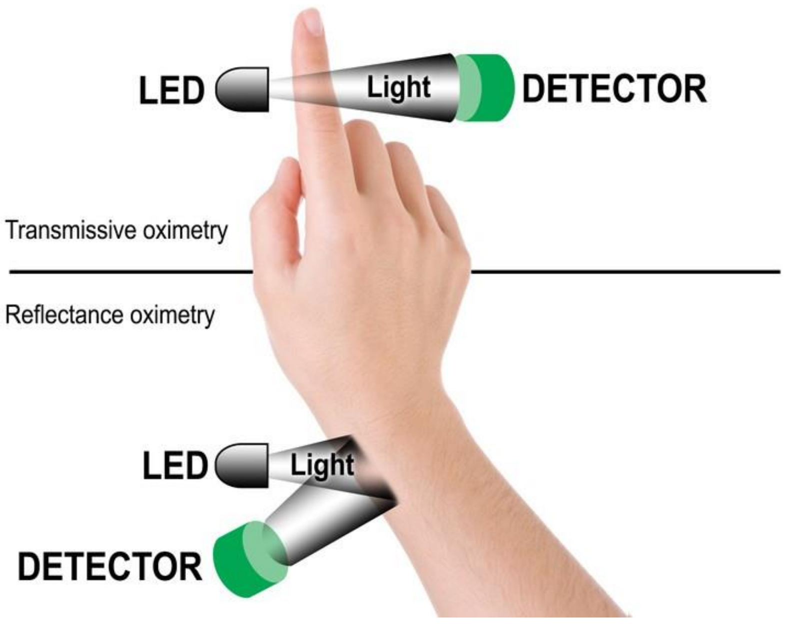


Figure 23. Diode oximetry [22].

ECG (Electrocardiogram) measure the electrical signals inside the heart, often performed in a hospital. This measurement is done by expensive equipment that connects to analog leads which are put on the patient's chest. This standard equipment in hospitals although expensive is very accurate but not portable and does not always have the ability to provide data wirelessly to a device for remote constant monitoring.

Modern electrical engineering applications have made it possible to have small integrated circuits that can perform this measurement with very low power and when combined with wireless technology such as Bluetooth, the analog leads with cumbersome wires can be eliminated and wireless single-pole patches can be used. The measurement data can also be sent wireless over Wi-Fi to an Android device or computer, providing constant measurement without the need to be next to the patient. There are several integrated circuits on the market that provide these ECG measurement abilities.

An ECG waveform represents the electrical signals of the heart using a voltage vs. time graph. P-waves represent atrial depolarization. The PR-interval represents a period of time (in milliseconds) taken for the atria and ventricle electrical activity. The QRS duration is the depolarization of the ventricles. The ST-interval represents an isoelectric line which is a time period between the depolarization and repolarization of the ventricles. The basic understanding of these waveforms is important to us so we obtain the correct output.

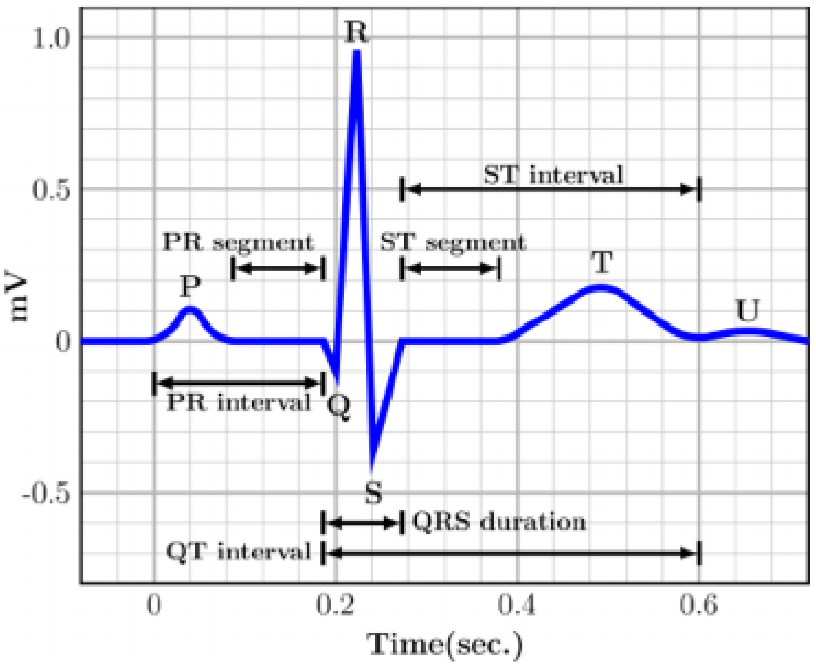


Figure 24. ECG Waveform

The analog connection through electrodes is commonly used through a wet/dry electrode that makes an ohmic contact to the body. Recent research has developed no contact electrodes that use a capacitance electrode which can be connected on the clothing. According to the Intelligent Assistive Technology and Systems Lab at the University of Toronto “The sensor produced highly accurate heart rate measurements (<; 2.3% error) via either direct skin contact or through one and two layers of clothing. The sensor requires no gel dielectric and no grounding electrode, making it particularly suited to the “zero-effort” nature of an autonomous smart home environment [24].”

The possibility of using no contact capacitive sensors is something we might try but there are none on the market, so we will have to design our own. If designing capacitive no contact electrodes becomes too complicated, there are several wireless ohmic contact ECG electrodes on the market – Alibaba has many options ranging from a few cents per unit. Considering our constraint of low cost and limited time, this might be the best option. We will have to further research and investigate this engineering problem.

3.2.3.5 MAX30112

Maxim Integrated manufactures several pulse oximeter and heart ICs. The MAX30112EWG+ has an analog front end with a high-resolution, optical readout signal-processing channel with built-in ambient light cancellation, as well as high-current LED driver DACs, to form a complete optical readout signal chain. Maxim integrated states “the MAX30112 offers the lowest power, highest performance heart rate detection solution for wrist applications. The MAX30112 operates on a 1.8V main supply voltage, with a separate 3.1V to 5.25V LED driver power supply. The device supports a standard

I^{2}C compatible interface, as well as shutdown modes through the software with near-zero standby current, allowing the power rails to remain powered at all times." [25].

This IC meets the constraints of our design; it doesn't have all the features of other sensors available but does include low power and optical pulse oximetry. The MAX300112 uses I2C to communicate data which is slower than Serial Peripheral Interface (SPI) and might prove to be a disadvantage of a constant reading on measurements. The chip also provides flexibility on the location of the sensor which is benefit in case the application of sensor on the wrist proves be an issue. The price on Mouser Electronics is $5.89 per unit, assuming our device will implement only one sensor of this kind the price is well within our budget.

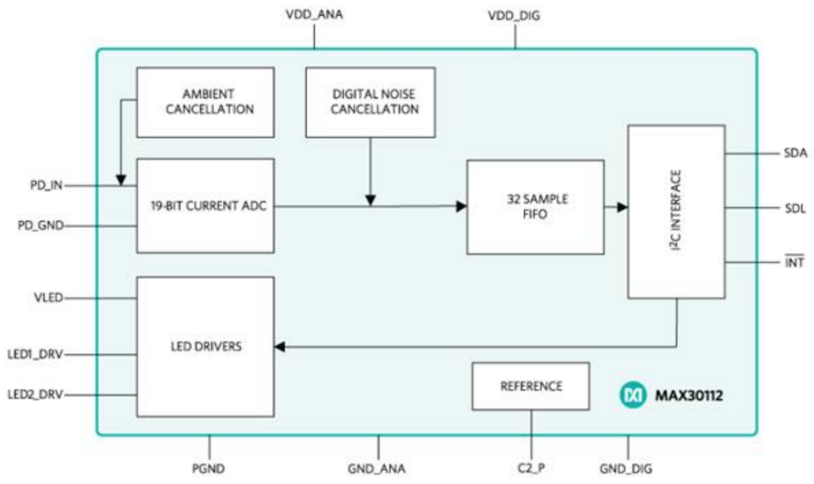


Figure 25. MAX300112 Block Diagram [25]

3.2.3.6 MAX30101

The MAX30101 is another integrated pulse oximetry and heart-rate monitor manufactured by Maxim Integrated that implements finger based sensor location. It is a complete system solution to make the design process easier for wearable devices. The chip operates on a single 1.8V power supply and a separate 5.0V power supply for internal LEDs, it also uses I2C interface. The Heart-Rate Monitor and pulse oximeter are implemented using a LED reflective solution.

Size will not be an issue with this chip, it is 5.6mm x 3.3mm x 1.5mm 14-Pin. It includes an Ultra-Low Power operation mode including a programmable sample rate and LED Current which will be very useful to our device to meet our power requirements. The Low-Power Heart-Rate Monitor is operational at < 1mW and can use a shutdown current of 0.7µA. These specifications meet all our constraints and similar to the MAX300112 but does not offer the flexible of sensor location; it must be measured at the fingertip which will need an analog device connected to the fingertip.

Figure 26. MAX30101

3.2.3.7 MAX30003

The MAX30003 manufactured by Maxim Integrated has a single biopotential channel for providing ECG waveforms and heart rate detection. It uses a 28-PIN TQFB with Clinical-Grade ECG Analog Front End. The built-In heart rate detection with interrupt features prevents the need to use a heart rate algorithm on the microcontroller freeing up resources for other tasks. The device is applicable to single lead wireless patches, single lead event monitors for arrhythmia detection, chest band heart rate monitor, bio authentication and ECG-On-Demand applications. It uses SPI protocol to communicate data which is much faster than I2C. This device provides a lot of applications for more sensors which is the goal of our project to implement as many medical sensors as possible. It is also programmable to send data to an android device which will serve as the data gathering tool in our project.

Figure 27. MAX30003

3.2.3.8 AD8233

Analog Devices offers the AD8233, another IC for ECG and biopotential measurement applications. Like the MAX30003, the AD8233 is a fully integrated single-lead heart rate monitor with an analog front end and useful for the type of implementation we are trying to achieve. In comparison with the MAX3003, the supply current of the AD8233 is less (50µA) vs the 100µA of the MAX3003. The package size of this IC is 2mm x 1.7mm x

0.5mm, which is meets our constraints of small form factor. The price from Digi-Key Electronics is $4.42 per unit which is very cheap but price is not the only factor in our choice.

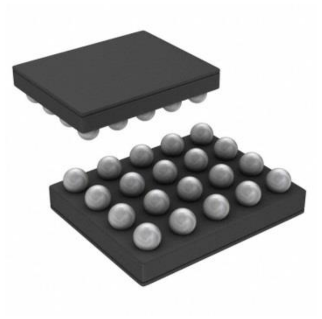


Figure 28. AD8233

3.2.3.9 AFE4400

Texas Instruments manufactures the AFE4400, another fully-integrated analog front-end for pulse oximeter applications. It features a LED driver with H-Bridge, Push, or Pull. Low-noise receiver channel with an integrated analog-to-digital converter The timing control is very customizable, which is needed for us because we do not yet know how we will need our device to be programmed until it is fully tested. The Low Power operates at 100µA. The device communicates to the microcontroller using SPI interface. The package size is very small, VQFN-40 (6 mm x 6 mm).

3.2.4 Temperature Sensor

A temperature sensor for the watch would be an excellent addition to the medical sensors available. Unfortunately, measurement from extremities tends to fluctuate more than the core body temperature, which makes measurements unreliable. We can discuss some workarounds an possible part selection that could still work.

A possible workaround to solve the issue of the imprecise temperature sensor would be to get the measurements from the core of the body. A solution could be to have a thermocouple run up the arm of the patient, an rest somewhere near the torso. While the mouth is a common location to measure the temperature, there are sanitary concerns when this takes place. A better location would be under the armpit, which is considerably less germ-filled.

A better solution would be to avoid contact with the patient at all. Infrared sensors allow for measuring thermal heat as it approaches the infrared ration levels. This could be implemented by having the patint raise their watch to their forehead in order to read the skin temperature. Compraing ti the ambient temperature, a temperature gradient can be deduced.

Both of thses solutions are impractical though. The thermovoule runingup the boddy defeats the putpose of a mobile watch, and the burden of having to raise the watch get the temperature makes the watch more cumbersome than helpful. More solutions can be sought, but the temperature sensor will be a stretch goal.

3.2.5 Accelerometer

During our initial discussion and project research we entertained the possibility of implementing an accelerometer to have an additional orientation and inertial navigation to our device. Upon further research we found this implementation is not entirely accurate although, the error can be corrected through software.

Using the equations of motion it is known that to obtain position from an acceleration value you need to integrate twice, this method has very bad error and it is essentially not useful. It is not the accelerometer noise that makes this error but rather but gyro white noise according to findings by the University of Cambridge [26]. The figure below from the technical report by the Computer Laboratory at Cambridge illustrates this error

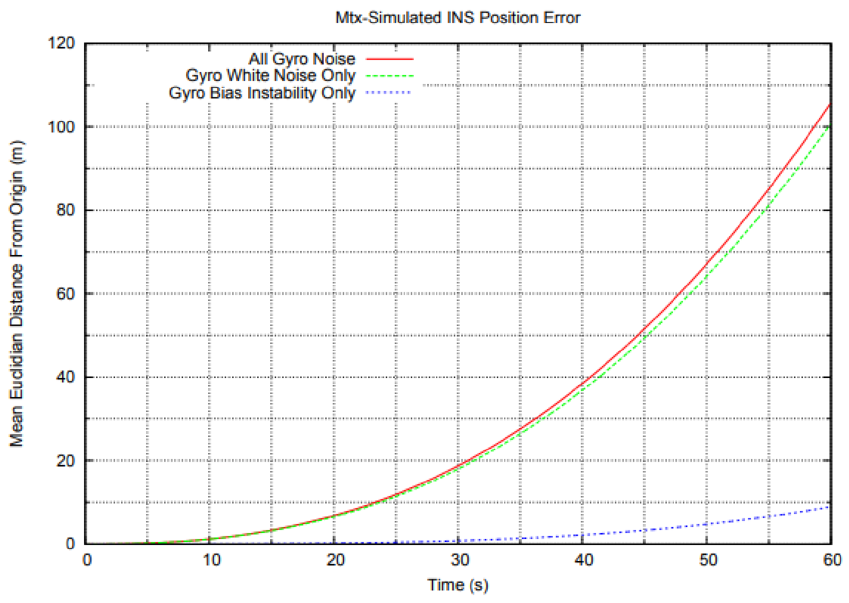


Figure 29. Position Error [26]

There are many types of different accelerometers that are used for purposes of measuring position of a human body in medical and wearable device applications. All of these different kinds of accelerometers are designed on integrated circuits. “Characteristics of physical activity are indicative of one’s mobility level, latent chronic diseases and aging process. Accelerometers have been widely accepted as useful and practical sensors for wearable devices to measure and assess physical activity.”[27].

These inertial sensors use linear acceleration along spatial directions to determine position as according to Newton's Second Law. Some other possible features we will try to include that are made possible by these types of sensors: patient sleep time, fall detection and wake up of the LCD display on the device. According to a study submitted to the National Library of Medicine, "Sleep time duration can be determined from a wrist-worn accelerometer" [27].

The difficulty of implementing fall detection, postural sway, and other types of movement that would be of interest in a medical type device is the location of the sensors. When reviewing the study of accelerometer sensors in the National Library of Medicine cited in this paragraph, when these sensors are placed on the wrist they are only useful for determining steps, MET (metabolic heat measurement), activity intensity level. The study used for their wrist mounted sensor an activity monitor GT3X by ActiGraph, which contains a 3-axis accelerometer and has digital filtering technology. There are several 3-axis accelerometers on the market that we can implement in our design. For our purposes we only require a sensor that will measure up 2.5g or 3g.

3.2.5.1 ADXL345

Analog Device's ADXL335 is a 3-axis digital accelerometer sensor measuring up to +/- 2g,4g,8g,16g of acceleration with 282LSB/g of sensitivity at all g-ranges. The communication protocol is both I2C and SPI interfaces. The sensor can measure static acceleration of gravity for angular-sensing and dynamic acceleration from motion. Activity and inactivity sensing are a feature including on this chip that we also desire to have for purposes for the LCD display and possibly a movement algorithm. It features an integrated memory management system with a 32-level FIFO buffer that will help keep our device low power (23µA measurement mode, 0.2µA standby mode) and free up system resources. The size of the IC is 3mm x 5mm x 0.95mm, LGA package. Digikey offers the ADXL345 for $8.06 per unit.

The benefit of this particular accelerometer is the reference designs provided by Analog Device's. The reference design is a low-g acceleration using the ADuC7024 Precision Analog Microcontroller. This design can be useful in providing a position sensing ability in our device, particularly if the patient was suddenly moved. Wi-Fi cannot provide the sensing abilities of sudden impulse based movement and it is difficult to measure vertical changes using Wi-Fi positioning as well. If we decide to use this particular MCU, then this reference design will be very useful in providing a feature of movement detection and vertical position.

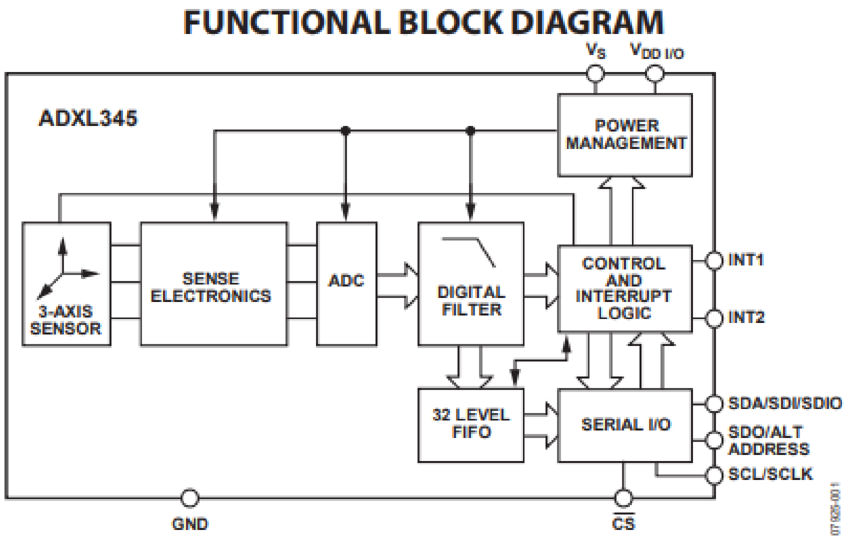


Figure 30. ADXL346 Block Diagram

3.2.5.2 MPU-6050

The MPU-6050 is a 6-axis accelerometer and gyroscope by TDK, which offers us an additional dimension of measurement – rotation and twist. It offers the same g-ranges, protocol communication (I^2C and SPI), and FIFO buffer as the ADXL346. The sensor has DMP ("Digital Motion Processor") a firmware that can do complex calculations with the provided sensor values directly on the chip. The low power mode has selective rates (10µA@1Hz, 20µA@5Hz, 70µA@20Hz, 140µA@40Hz). The package size is 4 mm x 4 mm x 0.9 mm, QFN package with a price of $8.29 per unit on Digi-Key Electronics.

3.2.5.3 Accelerometer Selection

For our implementation the TDK MPU-6050 exceeds our requirements for an accelerometer and includes a gyroscope. The on-chip firmware which can perform its own calculations without using additional resources of the microcontroller is a huge bonus. There are a few source codes on the internet we can evaluate for calculation purposes.

3.2.6 Microcontroller

The brains of the smart hospital watch will need to be able to communicate and control multiple sensors, displays, and other peripherals. The engineering requirements for the watch require that the processing unit be both compact and energy efficient while still being able to perform computations and be more affordable. Microcontroller units (MCUs) are all-in-one integrated chips (ICs) that fill these criteria. Three MCUs that will be compared are the ATMega328PB, MSP430FR4132, and the MSP430FR2033; specifically comparing the cost, power consumption, ease-of-use, and memory size.

3.2.7 Serial Communications

Serial communications are ways to allow communication between devices using fewer pins. This is very critical when concerning MCUs because of their limited pin count and missing communication buses.

3.2.7.1 I^{2}C

I^{2}C Protocol is a synchronous, multi-master, packet switched, single-ended serial computer bus which was invented by Philips Semiconductor in 1982. It uses an Open-drain/Open-collect circuit to communicate the data signal. Bitrate modes of 0.1 / 0.4 / 1.0 / 3.4 / 5.0 Mbit/s can be selected. The bidirectional interface contains a controller (master) that communicates to slave devices. The slave must be addressed by the master to transmit data. Every individual device contains a specific address. Devices can have one or multiple registers where data is read, written, or stored. [28]. The two signals of the bus are SCL (clock) and SDA (data), these two lines make I^{2}C simpler than SPI. Another advantage of this protocol is the ability to support multiple slaves (devices) on a single bus through chip addressing eliminating the need for select lines unlike SPI which requires select lines. Acknowledgement and no acknowledgement (ACK, NACK) communication is another feature of this protocol that SPI does not include. This protocol follows a standard that ensures there are not several different variations of the protocol. The downsides are that I^{2}C uses more power than SPI, it is slower than SPI

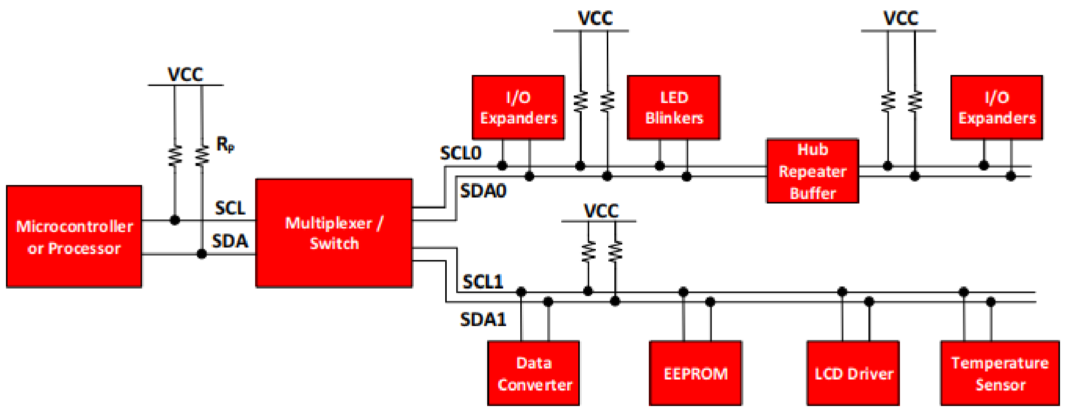


Figure 31. Example I^{2}C Bus [28]

3.2.7.1.1 Addresses of Parts

Unlike the other serial communications, the I2C does not have any chip select wires. This is beneficial because it lowers the number of necessary wires. On the other hand, each device must have a unique address in order to be called upon. When the address uses 7 bits, there are a maximum of 127 devicees that can be called. To add to this, many devices have their addresses built into the IC and might only have a single bit that can chage, resulting in two possible addresses. The table below summarizes the known addresses of the components that we will be using.

Compnent	I2C Address	Alternate I2C Addresses
MPU9150 (Motion Sense)	1101000	1101001
RF430CL330H (NFC)	0101000 to	0101111
ESP-WROOM32(WIFI)	0000000 to	1111111

3.2.7.1.1 Choosing Pullup Resistor

The I2C requires an external pullip resistor in order to drive the output to a high voltage when no IC is using it. This makes it so that any device canstart the communication by pulling down the voltage lines of the data line. However, differing resistors have different affects. The minimum resistance is given as

$$Rp(min) = \frac{(Vcc - Vol(max)}{Iol}$$

Where Vcc is the high voltage (5V), Vol is the valid logical low (0.7V), abd Iol is the valid logical low. In conjunction with the line capacitance, the RC circuit determines the delay time in order to get switch. The time constant would be 1/RC. However, if the resistance is too high, the line might not rise to at the right time.

In out design, we chose Rp to be 4.7 kOhm, but more testing might be required.

3.2.7.2 Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous interface bus developed by Motorola in the 1980's while there is no official standard for SPI, the protocol has become an unofficial standard among electrical engineers in embedded systems. SPI uses full duplex instead of half duplex used by I^2C, and can only use one master to communicate to many slaves.

SPI has four major lines as follows:

- MOSI (Master Output/Slave Input) – Master TX to slave
- MISO (Master Input/Slave Output) – Slave TX to master

- SS[N] (Slave Select, “N-number of slaves” active-low signal) – Master controls this signal, which selects that slave that is to be TX or RX data to or from the slave device.

- SCLK (Serial Clock) – data synchronization clock

These lines can be daisy chained together to connected multiple slave devices. All communication is controlled by the master. When data is requested (RX) or sent (TX) to a slave it pulls the SS line low for that device to activate the clock frequency for the master and slave. Full duplex mode will allow transmission of data over the MOSI to the slave from the master. Data to the slave from the master is sent over the MISO line. There are four main modes in SPI:

- Mode 0 (non-inverted clock logic low): Clock is configured in a manner so that the data is sampled at on the rising edge of the clock and pushed out on the falling edge

- Mode 1 (non-inverted clock logic low): Data sample on falling edge and pushed out on rising edge of clock

- Mode 2 (inverted clock logic high): Data sample on rising edge and pushed out on falling edge

- Mode 3 (inverted clock logic high): Data sample on falling edge pushed out on rising edge

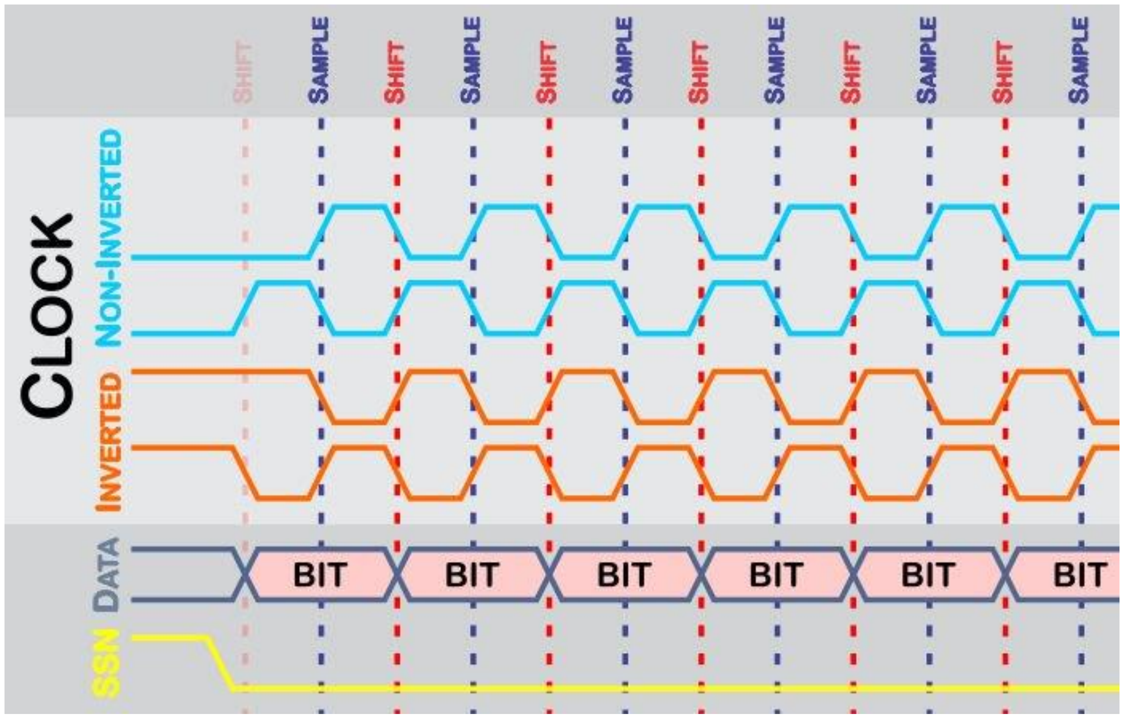


Figure 32. SPI Clock timing diagram

Advantages of SPI: protocol is simpler, faster speeds (push-pull instead of open-drain/open-collector), select lines allow chips of the same type to be connected). Disadvantages: Bus voltage has to be supported by all devices (logic levels cannot be mixed like I2C), three lines plus select lines (one for each device).

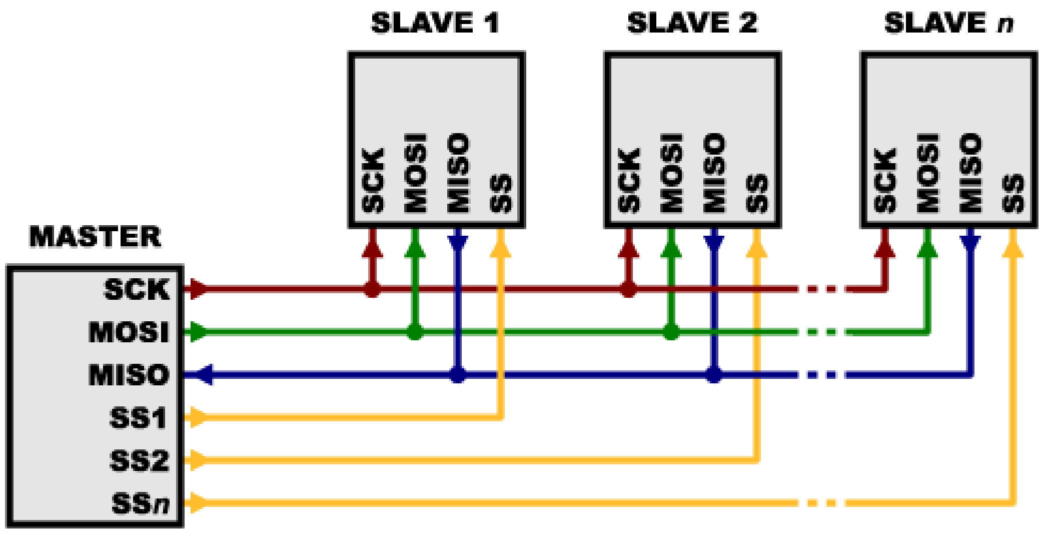


Figure 33. SPI Bus [29]

3.2.7.3 Interface

There will not be much of a user interface with the device: this is done on purpose to reduce the possibility of user created mistakes and misuse of the device that could put the patient in danger. Most of the interface will be either notifications and at most will be buttons.

However, watch faces will allow the communication of data through the display. More is discussed about the software design in a future section.

3.2.8 UART Communications

UART Communications is a physical two wire connection to transmit data between devices. The benefit of UART over other methods is the simplistiy of using only two wires to connect two devices, one for a transmission and one for a recevice. The data is transmitting asynchronously, meaning that no clock is required to synchronize all the bits. Asynchronous transmission makes use of the bits telling the transmission when to start and stop. A sample rate between the two devices is called the baud rate.

The devices are hooked in a parallel structure to transmit data from their respective data bus. Every frame or packet of the UART has certain properties which determine the actions of the frame. A start bit, parity bit, and stop bit – these bits are removed when the frame is sent back out to the data bus by the receiving device.

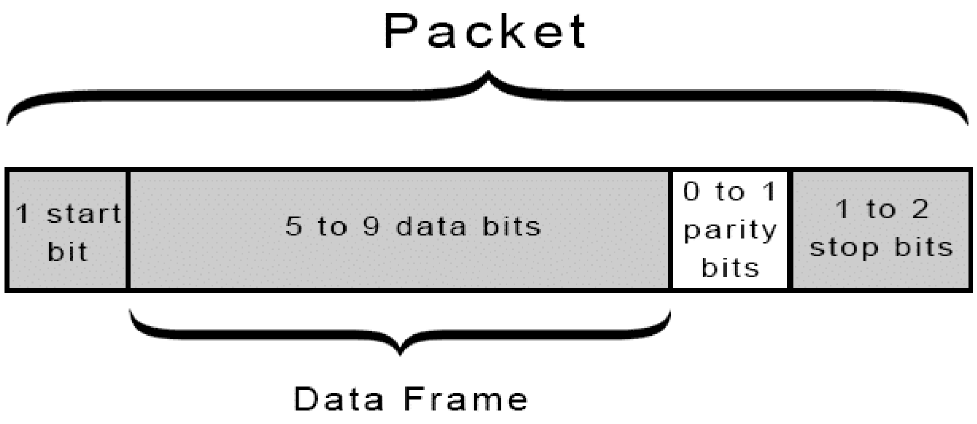


Figure 34. UART Frame

Start bit – the receiving device will monitor the line which contains a high voltage at idle, when a transmission occurs the line is pulled to low and the receiving device will recognize that a transmission is occurring and starts to read the *data frame*.

Data frame – this portion of bits is the data being sent (5-8 bits parity, 5-9 bits no parity).

Parity bit – this bit is a method of error detection, it determiners if the number is odd or even. The error detection is accomplished by telling the device if the bits have been altered during transmission. The 0 (even bit parity), the on bits in the frame should correlate to an even number. The 1 (odd bit parity) the off bits in the frame should correlate to an odd number. This ensures that the data in the *data frame* has not been altered in transmission.

Stop bits – like the opposite of the start bits, the line is pulled from low to high to signify the end of a the frame.

3.2.9 Buttons

There will be a couple of buttons on the watch to provide minimal control of the watch. Since the watch will be mostly autonomous, the watch does not need a capacitive touch screen, and was simplified to a reset button, a menu button, and a panic button combination.

3.2.9.1 Panic Button

A simple button that can be included on the project is a panic button. To interface this button to the microcontroller, a simple connection can be made from the button to the micro-controller input. From here, the software code for the micro-controller would take care of the rest. Such that if the button is pressed, an interrupt occurs, and a signal is sent from the device to hospital system: whether that be a pager system or just straight to their computers. This signal would be sent using the WIFI module; when the interrupt occurs on the micro-controller, a certain signal will be sent to the WIFI module and then transmitted.

Of course, for added patient security and benefit, a mechanism would need to be placed such that the signal continues to be transmitted to the hospital system until the hospital responds to the distress call. This would avoid any failure in signal transmitting but also any accidental cancellation or “no-registration” of the signal. Of course, this feature would need to be well thought out as a constant raise of non-emergency by the patients can be tasking for hospital staff. An alternative to this is the use of a sensor or sensor information instead of a button. If the sensors’ information suddenly changes, then an alert can be raised. This will reduce the possibility of “non-emergency” but will depend on the accuracy of the device’s sensor. Since this is such a sensitive feature, adequate testing will need to be conducted.

3.2.9.2 Reset Button

A simple but valuable button to have on this device is a reset button. This feature will be added on this device to restart any chip or device that seems to be out of “whack.” This would save time since many problems can be solved by simply resetting a device: the common “Have you tried turning it on and off again?” comes into play at this point. Usually such a button would be present directly on the board, but because our PCB board will be inside the case, we will need to have an external button. This button will be placed like common reset buttons found on other devices where the use of a pen or of a small tip object needs to be used to press the reset button. This reset button will probably be connected to the input of the MCU such that when this signal is received a command of resetting is executed. If the MCU has an option of a reset pin, then the button will be tied to this pin.

3.2.10 Vibration

A vibration feature will be added so that a hearing and visually impaired can also receive notifications from the device. Such an application would be if some sort of alarm is going off by the watch; the hearing impaired could be notified by the vibrating feature of the watch. This vibration mechanism is done by placing a simple vibrating mini motor disc placed on the inner surface of the device. A $2.00 mini motor disc found on this website [30] starts vibrating when it receives a voltage between 2V and 5V and stops vibrating when it is disconnected; it only contains two wires which serve as the voltage connections. This can easily be interfaced with the MCU by supplying the voltage directly from it.

An unfortunate effect of having moving parts connected to a circuit is the momentum of those parts inducing a current. When a motor’s supply voltage is stopped abruptly, the momentum of the part will keep the part rotating. This motion will produce a current spike back into the MCU and could result in unintended behaviours of even damage. One way to combat this is to include a ramp down in the software. The more electrical solution to the problem is to put a diode in parallel to the motor. That way when the voltage is reversed due to the momentum of the motor, the diode provides a safety measure against reverse voltage and current.

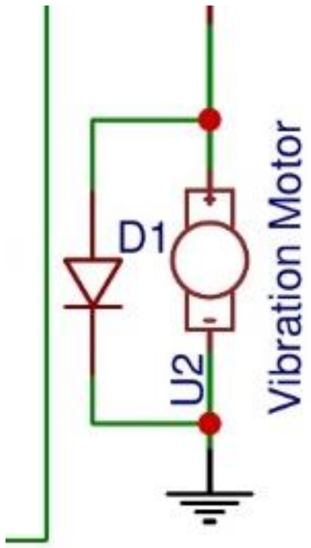


Figure 35. Diode Protection against Reverse Voltage of Motors

3.2.11 Alarm/Speaker

An audio output can be a beneficial component to the smart watch. The most important function of the audio device would be as an alarm. This alarm can sound when the panic mode is activated. This assure that any personal around the patient will be more able to find the patient. Another possible use for the speaker could be to notify the patient if their heart rate is lower than expected, similar to how hospital heart monitors already do. Finally, the speaker can beep when a patient is not near where they are expected to be.

A few a device that can be used as speakers, buzzers, and piezos. Speakers allow for much more complex sounds, including voices, but are often more expensive and use much more current. Piezo and buzzers are low cost sound systems that convert a voltage to motion, resulting in a sound. The Limitation of these sound systems is that they can only output one tone at time, resulting in less than natural sounds. However, it suffices for alarms and beeps.

Many speakers will often require a higher voltage in order to resonate and create sound. Many MCUs cannot provide the adequate voltage in order to provide the sound, so a MOSFET can be used to trigger the sound. The voltage difference at the gate is provided by the MOSFET, allowing for the current to run through the speaker from the higher voltage source.

This design allows for PWM to be provided at a 3.3V level while still allowing the 5V across the speaker. The other protection taken is a current limiting resistor. The 150 Ohm resistor allows for a maximum current of 5/150=33mA.Many speakers have a very low resistance and can be easily blown.

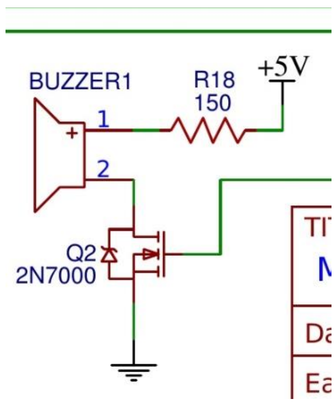


Figure 36. Buzzer Schematic

3.3 Strategic Component and Parts Selection

Comparisons of components with tables including key traits (efficiency, size, power consumption, etc.) and chosen components.

Part selection is a major part and process of any project. Using the house of quality diagram and individual intelligent part requirements, the most ideal chips, sensors, etc. were chosen and the details of "how" and "why" will be detailed in the subsections to come. Much research was made during this process and the research for each individual part of the project can be found in the main section named "Research." Since actual implementation of the whole system has not taken place, it is possible for there to be a change from one chip to another one that was also researched due to new and arising demands or needs.

3.3.1 WIFI Chips

There are countless WIFI chips and modules available from many different providers: even within the same company or manufacturer. To boil it down to the ones shown below two strict requirements were placed: the module must include an antenna, and the module must have ample developer support already existing on the internet. The antenna requirement was used to facilitate the design by avoiding the need for designing/buying an external antenna and then having to go through the process of tuning it which can increase cost and implementation time. The support requirement was placed to avoid any chip that might be under-developed or too new to the market that might not have enough examples for us to use in order to more quickly and easily solve problems encountered during implementation and write code for this application. Both of these pre-filter requirements will help us save a lot of time that would otherwise be spent trying to make original functions for the chip or trying to be the first group trying implement a certain function on this chip. Although these two aspects could provide a good learning experience, the focus of this project is on the overall design of the system. If the sole focus of this project were to learn the workings of a particular chip, then both of these requirements would not be needed since learning these skills would be valuable.

After applying these two requirements, the group was able to boil down the plethora of WIFI chips to just a few. The following table summarizes the key information used when the WIFI chips were compared. This information was also used to determine which specific WIFI chip to be used. There were key features that were considered when reading the different datasheets but some of those features and specifications were common across these chips so that information was not used:

Table 6. WIFI Chips Current Consumption, and Technology Comparison

Name		**TI CC3220MODASF12**	**ESP-WROOM-02**	**ESP-WROOM-32**
Current Consumption	Active	53 mA (Rx 1DSSS, OFDM)	50, 56, 56 mA (Rx b,g,n respectively)	95-100 mA (Rx b,g,n)
	CPU (Modem Sleep)	690 µA (DTIMM1)	15 mA (DTIMM 3)	1-4 mA (DTIMM3)
	CPU (Light Sleep)	115 µA	0.9 mA (U-APSD DTIMM 3)	0.8 mA
Technology		WIFI (802.11 b/g/n)	WIFI (802.11 b/g/n)	WIFI (802.11 b/g/n) & BT 4.2 & BLE
Communication		I2S (2) ,SPI (1),I2C (1),UART(2)	I2S (1) ,SPI (3),I2C (1),UART (2)	I2S (2), SPI (3), I2C (2), UART (3)
CPU Cores		2	1	2
Dimensions		20.5 mm x 25.50 mm	18 mm x 20 mm	25.50 mm x 18 mm
Price		$11.69	$2.70	$3.80

Of these three WIFI chips, the one this group has chosen to use is the ESP-WROOM-32. The main reason this chip was chosen was its compatibility with both Bluetooth BLE and WIFI which gives us the flexibility (if needed) to also use Bluetooth in the indoor localization (using Bluetooth beacons) and patient sensor information monitoring and transmitting. (to mobile devices for example). This does come at the price of greater power consumption, but the group is confident that the power supply designed for this device is more than capable of handling its power consumption. A plus side to this chip is the amazing price of $3.80 which makes it much more inexpensive than the TI chip. (this includes also the big difference in development board prices: $15 and $40 respectively.) The TI chip does offer better development support than the ESP chip and also includes the benefit of being able to use the same IDE as the MCU the group will be using as their main MCU. Just as a backup, the TI chip will be used if the ESP chip cannot perform or do what we need it to do. The two main reasons the ESP chip was chosen over the TI chip were the following: 1)Price of the DEV board and 2) The lack of Bluetooth technology in the TI chip. So for now, the ESP chip and the ESPS dev board will be used.

3.3.2 Bluetooth Chips

As with WIFI chips, there are also an abundance of Bluetooth chips. In order to narrow our search, the same two restrictions used for WIFI were used for the Bluetooth chip: a module containing the chip with an antenna must be available and as well as good developer support. The following table will compare the chips that met that criteria:

Table 7. Bluetooth Modules Comparison

Name	Sablex_R2	HM-10	RN4871	ATBTLC1000ZR
Current Consumption (at RF Port)	7.4, 8.4 mA (Rx, Tx respectively)	8.5 mA	13 mA (Rx,Tx)	5.66, 5.43 mA (Rx, Tx)
Technology	BLE (4.2)	BT 4.0 & BLE	BLE 4.2	BLE 4.1
Communication	I2S (1), SPI (2), I2C (1), UART (1)	UART (1)	SPI (1), I2C (1), UART (1)	SPI (2), I2C(2), UART(2)
CPU Cores	2*	0	0	1
Dimensions	11.63 mm x 17.86 mm	27 mm x 13 mm	9 mm x 11.5 mm	7.5 mm x 10.5 mm
Price	$14.44	$12.99	$7.25	$6.10

When it came to selecting Bluetooth chips, the main purpose for these chips would be for Bluetooth beacons; this means that the ideal chip does not need to be so complicated since the only thing it needs to do is broadcast a signal containing its UID and identification to the device. All the chips above can do this simple task, all are relatively small in size, BLE 4.0 compatible, and have similar antenna specs. Because of these reasons, the main deciding factor for choosing the ideal chip is the price: this makes the ATBTLC1000ZR the chip chosen for this project. In addition to being the cheapest, it also includes an integrated MCU which ensures that we will not need a host or external MCU in order to use this as a beacon. It is important to remember that this chip will only come into play if we decide to incorporate Bluetooth technology in our indoor localization mechanism. This decision will only be made after the device has been implemented and the accuracy of localization through WIFI is measured and recorded.

3.3.3 Microcontroller Chips

The microcontroller is the brains of a circuit, filled with the logic and programming. Below will be the comparison between three different MCUs. The first MCU is the ATmega328PB which is loved by electronics enthusiasts because of its integration in the Arduino Uno boards. The next two chips are part of the MSP430 family of MCUs created by Texas Instruments. They are designed to be low-power while still allowing heavier computations when necessary. The two MCUs from TI are the MSP430FR2033 and the MSP430FR4132. Some critical points when determining which MCU to use are the following: cost, power consumption, memory storage, size/packaging, GPIO pin usage, and the ease of use including programming environment and developer community.

3.3.3.1 Cost Analysis

MCUs are relatively low cost ranging between \$0.10 and \$10, averaging at just over \$1.09. [31] On the other hand, dedicated computer processors like those from Intel can range between from \$100 to \$1800, averaging around \$600 and microprocessors can average between \$50 and \$250 {digikey}. According to Digi-Key, ATmega328PB has a cost of \$1.61, falling at just over the median price for a microcontroller. The MSP430s both fall under the median price with the FR4132 at \$0.99 and the FR2033 at \$0.87, according to the Texas Instruments website. The advantage falls to the MSP430FR2033 for being the least cost.

3.3.3.2 Power Consumption

As the brain of the watch, the MCU must be running at all times that the device is reading from sensors, communicating, or even calculating. Therefor the power consumption of the MCU plays a vital part in the battery life. There are different ways to measure the power consumption of devices, but most batteries use milliamp-hours, or mAh. For the MCUs, we will use the maximum current drawn at a clock rate of 4Mhz and then standardize to an hour. Any special power saving modes will also be considered.

The ATmega328PB can operate at a few different power modes. Active mode uses the maximum computing power and therefore the most electrical power as well. At 4MHz and Vcc of 3V, the ATmega typically uses 1.4 mA. It also has an idle mode which allows for interrupts but stops the CPU, which typically consumes 0.4 mA. A special power-save mode with a frequency of 32kHz and voltage of 3V can consume 2.1 µA, however the only interrupt can come from the clock.

The MSP430s are specifically designed for reduced power consumption, including many power saving modes. The FR2033 and the FR4132 both use 126µA/MHz, resulting in 0.5 mA at 4MHz and Vcc of 3V. The MSP430 equivalent of the ATmega's idle mode is LPM0, which halts the CPU as well but allows for interrupts and immediate response. This mode uses 80 µA at the same voltage and clock rate. The counterpart to the power-save mode is the LPM3.5, which can only use an interrupt from a 32kHz clock, and uses 0.77µA.

The advantage of the power consumption ties both the MSP430FR2033 and the MSP430FR4132, both with identically low power consumption. The ATmega328PB was not nearly as power efficient. The table below shows an estimated time that a 30mAh battery would last if just the processor was used. (Assuming it would be perfectly efficient.)

Table 8. Time to Discharge 30mAh Battery with only Microcontroller

	Hours if 100% Active	Hours if 20% Idle	Hours if 20% LP	Hours if 20% Idle 20% LP
ATmega328PB	21.4	25.0	26.8	32.6
MSP430s	60.0	72.1	75.0	94.9

Realistically though, most power would be consumed by the sensors, communication systems, displays, and any other peripherals.

3.3.3.3 Memory Storage

A key difference between microprocessors and microcontrollers is the memory. Like in computer processors, microprocessors have very limited onboard storage and have to rely on external memory storages like RAM chips and hard drives. Microcontrollers on the other hand have storage built into the chip. Each IC has its own form of storage which can range from read-only memory, write-once, rewriteable flash, or even purely volatile memory. The data on these devices is normally divided into nonvolatile memory and some sort of RAM. The nonvolatile memory is normally used to store the programming of the MCU and data that will be mostly constant. The size of the nonvolatile memory can often determine how complex expansive the program can be. On the other hand, RAM is significantly faster and is used to store variables that are adjusted frequently and for calculations. A small RAM could mean that the calculations will take much longer due the high cache miss rates, resulting in having to pull date from the nonvolatile memory. The memory type and size of each MCU must be compared.

The ATmega328PB has three different kinds of memory: Flash program memory, SRAM data memory, and EEPROM data memory. The flash program memory is meant a readable section of the memory where the instructions of the program are written once. However, the ATmega has protocols to reprogram (rewrite) this memory. This memory is 32KB, organized as 16K *16. The next memory that the ATmega has is the SRAM, which stand for Static Random Access Memory. An important note of this memory is that SRAM is a volatile memory and data will be lost of the power supply drops too low or the device shuts down. SRAM starts 303 Bytes for the general-purpose registers, I/O registers, and External I/O registers. Following this, there are 2KB of data called the internal SRAM. The figure below shows a map of the SRAM.

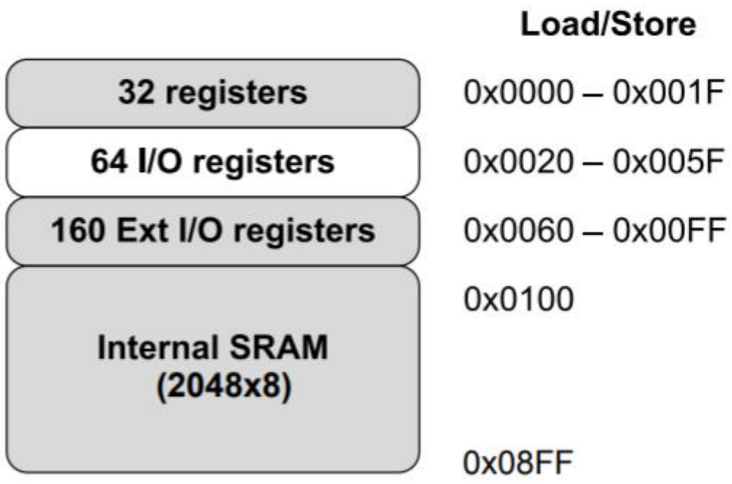


Figure 37. Map of ATmega328PB's SRAM

The final data used for the ATmega is the EEPROM data memory, which stand for Electronically Erasable Programable Read Only Memory. This data is non-volatile, and contrary to the name, is not just read only. While the EEPROM is normally written using an external programmer, the ATmega has procedures to write to the 1KB of EEPROM. However, this process can lead to data corruption and requires many steps in order to safely assure no data loss.

The MSP430s have similar memory types but the data is organized somewhat differently. The two main types of data storage are FRAM and the SRAM. The FRAM is similar to the Flash memory of the ATmega in that it is a nonvolatile memory that is used for storing the programs and constants. FRAM stands for Ferroelectric Random Access Memory. This memory uses a special ferroelectric transistor that changes magnetic polarity when under an electric field but keeps the polarity when the field is removed. This means that FRAM uses less power than Flash and has better write/read performance. [sThe FR2033 has 16KB of FRAM and 2 KB of SRAM while the FR4132 has 8 KB of FRAM and 1 KB of SRAM.

By absolute size comparison, the ATmega328PB seems to have the advantage with 33 KB of nonvolatile memory while matching the 2KB of volatile memory of the MSP430FR2033. However, a key note is that the slow read/write speeds of the flash memory (and even slower times of the EEPROM) might make the FRAM based systems more efficient with their data.

3.3.3.4 Packages and GPIO Pins

The usage of any integrated circuit will determine the limitations that will be imposed when considering PCB design. Many different packages have different number of GPIO pins, different sizes (which affect the footprint on the PCB), and even the ease of soldering that is required to add to the PCB. Each of these components will be discussed for each available package of each chip. First some definitions are important.

3.3.3.4.1 Package Definition and Visuals

As electronics continue to shrink and condense, the components of the circuits must also find a way to take up less space. Traditionally through-hole components were used, as can be seen by many 80s and 90s video game systems. As the circuits condensed, surface mount devices (SMD) became necessary.

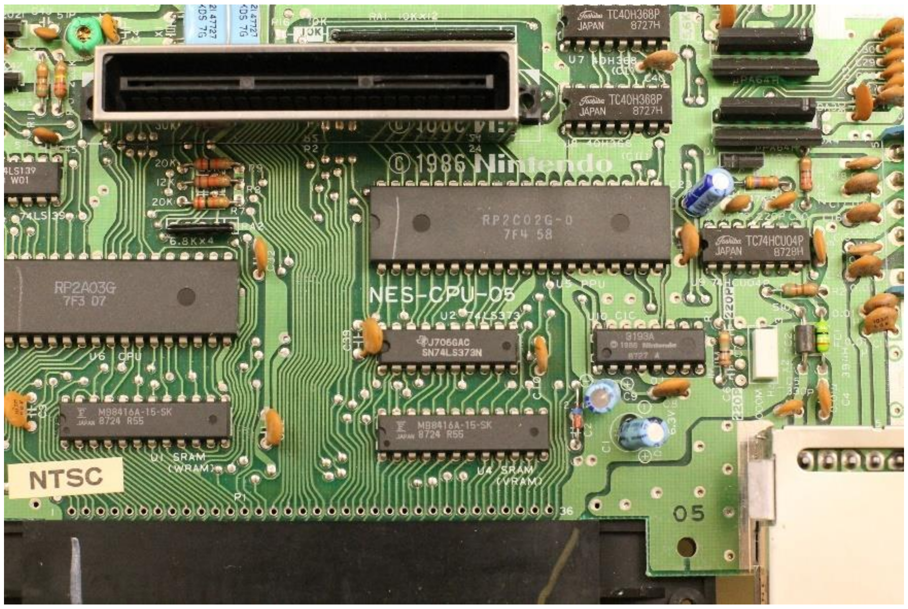


Figure 38. NES Motherboard with Through-hole Components [32]

A smart watch must be compact and light, meaning that we must use SMD in our design. A few different packages for ICs (from least to most space saving) are DIP, TSSOP, TQFP, QFN, and BGA. Our MCU must have a package that will allow for saving space and easy circuit design.

The most commonly seen package for electronics enthusiasts are the dual in-line packages (DIP) because of their simple way of mounting to a breadboard/perfboard for prototyping. DIP have pins on the longest sides of the IC, with the first pin nearest the dot/semicircle and the rest following in order counter clockwise. These packages cover the most amount of space and require through-hole PCB design. The thin shrink small-outline package (TSSOP) is the next easiest for a hobbyist to use. It looks like DIP except the leads are much shorter and must be surface mounted. The thin quad flat-pack (TQFP) is very similar to the TSSOP except that the pins go on all four sides of the IC and are generally square instead of rectangular. The quad flat no-lead is similar to the TQFP except that the leads are removed entirely. At this point, hand soldering becomes very difficult. The final package is the ball grid array, which places the pins under the chip and requires a soldering oven to properly mount. This is the most space saving but requires much more planning when designing the PCB.

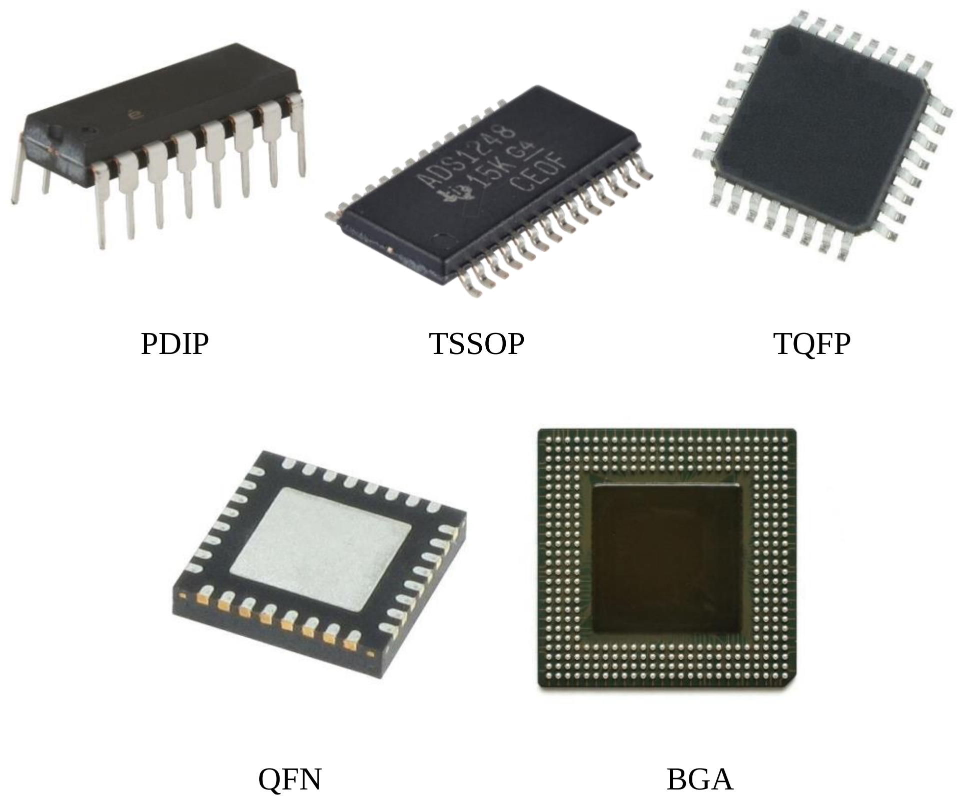


Figure 39. Different Package Options

Another important point when deciding the best package is the number of general purpose I/O pins. A limited number of pins could require extra hardware to multiples the pins, and thus inducing a higher cost and program complexity. The number of GPIO pins should be maximized while choosing the most streamlined version of the package.

3.3.3.4.2 Available Packages with GPIO Count

The ATmega328PB has a predecessor which came in many different packages including DIP, however this model only comes in TQFP and QFN, both of which are very space saving. However, due to the specialized soldering equipment needed for the QFN, the TQFP would be the desired package. Notably, this MCU has a total of 32 pins of which 23 are GPIO. The total size of the 32 pin TQFP ATmega 328 is 9x9mm^2 with leads.

On the other hand, both MSP430s come in TSSOP and LQFP (Light QFP, very similar to TQFP but slightly taller. 1.0mm vs 1.4mm) [33]. The LQFP package would be ideal since it would use the least amount of space due to its pins being on all four sides. Both the FR2033 and the FR4132 have 64 pins of which 60 are GPIO. The advantage goes to the MSP430s because of the larger number of GPIO Pins

Table 9. Microcontroller Feature Comparisons

	Package	Pins	GPIO Pins	L (mm)	W (mm)	Area (mm^2)
Atmega328PB	TQFP	32	27	9	9	81
	VFQFN	32	27	5	5	25
MSP430FR4132	TSSOP	48	44	8.3	12.6	104.58
	TSSOP	56	52	8.3	14.1	117.03
	LQFP	64	60	12.2	12.2	148.84
MSP430FR2033	TSSOP	48	44	8.3	12.6	104.58
	TSSOP	56	52	8.3	14.1	117.03
	LQFP	64	60	12.2	12.2	148.84

3.3.3.4.3 Specialized use of GPIO Pins

Microcontrollers often multiplex their GPIO pins in order to add extra functionality to the MCUs. These extra functionalities can come in a variety of options but the ones addressed in this section will include analog-to-digital converters, LCD compatibility, and Serial communications.

3.3.3.4.3.1 Analog to Digital Converter

The world around us is filled with information that often has a range of possible results. The sound that we hear are waves with a volume and frequency, the colors we see have different hues and saturation. Our human minds can understand these values as being within a range without specifically having to know exactly how much red is in an apple. Computers on the other hand must model their understanding of the world with numbers. Computers must quantify the amount of red in an apple as a series of ones and zeroes, binary.

When a sensor takes a measurement of the real world, it will often convert it to an analog voltage. An example would be a photodiode. When the photodiode has no light, it produces no voltage, but as the light increases, it outputs more voltage. Analog to digital converters (ADCs) approximate this voltage into a sequence of binary numbers. Therefore, MCU needs an analog converter when getting measurements from sensors.

One of the most common analog to digital approximation is the successive approximation ADC (also known as SAR because a register is used to store the result). The way that SAR ADC works is by successively comparing whether a voltage is above or bellow a specific reference voltage, and then storing the result. For example, if an ADC has a maximum voltage of Vmax, the reference voltage Vref would be begin at ½Vmax. If the input voltage Vin is below Vref, the most significant bit (MSB) is set to 0. Then the new Vref would be the middle value between 0V and ½Vmax, meaning Vref is ¼Vmax. If Vin is above that value, then the second MSB would be 1. The next Vref would be half between ¼Vmax and ½Vmax, or ¾Vmax. The process continues until the least significant bit is approximated.

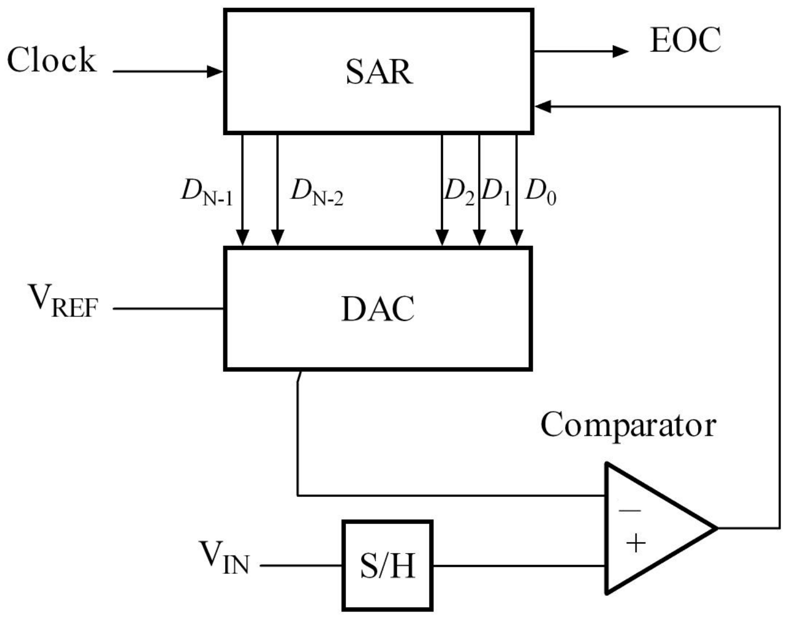


Figure 40. SAR ADC [34]

The key points when looking at an MCUs ADC are the number of analog inputs, the number of bits in the SAR, the conversion time, and the sample per second. The number of analog inputs will determine how many analog devices can be used without the need of extra hardware like external muxes.

The ATmega328PB has 8 channels that can be used for the analog inputs, however 2 pins are used in an I^2C and another 4 in an SPI, however the ATmega has multiple SPI and I^2C. The MSP430s both have 10 channels that are used for analog inputs, with 4 pins used for SPI and two of those are also used for UART. Both the ATmega and the MSP430s have a 10 bit resolution. At 3V max, the precision can be calculated to the $3/2^{10}$ V or roughly 3mV.

Table 10. Microcontroller Communication Features Comparisons

	Bits Resolution	Analog Pins	Used for Serial Communication	Alternate Comm Port?
ATmega328PB	10	8	6	Y
MSP430FR4132	10	10	4	N
MSP430FR2033	10	10	4	N

The final key points to the ADC are the conversion times and samples per second. The conversion time determines how quickly the value can be converted and used by the MCU for calculations. This is a bottleneck in the program that can immensely slow down the efficiency of the MCU. The ATmega328PB claims to convert between 13 and 260 µs. The MSP430s claim to have a conversion time between 2.18 and 2.67 µs, an immense improvement. The number of samples per second are describe in the table below.

Table 11. Microcontroller ADC comparisons

	Min. Conversion Time (µs)	Max. Conversion Time (µs)	Min Samples per Second (ksps)	Max Samples per Second (ksps)
ATmega328PB	13	260	3.85	76.92
MSP430FR4132	2.18	2.67	374.53	458.72
MSP430FR2033	2.18	2.67	374.53	458.72

For our purposes, the analog devices that we will measure will be sampled less than 200 times so both chips surpass the requirement. The MSP430s have the advantage of having more analog pins but they are used by the serial communications and have no alternatives. Effectively, the ATmega takes the edge because it ends up with more available analog channels with more than enough sampling rate.

3.3.3.4.3.2 LCD Compatibility

Our watch will require a way to display the data that t is tracking, including the heart rate, oxygen levels, the patients name, ID, and allergens, and even the current time. While the specific LCD options will be discussed in a later section, a very important note about these displays is that they often require may GPIO pins and sometimes additional drivers to control properly. One benefit that the MSP430FR4132 has is that allows for driving a variety of LCDs within the MCU, not requiring an external driver. The FR4132 allows for controlling an LCD by directly writing to onboard memory, and has the option for 2-MUX, 4-MUX, and even 8-MUX which allows for controlling large displays simpler. It also allows for the display to be used in its lowest power state, LPM3.5, which could prove useful for saving energy on computation.

The ATmega328PB and the MSP430FR2033 do not have specialized hardware for driving LCDs so the advantage goes to the MSP430FR4132 in this case. However, all of the chips can use GPIO pins to control the LCDs as well, it would just be more complex. Another note is that many LCDs also come with the drivers built into the package, including their onboard memory and settings. These displays require less complicated controls to modify, and also mean fewer GPIO pins necessary. Some LCDs have built in processing hardware and will just communicate with the MCU using just a few pins and serial communications.

3.3.3.4.3.3 Serial Communications

Serial communications provide ways to communicate with external devices using only a few pins. Parallel communications transmit lots of data at a time using many pins, like for example the parallel port in older desktop computers used the DB25 standard which used 25 pins to communicate. Microcontrollers simply cannot commit all the pins or else it could not have many GPIO left. Fortunately, serial communications use much fewer wires at the cost of sending the information serially, or bit by bit.

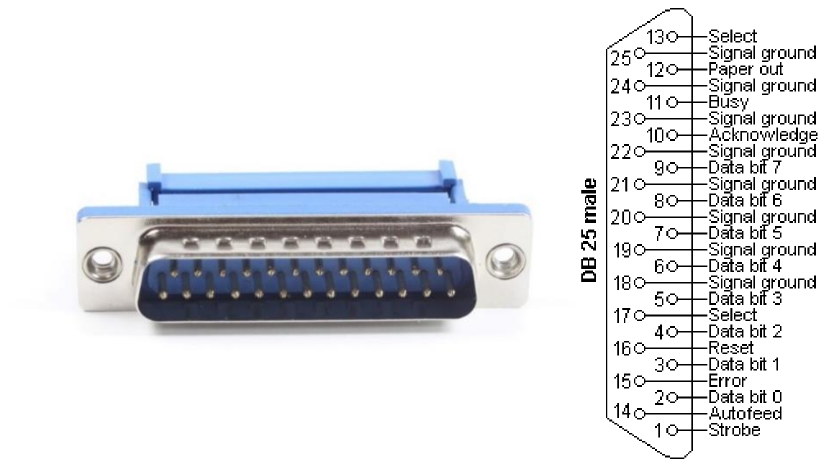


Figure 41. DB25 Male Header and Wiring Guide

The serial communication standards are discussed more thoroughly in section 2.8, but here the different communication methods available to each microcontroller.

The ATmega328PB has a few different available modes of communication. Atmel has adjusted the standard of I^2C and formed TWI, however they function nearly the same and so will be called I^2C.

The benefit of the I^2C standard is that it uses only two wires to interface with a number of devices. The ATmega can use two separate I^2C buses, one using pin 27 and 28 as the data and clock lines, respectively; and the other using pins 3 and 6 as the data and clock lines, respectively.

The next serial communication used by the ATmega is the Universal Synchronous Asynchronous Receiver Transceiver, or USART. The benefit of this communication standard is that it is highly customizable, allowing for different baud rates, odd or even parity, synchronous or asynchronous communication, and much more. The ATmega can use two different USART busses. Bus 1 uses pin 2, 30, and 31 as the clock, receive, and transmit lines, respectively; and Bus 1 uses pin 15, 16, 17, and 31 as the clock, receive, and transmit lines, respectively

The final serial communication built into the ATmega is the Serial Peripheral Interface, SPI. The benefit of this standard is that it allows for the MCU to decode the clock and rate of transfer to the devices, making the MCU the master and the devices the slaves. This is connected in a circular pattern where the data is passed from the master, through the slaves, and back to the master. However, the standard uses at least 4 wires. The ATmega has 2 SPIs using pins 14, 15, 16, and 17 for the slave select, master-out slave-in, master-in slave out, and clock, respectively, for the first SPI; and pins 19, 22, 23, and 24 for the slave select, master-out slave-in, master-in slave out, and clock, respectively, for the second SPI.

The limitation of the ATmega328PB when it comes to the serial communications is that the USART_1 and the SPI_0 share some pins and therefore cannot be used at the time. Luckily there are alternative versions of each communication protocol so there is always an alternative. The table below shows the pins used in each serial communication.

Table 12. ATmega328PB Pin Usage

Communication	Pins Used	Conflict
USART_0	2, 30, 31	
USART_1	15, 16, 17	SPI_0
SPI_0	14, 15, 16, 17	USART_0
SPI_1	19, 22, 23, 24	
I2C_0	27, 28	
I2C_1	3, 6	

The MSP430 use the same communication standards that the ATmega328PB does besides the USART, instead just using the UART and leaving out the synchronized version. However, the MSP430s use just 8 pins total for all the standards. The communication standards are divided into two groups eUSCEI_A0 and eUSCEI_B0 (Enhanced Universal Serial Communication Interface). eUSCEI_A0 can be used for either SPI (using pins 32 to 29 as slave select, master-out slave-in, master-in slave out, clock, and slave select, respectively) or UART (Using pin 24 for tansmit and pin 23 for receive). eUSCEI_B0 can be used for either SPI (using pins 24 to 21 as slave select, master-out slave-in, master-in slave out, clock, and slave select, respectively) or I^{2}C (Using pin 30 for data line and pin 29 for clock line). This limitation means that Three communication types can't be used at the same time. The table below hows the information more clearly.

Table 13. Communication Standards Difference

	PIN	UART	SPI
eUSCEI_A0	24	TXD	MOSI
	23	RXD	MISO
	22		CLK
	21		SS
	PIN	I2C	SPI
eUSCEI_B0	32		SS
	31		CLK
	30	SCL	MISO
	29	SDA	MOSI

3.3.3.5 Ease of Use

A critical factor when selecting the appropriate microcontroller is the ease of programming the MCU. Each Microcontroller uses each its own development hardware and software environment.

While the ATmega has a vast community of hobbyist that use it through the Arduino, the MSP430s have professional documentation and programming software. The Code Composer Studio is the IDE for the TI chips, and it has interactive memory management of the chip while debugging. And most helpful is the single step instruction steps which aloe for immense debugging capabilities.

3.3.3.6 Summary of MCU

The chosen component is the MSP430FR4132, and if that part is unavailable, the MSP430FR4133, which is the same chip with more memory. The table bellow summarizes the key features of the chips.

Table 14. MCU Comparisons

	ATmega328PB	MSP430FR2033	MSP430FR4132
Cost	$1.61	$0.87	$0.99
Lowest Power Mode	2.1uA	0.77uA	0.77uA
Idle Power	400uA	80uA	80uA
Typical Power	1.4mA	0.5mA	0.5uA
Memory (Volatile)	2KB	2KB	1KB
Memory(Nonvolatile)	32KB	16KB	8KB
Smallest Package	81mm^2	148.84mm^2	148.84mm^2
Pins	32	64	64
ADC Resolution	10	10	10
LCD Chargepump	N	N	Y
SPI	Y	Y	Y
I2C	Y	Y	Y
Best Compiler	-	W	W
Largest Community	W	-	-

3.3.4 LCD Display

The LCD display we would like to implement has to be small enough to fit on the wrist but large enough to clearly display information. This size would be comparable to a display commonly found on a smartwatch (1.5-2 in). We do not require our display have touch capability. Ideally we would like to use a cheaper TFT display to keep the cost low.

A potential choice for our display is the NHD-1.8-128160EF-CSXN#-F which has a resolution of 128x160 with a 1.8 in display that uses parallel interface. It requires an operating supply voltage of 2.8V and supply current 30mA. We desire to turn off the screen after a couple of seconds until movement is detected to converse power since there is no low power mode. The unit cost from Mouser Electronics is $16.25

Another option would be the NHD-0216HZ-FSW-FBW-33V3C, which offers a simpler display. It is a 16x2 charater LCD display with serial communications. It has a white LED backlight which can be turned off to conserve energy. Without the backlight, the device consumes 1.5mA and about 20mA with the Backligh on. The unit costs $11.60 from Mouser.

3.4 Parts Selection Summary

After comparing a majority of different components we have come to a decision for each technology and chosen what we feel is best for our device. The final parts we chose are the ones that meet our requirements and standards and will be expected to perform as we planned. Along the way some parts may still need to be swapped for a different one, but we will find that out as we go into testing the overall system in senior design two. The way we chose parts was by splitting the different technologies between our group and then individually researching similar devices with the features we desire. We then looked further into the variety of components in that realm of application and this is where we decided ourselves what parts are best for us to use based on comparing the voltages, current flow, price, size , and more to decide on our components.

3.4.1 RFID

As stated before we will be using RFID technology to store the patient's information for medical applications around the hospital. Looking between the many RFID options of being active, passive, separate, or connected to the main circuit we finally decided on the RF430CL330H. This part was the best RFID/NFC chip to choose because it was only $1.89, could do both SPI and I2C communication, NFC is more secure than normal RFID because it is one on one device communication, NFC can also be read and written. Allowing the NFC to be read and written is very convenient for our product due to the fact that we want to be able to modify the information if anything happens or changes. More features of our choice were that it can easily be used with a smart phone device allowing us to be able to test it very well. It also has the ability to do blue tooth pairing and the best feature of all is that it is extremely compact, giving us plenty of work room for our watch design. This chip even states in the datasheet that it is good for medical devices, which is perfect for us. The RF430CL330H was by far the best choice for RFID technology.

3.4.2 Analog Front End / Pulse Oximeter

The Texas Instruments AFE4400 was chosen for our final selection in the analog front end part of our design which will drive the LED signals and convert the analog signal from the Nellcor-DS100 pulse oximeter probe. The reason we chose this particular component was the support and reference designs with this AFE in conjunction to the MSP430. All the other researched AFE’s were acceptable but the reference design was a huge plus when considering all the other circuitry involved in this project, it was imperative to make sure that the pulse oximeter functioned correctly – which the AFE provides the signal and communication to the MCU.

The Nellcor-DS100 pulse-oximeter SP02 fingertip sensor was select for the probe to receive the data for the heart rate measurement. This sensor was chosen because of it’s accuracy and wide usage throughout the medical industry. It also uses a DB9 connection that is easily interfaced with the chosen TI AFE and MSP430 using the TI reference design which will ease our design process throughout the limited timeframe.

Further ensuring accuracy the sensors features a digital memory chip that is embedded into the sensors. This memory chip provides calibration inside the sensors before it is calculated with an algorithm.

3.4.3 Voltage Regulators

Our main desires for choosing a voltage regulator to use was size and efficiency. We wanted something very compact, but at the same time very efficient so our device could be powered by the battery without wasting too much power and run for a long time. What we first came down to using was the TPS6306 that we actually found from a similar application device in the medical field. These voltage regulators are the smallest parts I have ever used, which was alarming at first but after much thought very good for our watch dimensions. It also operates at about 94% efficiency and can output 1.2 to 5.5 volts which covers all of our components requirements, which are all around 3.3 or 5 volts.

It even contains overtemperature protection and a low power mode which will be great for our watch to last a long duration of time. It will be a bit tricky getting these regulators on the board ourselves if we want to try and bake them on, but we may look into the option of a professional putting these small components on.

As we began to assemble and test our design we realized the TPS63036 were actually not as straight forward as we had hopped for. We had done all of the calculations for ouput voltage and making sure it worked properly, but due to lack of time, only making one PCB, and not being able to dissect it due its compactness we went with the REG104FA-3.3500. This regulator was a low dropout voltage type and automatically converted the input to 3.3V without any calculations. Very straight forward and easy to use, which was good for us time wise.

3.4.4 Power Supply

Now when it came to deciding what type of battery to use we knew from the beginning it was going to be lithium ion, but we didn't know how large, the capacity of it, and what type. There are many types of batteries out there from coin to block to cylindrical. At first when deciding what type of battery to use we were thinking of using a block battery which could easily fit underneath or above the over all device. Then we thought about it some more and realized for our prototype design the dimensions gave us a lot of room for a somewhat big battery to start out with.

We then changed our layout and looked into the cylindrical style battery to put off to the side of the over all device and settled on the EBL 18650 lithium ion battery. We chose this battery because it was very powerful with about 3000 mAh and a very long life. Although the major down fall to this battery is it is quite big, which is not bad for testing and the prototype. As time goes on though we may look into getting a smaller battery and determine how efficient our battery can actually be.

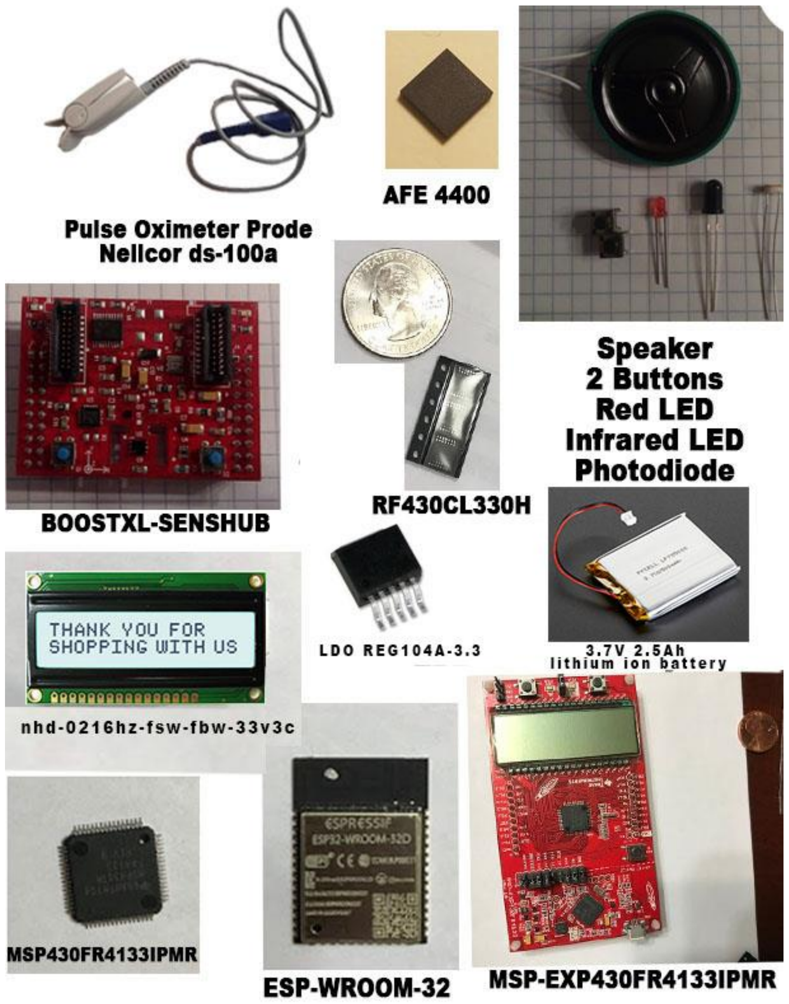
Later on in the experiment we changed our battery once more and finally went with the Adafruit 3.7V 2.5 amps per hour battery because it was flatter and could lay underneath our PCB nicely in the case.

3.4.5 Display

The final decision of the display was NHD-0216HZ-FSW-FBW-33V3C. It allowed for a simpler design since it used only character dispay which was simpler to program while still providing the critical information. With the toggling of the backlight, the LCD consumes 1.5 mA without the backlight and 20mA with it on.

Table 15. Final selected parts

Parts Selected
Nellcor-DS100
AFE 4400
REG104FA-3.3500
RF430CL330H
3.7V 2.5 Ah Adafruit battery
NHD-0216HZ-FSW-FBW-33V3C
MSP430FR4133
ESP-WROOM-32
MSP430FR4133



4 Related Standards and Realistic Design Constraints

Intro to standards and constraints and how they can affect design, both on the engineering and marketing. What they are, how they can improve quality, etc.

Standards are usually codes or rules that a designer must take into account when developing a product to ensure that once the product goes to the market, it will be compatible with other systems that already exist. This limits the designer in some aspects by deciding what the designer must include in his design to meet the standard; in the long run this is good because it helps the designer narrow in his options and guide him to produce a good quality product. Design constraints are usually "wants' or "needs" of the user or market imposed usually by the customer or person requesting the product. Design constraints usually point to a feature or a goal of the design like maximum speed desired, size desired, etc. but they can also be constraints that are placed by the environment such as climate, weather, operation region. These constraints give the designer broad or small goals that he must achieve in his design; good constraints give flexibility to the designer, but constraints can be generic and specific (eg. watch must be able to locate patient: watch must be able to locate a patient within 1 m accuracy and within 0.5 seconds.) This portion of the document focuses on the impact of realistic design constraints and how they affect the design of the Smart Hospital Watch. It will also identify and examine any standards that are related to the technology used in our system design.

4.1 Standards

"Engineering standards are documents that specify characteristics and technical details that must be met by the products, systems and processes that the standards cover" [X]. Some standards are mandatory and at times are voluntary; nevertheless, abiding by existing standards ensure safety, practicality, and compatibility with systems that already exist. Trying to "create" something without using standards would be dangerous and result in a product that would not fit in with the rest of the world making it some cases useless. There are many standards and regulations to meet when designing the smart hospital watch. Especially in the environment of a hospital setting where there is sensitive personal data, medical equipment for monitoring people, and much more. Meeting the standards will allow us to create a product that is simple to develop and easily accepted into the hospital setting. A few standards we will encounter are wireless communications,

4.1.1 Search for standards

Searching for standards can be quite difficult depending on where you look. Some websites require you to purchase the documents while some are free due to the status of being students at the University of Central Florida. The main websites we used to research the standards that will affect us and our device were https://ieeexplore-ieee-org.ezproxy.net.ucf.edu/document/7927764/ and https://www.fda.gov/medicaldevices/digitalhealth/wirelessmedicaldevices/default.htm. These two websites were very helpful in finding the standards that applied to our device and the technologies we are using.

4.1.2 Wireless Communication Standards

Wireless Communication standards cover a very broad spectrum of topics covering communication protocols, frequency spectrum usage, RF coexistence, etc. With the advancement of wireless technology, there has been a similarly large development of "wireless" standards. These standards exist to ensure that wireless communication, wireless devices, and wireless networks does not become a "big mess" to the point where it is impossible to communicate wirelessly or operate in specific frequency bands due to strong interference. This following section will present some of these wireless communication standards that are relevant to this project.

4.1.2.1 ANSI C63.27 Evaluation of Wireless Coexistence

As the title of this standards denotes. this standard deals with the appropriate ways to evaluate the co-existence of a final product (not of individual WIFI chips that make up the system since the standard states this can lead to very large inaccuracies) in the intended product environment: this is largely a testing standard. This standard provides 4 main evaluation setups for wireless devices depending on the kind of network and expected function that they are to make as well as guidance for test setup, parameter identification, and more. The evaluation setups in this standard are based on using an unintended signal or signals to interfere with the device and the intended signal to evaluate whether the device can still function properly. It also specifies the characteristics that the unintended signal or signals such that a proper test can be taken. For example, if testing a wireless device that uses a WIFI signal, Zigbee signals or Bluetooth signals cannot be used as unintended signals (according to the standard) because it has been shown that these signals have little to no interference with a WIFI signal: a separate WIFI signal needs to be used to test the intended WIFI signal. This standard also helps to define how to push the system to the point of failure so that the failure region can also be identified.

4.1.2.2 ANSI C63.18 RF Emission On-Site Evaluation

The full name of this standard is the "American National standard Recommended Practice for an On-Site, Ad Hoc Test Method for Estimating Electromagnetic Immunity of Medical Devices to Radiated Radio-Frequency (RF) Emissions from RF Transmitters." This standard provides an on-site (non-laboratorial) way of testing medical devices to other RF signals and was not made to replace the more rigorous laboratory EMC testing described in IEC 60601-1-2. This standard explains how to set up the test, what kind of RF transmitters to be used for testing, distance from transmitter from device, and variability control. These tests are conducted to make sure that an electronic device (such as the one the team is designing) can operate correctly even while receiving relatively powerful RF signals (relative to WIFI signals). This kind of testing is essential in hospitals where medical devices are exposed to other kinds of power RF signals incidentally: a perfect example of this would be in an emergency rom or in a patient room.

4.1.2.3 IEEE 802.11n-2009

The 802.11n standard was introduced to replace the outdated 802.11a and 802.11g with its main feature being the increase in maximum data rate (54 Mbit/s to 600 Mbit/s). This standard includes the hardware specification needed for wireless communication hardware such that the new technology can be implemented on it. The main new specification for the new hardware was the addition of MIMO technology which stands for multiple-input multiple-output, channels of 40 MHz to the physical layer, and multiple frame in one transmission (frame aggregation) to the MAC layer. This standard is backwards compatible with 802.11a and 802.11g, and newer standards there were developed post-2009 are compatible with 802.11n

4.1.2.4 BLE v4.2 Specifications

Although once standardized by IEEE (IEEE 802.15.1), the standard is no longer kept by IEEE. Instead, the specification for Bluetooth technology is kept by the Bluetooth Special Interest Group (SIG); although not explicitly called a standard, these specifications serve as qualifications for both communication protocols and technology requirements for hardware to be able to operate using BLE v4.2. This cover all the communication specifics including all the layers of the architecture. A comprehensive list of material can be found on their website; they also include qualification test requirements and specifications in development.

4.1.3 Logic Standards

Computers work in a binary state, having a state that is logic 0 and another that is logic 1. Different technologies can distinguish the states by using electrical properties of semiconductor materials. The two discussed standards will be CMOS and TTL, two voltage levels and thresholds that will play a critical part in module compatibility between the MCUs and components.

4.1.3.1 TTL Logic

Transistor-Transistol Logic was the standard that was mostly used when Bipolar Junction Treansistors were the primary source of logic components. TTL needed many more transistor in order to accomplish the same things that current Field Effect Transistors can do more efficiently. The primary difference in the logic difference is that the “low” or “zero” state could be read in a range between 0V to 0.7V, and the “high” or “one” could be read between 2V and 5V. The range when outputting is slightly narrower with “zero” being 0V to 0.5V and “high” is between 2.7V and 5V [35].

4.1.3.2 CMOS Logic

CMOS stands for Complementary Metal Oxide Semiconductor, and is the standard that followed FET devices. While BJTs are minority carriers and are current controlled, FETs, or field-effect transistors, are majority carriers and are voltage controlled devices. This means that the power consumption is more efficient and can require fewer components when compared to BJTs. The standard is more variable when it comes to the value for the “high” state with the lowest range centralizing around 5V and the highest centralizing around 15V.

When dealing with circuits and microcontrollers, the low value when reading is 0V to 1.5V and the high value is from 3.5V to 5V. When outputting, the range dranstically diminishes, with a low values from 0V to 0.05V, and high values of 4.95V to 5V [35].

4.1.4 Design impact of relevant standards

As explained in previous sections, standards must be considered for many reasons such as safety, compliance, compatibility, usefulness, ease of use, etc. These standards directly affect the design of a designer: therefore, good knowledge and preparation research is needed to design a device properly. Based on the relevant standards presented on the previous section, certain parameters and aspects had to be considered in our design. This section will detail how the standards affected our design.

4.1.4.1 Impact of ANSI C63.27

This standard had a direct effect on the WIFI chip that the group researched to buy. A major requirement when searching for a chip was to ensure that the chip could be found in module form with a pre-made PCB trace antenna or a integrated chip antenna. This gives us the security that the chip has an RF antenna that has been optimally designed by its manufacturer, and therefore will more likely meet this standard that deals with unintended or interference signal effect evaluation.

This standard also provides a blueprint for the testing of our device when it comes to WIFI/RF signal reception and possible transmission. This standard will be one of the main influencers for the way we test our device.

4.1.4.2 Impact of ANSI C63.18

This standard had a direct impact on the actual design of the product PCB. This standard provides a blueprint and a testing “mark” that electronic devices must meet to be useable inside a hospital. This standard deals with the testing of relatively powerful RF signals emitting on a device and a device still being able to function properly. This made us consider how to properly shield the electronics on the PCB (especially components dealing with RF reception and transmission) while still providing ample operating space for the antenna of the WIFI module and the RFID tag to operate properly.

This also effected the way we tested our PCB. Not only should we consider the RF interference at the RF antennas but the interference with the actual device electronics. This will serve as a basis for testing of our final product.

4.1.4.3 Impact of IEEE 802.11n & BLE v4.2 Specifications

These standards had just a mild impact on the device design. The focus when considering these standards was to make sure the WIFI or BT module that we were considering and buying had compatibility with these standards. Since we are not developing our own WIFI or BT chip, the bulk of these standards (physical layers, hardware specifications) are not used.

4.1.4.4 Impact of Logic Levels

The key effect of the different logic levels is finding the intercompatilbilty between the microcontrollers and peripherals. Almost all devices are made with CMOS technology because of the compact footprint, reduced componenet count, and power savings. Howevers, many CMOS devices often use the TTL voltage ranges both for backwards compaitibility with previous designs, and also because the lower voltage serves to save power with the high switching devices.

The logic levels play an immense part with compatibility of design. Choosing incompatible logic levels could require more hardware to regain compatibility between devices; inquring a higher footprint size and higher costs.

4.2 Realistic Design Constraints

Any engineering design must meet or face many types of real world constraints in its design process. These types of constraints are economic, time, environmental, social, ethical, safety, and particularity in our case health constraints related to a medical setting. Below are short discussions on how these types of constraints will be dealt with

4.2.1 Economic and Time constraints

A major constraint for this project is budget, this project consists of about roughly 80% semiconductor chips. Although modern semiconductor chips are very cheap their respective development boards for testing are not, this makes a rather difficult situation in testing such a complex system to design without using pre-manufactured breakout boards or sensor boards within our limited time frame. We must choose the main essential components for which we spend a little more funds on to test. Our essential components consist of the microcontroller (MCU), pulse oximeter sensor, battery, and Wi-Fi – all other components are secondary to these main components functions and therefore we must obtain a means of development testing even if a development board from its respective manufacturer is outside of our budget. Another major constraint is the time and cost of PCB making, during the design process mistakes can be made on the soldering and another PCB will have to be used – which is expensive and time consuming. Time overall is a big constraint generally for us doing this project due to the fact that taking senior design one in the summer is very condensed compared to the fall semester. The summer senior design required us to be diligent in researching our components and especially ordering them on time. On top of that we had to test each component to make sure it properly worked for us, which included ordering extra breakout boards, development boards, and learning how to solder very precisely since our components are extremely small. The time constraint could be very stressful at times considering we were learning how to implement certain technologies we have never used before, but luckily there is a lot of information on the web to help us in our endeavor.

4.2.2 Environmental, Social, and Political constraints

One of the main constraints of our device is being in the environment of the hospital setting. In the hospital environment the need for certain health and privacy standards are very necessary when it comes down to the patient. Also making sure our device does not interfere with the other medical machines is a very big constraint.

A social constraint to be considered is the fact that someone may not want their exact location to be monitored at all times while in the hospital or even the fact that their information is stored on an RFID/NFC and could be stolen, even though the odds are very low of that.

There are not too many political constraints when it comes to our product, but one to be considered is if a hospital wants to purchase our product, but simply does not have enough money due to budget cuts.

Workarounds could include logic level shifters that can bring up the voltage, or simple pullup resistors. There even exist ICs that can solve the problem of shifting, without the need of many external components.

4.2.3 Ethical, Health, and Safety constraints

Since the application of our device is in a hospital setting, ethical, health, and safety constraints are extremely important.   These constraints will be explained in the following subsections.

4.2.3.1 HIPAA

Our project is intended to be in a medical setting, specifically a hospital. There are strict guidelines in HIPAA compliance that covers Protected health information (PHI). PHI under the US law is any medical record or medical information that can be traced back or associated to a specific individual [36]. Since our design will contain a RFID tag associated with a particular subject or patient (and important medical information stored) and that tag will be associated with the device that contains location and heart rate, we will need to comply carefully with these standards.

The Protected health information rules within HIPAA are defined as “individually identifiable health information” that is stored and sent over any medium. Specific the information referred to by the US Health & Human Services are [37]:

- Individual’s past, present, or future physical or mental health or condition
- The provision of health care to the individual
- The past, present, or future payment for the provision of health care to the individual, and that identifies the individual or for which there is a reasonable basis to believe can be used to identify the individual. Protected health info. (e.g. name address, birth date, Social Security Number)

We will be storing sensitive information associated with a RFID tag tied to the device via fields entered in a nearby device (Android/Laptop). We will have to take care to observe these guidelines.

4.2.3.1.1 Hospital Information on Patients

Patient health information is extremely serious to hospitals; the management, transmission, and storage of this information is therefore also very important. The major authority governing this aspect of patient information is commonly referred to as HIPAA (Health Insurance Portability and Accountability). This standard or set of rules and regulation were made law in 1996 and are enforced strictly. Specifically, implementation requirements for wireless transmission or communication of patient health data will not be found in the actual HIPAA act (HIPAA was enacted pre-internet of things); the information will actually be found in the Code of Federal Regulations Title 45, Part 14, Subpart C. In the subsections to follow, different aspects of these codes will be explored [37].

5 Project Hardware and Software Design Details

In the real world, if two different designers are given the same components and the same design goal, both designs would be different; this is to say that when it comes to designing, although a lot does depend on proven design methods, some of the design (if not a lot of it) is also left up to the designer to decide. It is imperative to understand that great parts with a poor design makes for a below average device, and this is why much though and detail needs to be used when designing system. In almost every system in work today, there are two major aspects: hardware and software. Hardware provides the physical platform on which the software can run and implement the functions the designer intends to do with the overall product. This section will focus on the details that went into designing the hardware and software of our system. The following subsections will focus on the overall system but also go into detail with each specific subsystem. Each subsystem will focus on one major technology involved in the design of the overall system or device. Underneath each subsystem details of specific systems underneath it will also be explained and detailed.

5.10 Initial Design Architectures and Related Diagrams

The overall system consists of at least 6 total subsystems (4 key subsystems and 2 secondary subsystems). The following systems to come will expand on the 4 key subsystems; this is because the 2 subsystems are not actual entire systems but rather just an additional chip that will interfaced, and, as explained earlier, they are not vital to the function of the overall system. The following diagram is similar to the overall block diagram seen in section 2.6 but not updated with the final parts the group has decided to use. This is the hardware over view of our system:

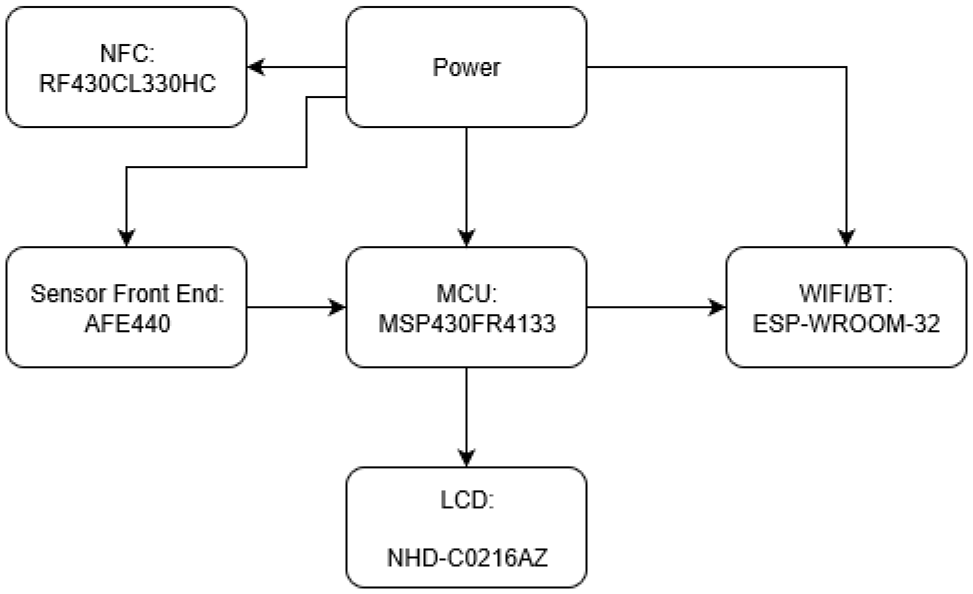


Figure 42. Hardware overview

Hardware is just only half of the project: the other half is the software. The software or code to be implemented in this project is complex but tthe following flowchart shows a general description of the software to be implemented in this project.

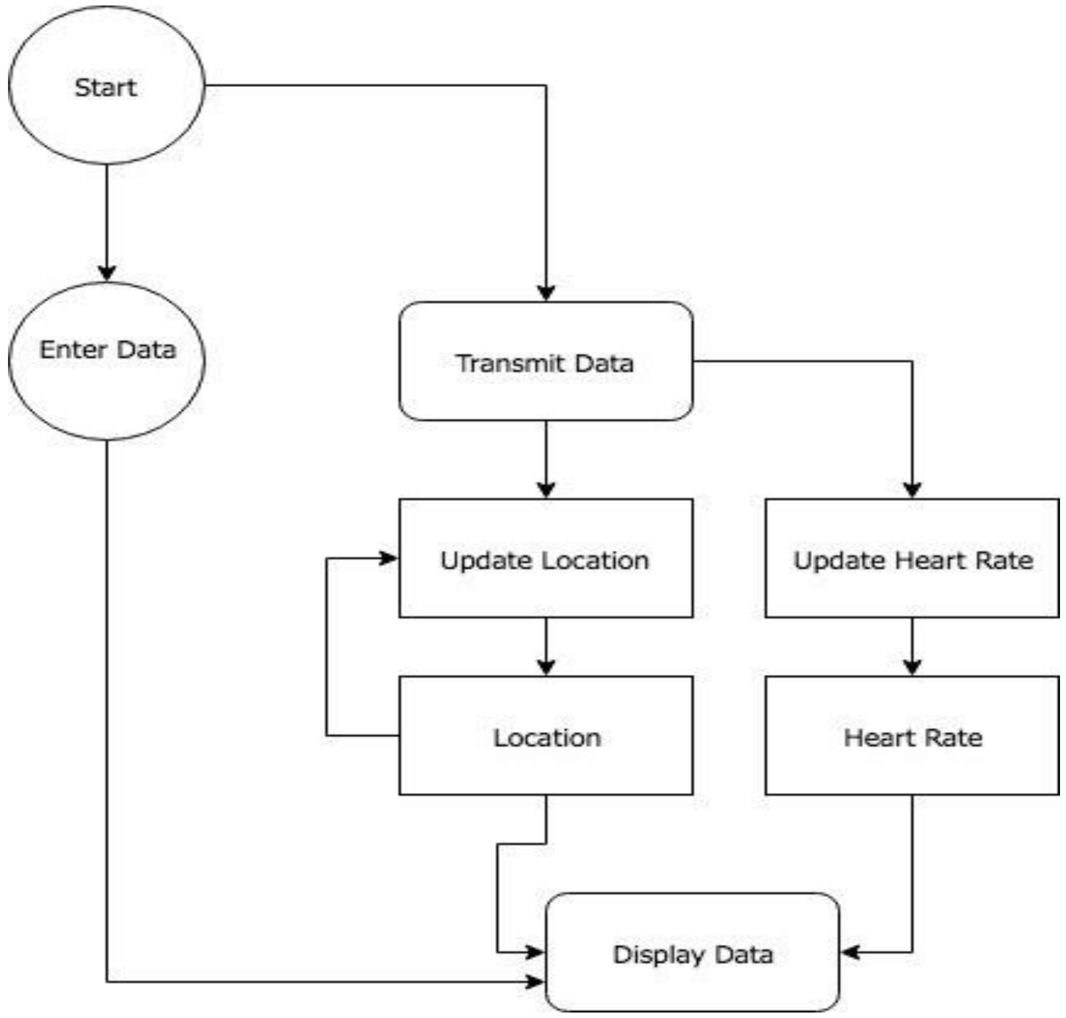


Figure 43. Flowchart of software

This flowchart shows that the software code written on the MCU will have three main tasks: display data, monitor data, and transmit data. The data will be displayed to an LCD screen; the data to be displayed on here will be some of the patient health information from the sensors such as heart rate. The data to be monitored will be the sensor (health) data and the localization data. This will be monitored and updated at interval times to save battery life of the MCU. All this data, both the sensor and the location data, will be transmitted to the hospital communication network. The description and block diagrams associated with these three main functions of the code programmed on the MCU will be explained in sections to come.

5.11 First Subsystem, Breadboard Test, and Schematics

The first subsystem of the watch pertains to the microcontroller. The microcontroller is the center of the design, connecting the logic of all other systems. The chosen MCU has 4 dedicated pins, two for power and two for crystals. The other sixty pins will be dedicated to the LCD and other peripherals. An important section of the microcontroller subsystem is the serial communication buses, particularly the SPI and the I2C buses.

Due to the very small nature of the microcontroller, the initial testing that took place was based on the development board. The benefit of the EXP-MSP430FR4133 is that it had an LCD built in, and therefore expedited the LCD testing even before we settled on a display.

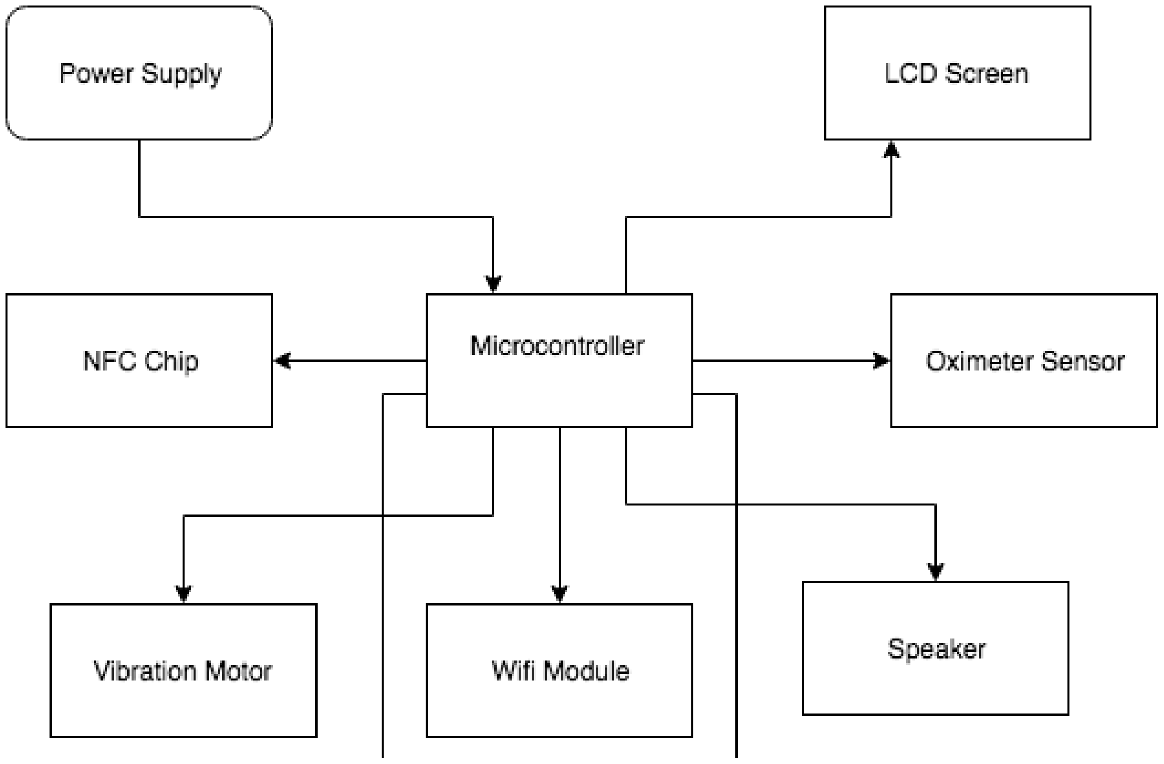


Figure 44. First subsystem-MCU

5.12 Second Subsystem

The second subsystem will be the analog front-end that connects the DB9 pulse oximeter sensor.

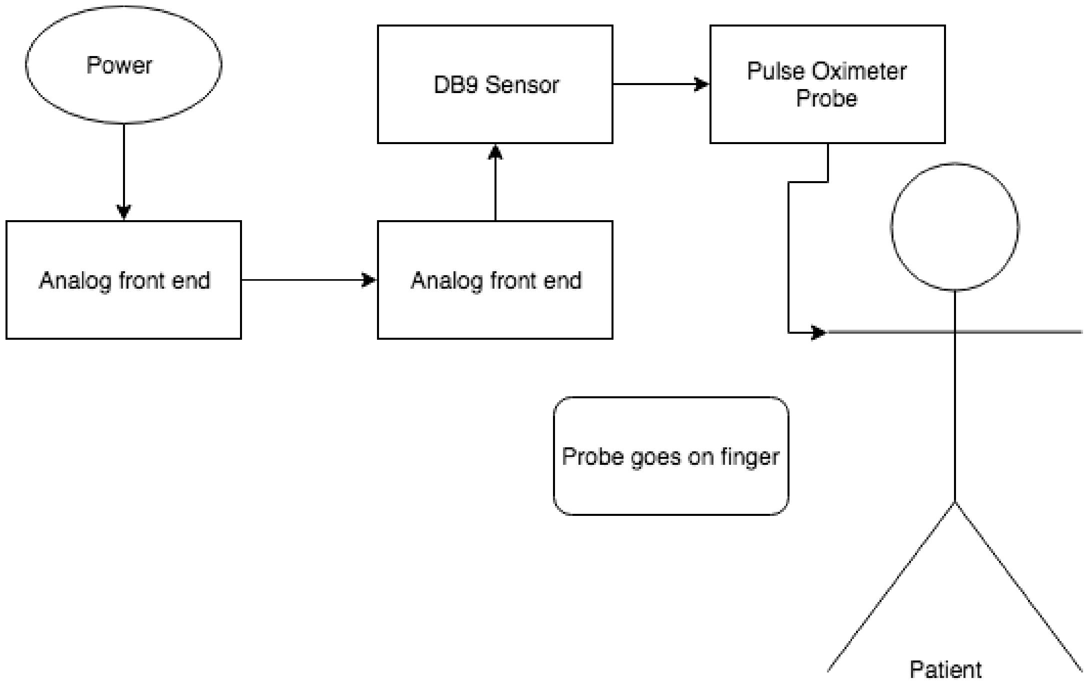


Figure 45. Second Subsystem Overall Diagram

5.13 Third Subsystem

The third subsystem will be the portion of our device that is power. This power subsystem mainly consists of voltage regulators which will be used to interface the power to the chips as shown in Figure 38. There will two separate parts of the power system. One will be hosted on the wearable device: this is the one mentioned above. There will also be another separate power system that will be responsible for re-charging the battery. This separate power system will be an external device that will be made in addition to the wearable device. This system will be composed of several more complex chips in addition to voltage regulations. Things that will be kept in mind when designing this charging power system are the following: voltage regulation, current limiter, and smart charging chip. This latter power system will be of very careful thinking since recharging a batter can be very dangerous especially if the recharging is done at a high current.

The latter system of recharging the battery has not be thoroughly planned or designed since this aspect of the project is not a main part but rather a stretch goal. Nonetheless, if this wearable health device were to be manufactured and sold, this part of the system will have to be developed. Once the actual implementation of the device is done, this will be one of the very first stretch goals that will be attempted to reach; but in the meantime, just a few thoughts of “how” are documented and ready to be explored later on. This paper is subject to future correction or updates since this project becomes more and more “concrete” as testing and implementation is done concerning this project.

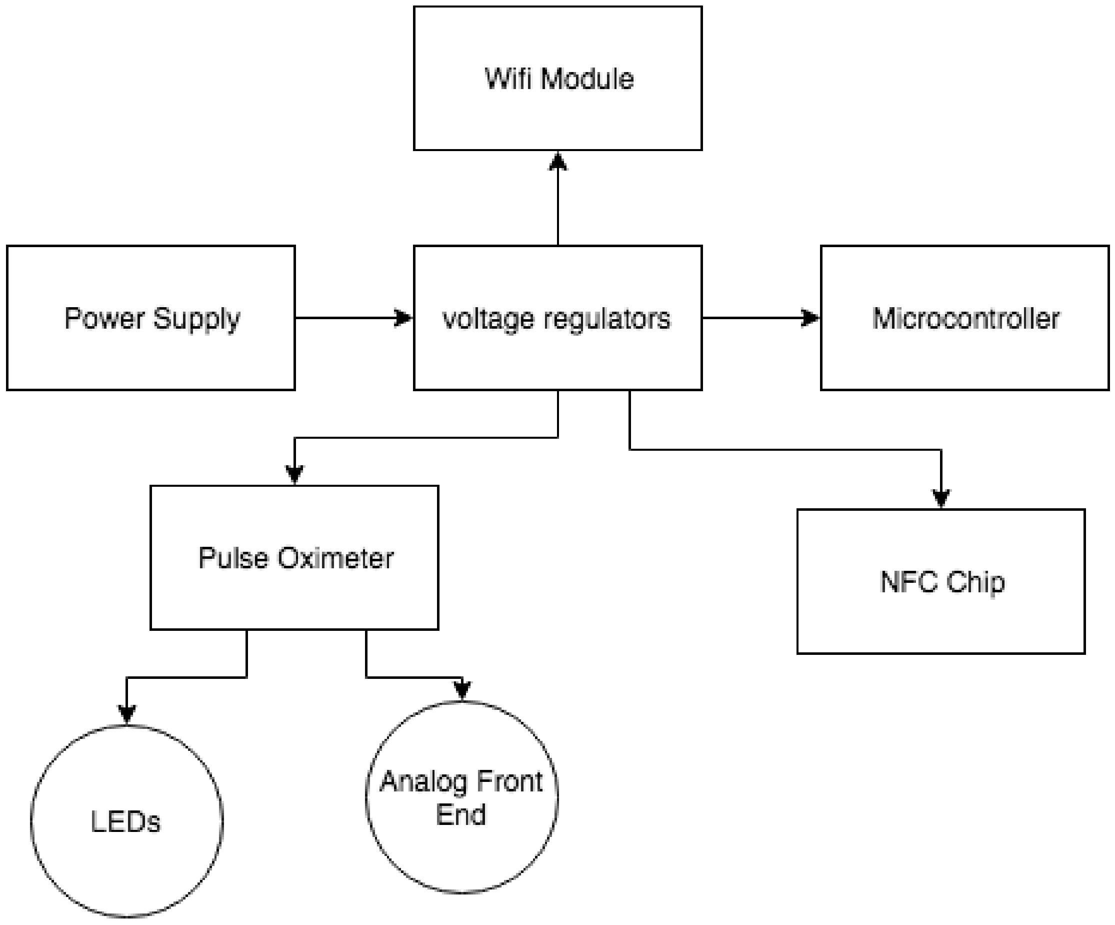


Figure 46. Third subsystem overview

5.14 Fourth Subsystem

The fourth subsystem is the one in charge of wireless connectivity through WIFI and possibly Bluetooth. A block diagram for this subsystem is not needed since all of the parts of this subsystem is contained in the chip module itself. Since we will not mess with the chip itself (no modification to the actual chip will be made) having a block diagram of the chips itself would also not be beneficial. This chip module includes the ESP32 Esspressif chip which is a dual core WIFI/BT compatible chip and incorporates a PCB trace antenna among other things. This makes this chip a system-on-chip module. this chip will be the gateway of communication to the outside world but will also be the biggest consumer of current.

5.15 Software Design

The software design of any project is critical: it's what gives life to a physical design once the proper parts or components have been put in place. Without proper software design, even chips or components that have been designed to operate in low power consumption can end up consuming a lot of power. The software design of this project is just as important as the hardware design. There will be two main programs in the implementation of the software and coding of this device. One part will be run by the wearable device itself using the main MCU and the MCUs included in the WIFI/BT chip, and the other part will be run by a separate server which in its most simple and crude implementation could just be a software on a separate computer. Because one of the goals of this project is to make it as low power consuming as possible, the code that will be run on the MCUs will primarily be to monitor the data from the sensors and WIFI/BT chip. The calculations and logging of data will be done by a separate server (which can be just a separate computer) that will communicate with the device over WIFI.

5.15.1 Patient info Read/Written via NFC

Due to the HIPPA regulations regarding safe wireless communications, personally identifiable information must be transferred via Wi-Fi. This is due to the fact that NFC does not have any built in encryption and safety measures like WIFI does. Also there is risk of eavesdropping, where a malicious party can be a short distance away from the NFC device and still intercept the data transfer [38]. To circumvent this issue, our solution will be to send an encrypted patient ID via NFC, and then download the patients data to the device using secure WIFI. A similar idea will happen when reading the watch with an NFC reader, except that the encrypted ID will be sent to the reader and the reader will download the info via WIFI. Although this does add additional "response time" of this system, in this case, the safety and security of the patient information takes priority although this additional time will not be particularly large. An extreme scenario where this system might fail is when the WIFI of the hospital system is down; in this case, this NFC reading and WIFI downloading process or system will not work. This system failure would occur under two general scenarios in a hospital: 1) There is a power outage at the hospital, and as a result, the WIFI system cannot operate or 2) The WIFI system begins to experience issues to the point that it cannot operate correctly. In either case, both of these major scenarios can be considered as a highly unlikely events that would probably rarely occur at a hospital; this is because many hospitals if not all hospitals in the US have backup generators that are designed to sustain the hospital when there is a local power outage in the area (although there are documented cases of generators failing, the majority have operated correctly). Also, in case of a WIFI system bug or issue that does not allow the network to function properly, a support group which can help fix the system is just a call away. For the second case, it would be interesting for a stretch goal to research into portable, secure, HIPAA approved WIFI systems that can be momentarily used while the main system of the hospital is restored to its original functionality. Nonetheless, below is a block diagram which demonstrates the software operation of the NFC system.

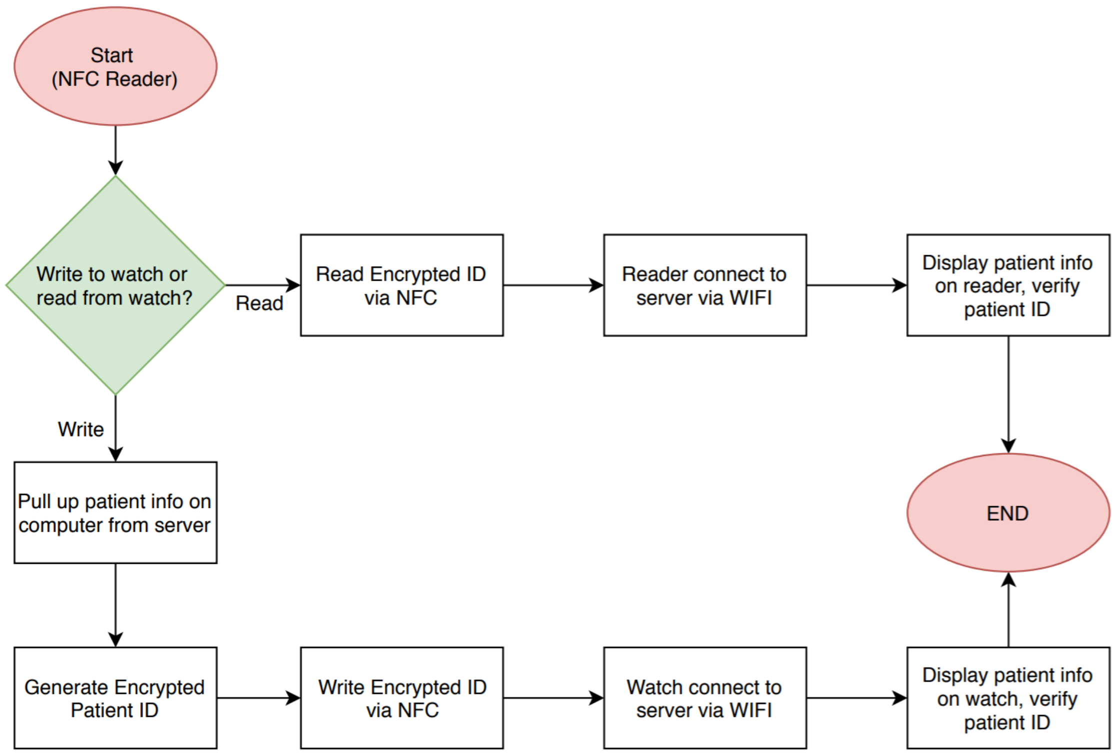

Start
(NFC Reader)
Write to watch or read from watch?
Read
Read Encrypted ID via NFC
Reader connect to server via WIFI
Display patient info on reader, verify patient ID
END
Write
Pull up patient info on computer from server
Generate Encrypted Patient ID
Write Encrypted ID via NFC
Watch connect to server via WIFI
Display patient info on watch, verify patient ID

Figure 47. NFC Software Flowchart

5.15.2 Mapping

The localization of the patient is one of the primary goals of the watch. It allows for a second pair of eyes, making sure that patients do not get lost or are not taken to the wrong location (like an ER). In order to localize the patient, we will be using WIFI trilateration in combination with probabilistic mapping.

There are three major sections that will compose the mapping and localization. The first section will be the initial mapping of the hospital grounds, and setting up a server with the map data. The second step will be watch-side localization via WIFI, which will test the connections and give the raw data to the server. The final section will involve taking the raw data provided by the watch, determining the probabilistic location of the patient, and alerting the patient if they are where they should not be.

5.15.2.1 Initial Mapping and Server Setup

Many robotics systems have the problem of imprecise movement when running for a long time. Part of the issues is not having a feedback telling the robotic system where it is. More complex systems can use radar and other sensors to survey the surroundings, and using this information to build a map and localize the robot simultaneously, a process known as SLAM (Simultaneous Localization And Mapping). This is crucial when exploring unknown environments. However, maps are much more effective when the environment is known. The first step of the project is to build the map.

The map can be based on existing blueprints of the building, but they must be digitized and trained to know where the wireless routers are located. Once the router locations are included on the map, a server can estimate where a patient will be depending on the strength of three wireless signals, or even learn patterns and probabilities of how patients can travel. Because Wi-Fi strength is fluctuating and prone to disturbances, the server will need to analyze the data and find probability fields of where the patient could be. The raw data will come from the next step, watch-side localization.

5.15.2.2 Watch-side localization

A little contrary to the title of the section, watch-side localization does not allow the watch to independently figure out where the patient is located. The watch is only able to accrue the raw data of the strength of the routers. This raw data can either be averaged or transmitted in-full to the server. The watch could "technically" locate a patient independent of a server, but this would involve complex algorithm processing on the watch along with keeping a database of a plethora of access points on the memory of the MCU. Not only would there be a concern for memory storage on a device as small as the MCU but also, and most importantly, there would be power limitations and issues. There is no reason to delve into trying to accomplish this goal just on the watch when communication with a server or external computer can be made possible over WIFI because of the WIFI/BT chip incorporated on this design. The server will be much more powerful than the watches and will not be hindered by limited battery power.

The watch will be programmed to wake up from a low power mode, either due to movement or an elapsed amount of time. The watch will either then connect or remain connected (there is a low power mode option on the WIFI chip which enables it to enter a low power mode but remain connected to a WIFI network) to the strongest AP signal while simultaneously scanning other AP signals. It will record the strength of each AP signal for a set amount of time, T, or for a certain amount of measurements, M. Increasing either T or M could increase the precision of the final result but would also mean that more power would be consumed. Testing for an ideal amount will be discussed in a later section since a balance between these two parameter aspects will need to be found. After the strongest signal is measured, the second strongest will be measured again. This will be repeated a third time to allow trilateration (a minimum of three AP signals is needed to implement trilateration.)

The raw data is sent to the server which will process it and determine if the patient is where they are expected. This will require additional integration with the hospital system already in place (in actual testing this will need to be mocked or simulated using an external software since the actual hospital software with patient schedules will probably not be available to us under HIPAA). In order to make sure there is proper communication, an "ok" signal will be sent from the server to the watch when there has been proper transmission and reception. Until the watch receives this signal, it will continue to attempt to transmit the information or new position information collected. Below is a software flow chart from the "watch" perspective.

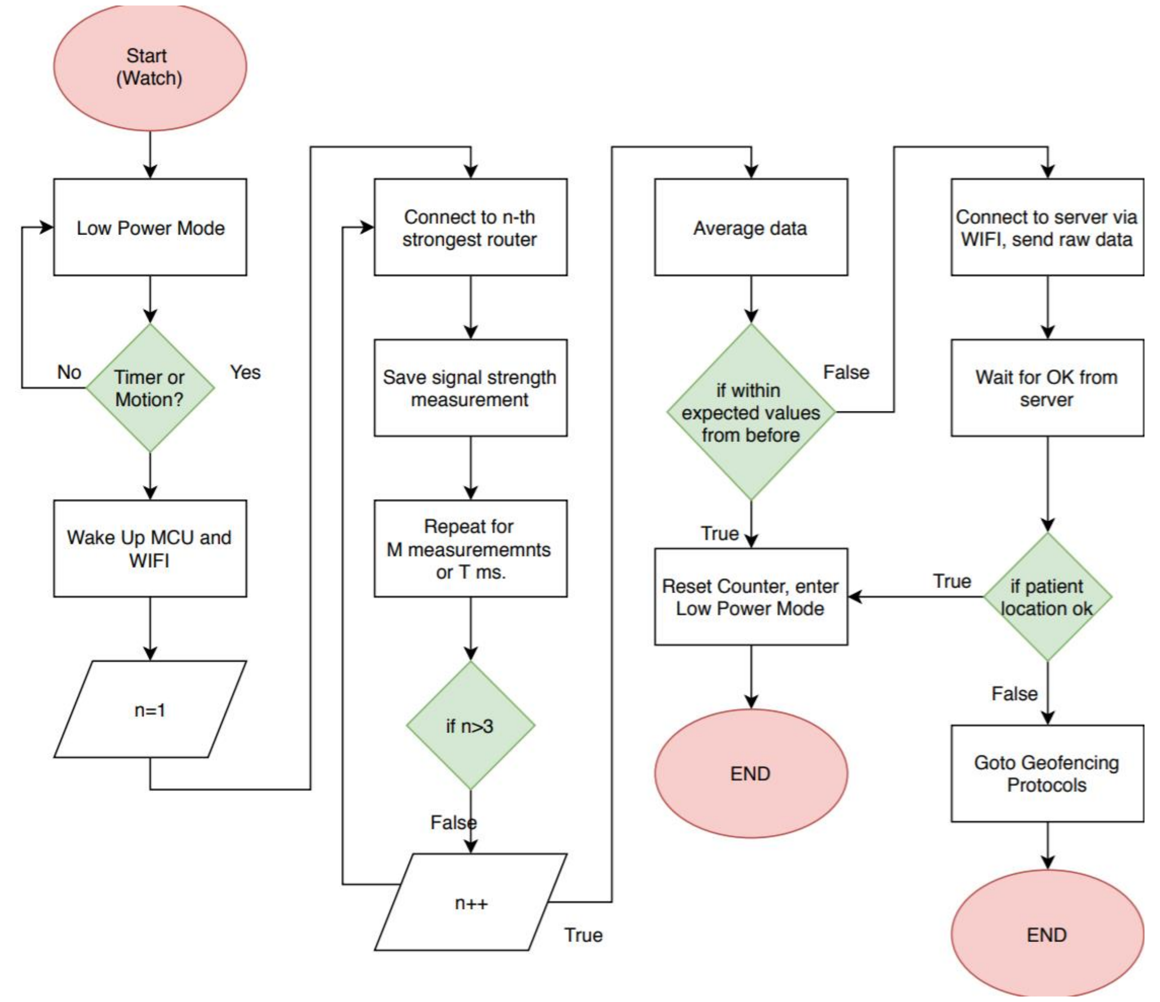

Start (Watch)
Low Power Mode
Timer or Motion?
No
Yes
Wake Up MCU and WIFI
n=1
Connect to n-th strongest router
Save signal strength measurement
Repeat for M measurememnts or T ms.
if n>3
False
n++
True
Average data
if within expected values from before
False
True
Reset Counter, enter Low Power Mode
END
Connect to server via WIFI, send raw data
Wait for OK from server
if patient location ok
True
False
Goto Geofencing Protocols
END

Figure 48. Acquisition of data by watch for localization

Once the server receives the information, it needs to process it and determine location. From this server, this information will be available to authorized devices for accessing. A main computer can easily access this information and from there additional software can be coded to implement a map or even make the information available for hospital systems and software that are already being used. One big implementation of this would be in geofencing which will be discussed in a later section. If this information could be made available to a patient's schedule that is already recorded on a system, additional software can be implemented to raise alerts when a patient's location does not match his itinerary.

In order for the server to localize the patient, it will receive the RSSIs and MACs data from the watch, run two main algorithms, store this information and make it accessible to authorized network devices. The following block diagram shows the data processes that would need to be implemented by the hospital server in order to localize a patient:

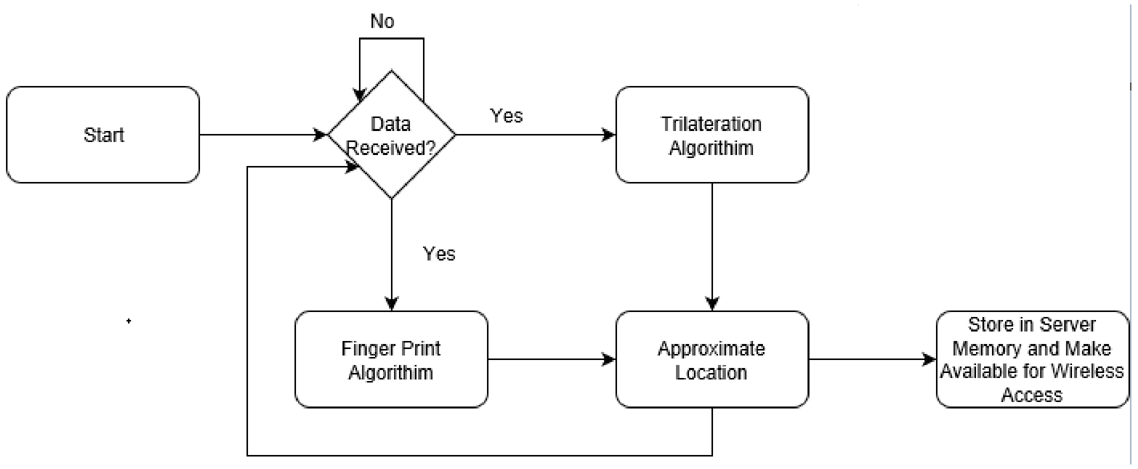
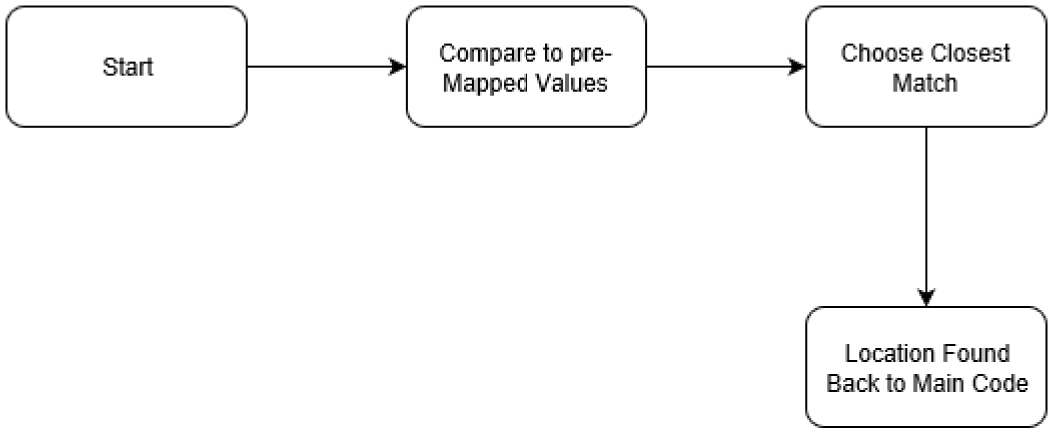


Figure 49. Flowchart of location data processing by server

The finger print and trilateration algorithm are two separate ways for implementing indoor localization. The combination of both can increase accuracy. Finger printing has a "setup" stage where strategic positions are mapped according to the RSSI of the surrounding APs and attributed a physical position. Then, these values of RSSI are used to compare to the received data from the watch and the closest location is picked. This is ideal because it is very immune to signal attenuation due to walls or objects. Trilateration uses the RSSIs received from at least 3 APs, applies converts them to physical distances through empirical formulas, and finds the meeting point of these distances (each distance is a measurement of radius from its respective AP). This method is self-adaptive to environmental changes but is subject to bigger issues stemming from signal attenuation because of objects or walls. The following are block diagrams exemplifying them both.



Start
Compare to pre-Mapped Values
Choose Closest Match
Location Found Back to Main Code

Figure 50. Flowchart of finger printing algorithm

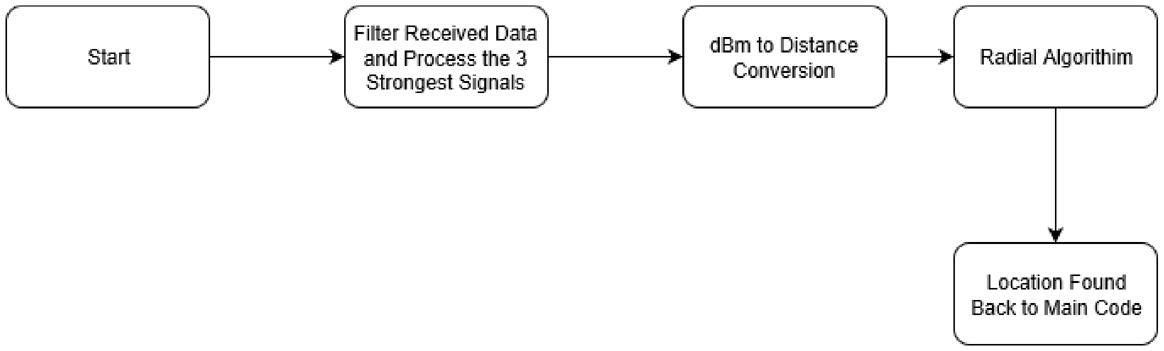

Start
Filter Received Data and Process the 3 Strongest Signals
dBm to Distance Conversion
Radial Algorithim
Location Found Back to Main Code

Figure 51. Flowchart of trilateration algorithm

5.15.2.3 Geo-fencing

The final section of the mapping would be the geo-fencing. Geo-fencing is a way to limit where a person can be in without needing physical fencing. This is a key goal to keep the patients out of where they should not be going, like for example an operating room when they should be getting an x-ray. The watch-side localization is able to estimate the position of the patient after communicating with the server once. (By comparing the average values to the expected values of the wireless strength.) But how does the server know where the watch and patient are?

Combining the initial mapping with the watch-side mapping, the server is able to employ statistical analysis to determine where the patient could be. Due to the more imprecise nature of WIFI, a set of probability clouds must be used to estimate the location of the patient. This allows a refining of the raw data provided from the watch, and then allows this data to be imposed over the map.

The next step of the server is to determine what locations the patient is allowed to venture into. This is accomplished by providing the server with critical information on the patient. These would include the patients intended room when not being tested, which would be the default expected location. Then hallways and restrooms would be considered accessible areas. The next data that the server would receive would be of any upcoming or ordered test and procedures. That way, if a patient is scheduled to go to the x-ray room, then the server would expand the available space of the patient ot include the room.

The key point here would be that when certain areas on the map are accessed, the server can tell the watch whether it should track the patient more precisely by increasing M or T or alert the patient by putting the watch in high alert mode. In this mode, the watch would vibrate and beep, making it clear to the patient and any transporters/nurses near them.

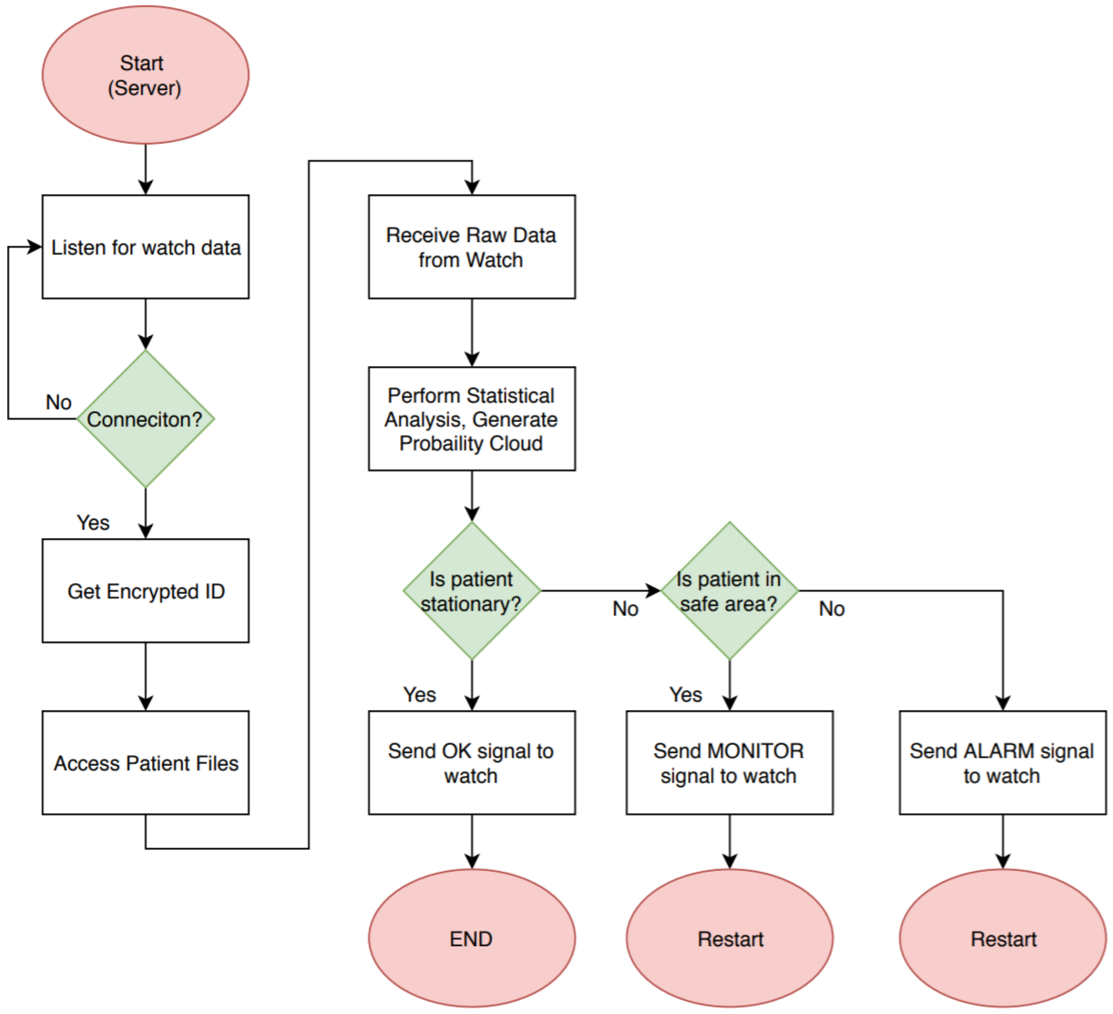


Figure 52. Software flowchart of geo-fencing

5.15.3 Panic Mode

Hospitals are full of patients that are not at their prime. Patients normally arrive at a hospital due to an illness or ailment that they seek medical attention for. Hospitals expect that the patients can possibly need emergency assistance even within the hospital. A common emergency signal implemented in hospital bathrooms are a rope on the wall that will notify the nurses. These come in handy when a patient is unable to stand up after falling in the restroom. These rooms are often more isolated, and yelling might not be enough to alert the patient.

Our system will include a similar idea of an emergency signal. The design will include a few buttons to change between the different displays. When both buttons are pressed at the same time for more than a couple of seconds, the device will go into alarm mode. In this mode, the watch will do a few things in order to get attention to the patient as soon as possible.

The first step will be to turn on the speaker and let out an auditory alert that the patient is in distress. This would not be a very loud alert due to the size and cost constraints of the design, but it could allow for any hospital staff in the area to approximate the location of the patient through sound.

Another alert system that will be included will be the vibration motor. This would allow deaf and hard of hearing patients to be able to tell that the emergency mode is on. It also helps to notify patients that might have accidentally pressed the combination, by showing them that the sounds are coming from their wrist.

The most important part of the emergency response would be the ability to locate the patient as soon as possible using the localization techniques used above. When the system goes into emergency alert mode, the watch will tell the server that the patient is in need of help. The server would immediately send the information to the nurses in the area so that they might find the patient quickly. Another similar response to the emergency response would be an automatic MIA response. If the watch misses too many scheduled communications with the server, then the patient can be assumed missing an a nurse can be sent to check where the patient was last seen.

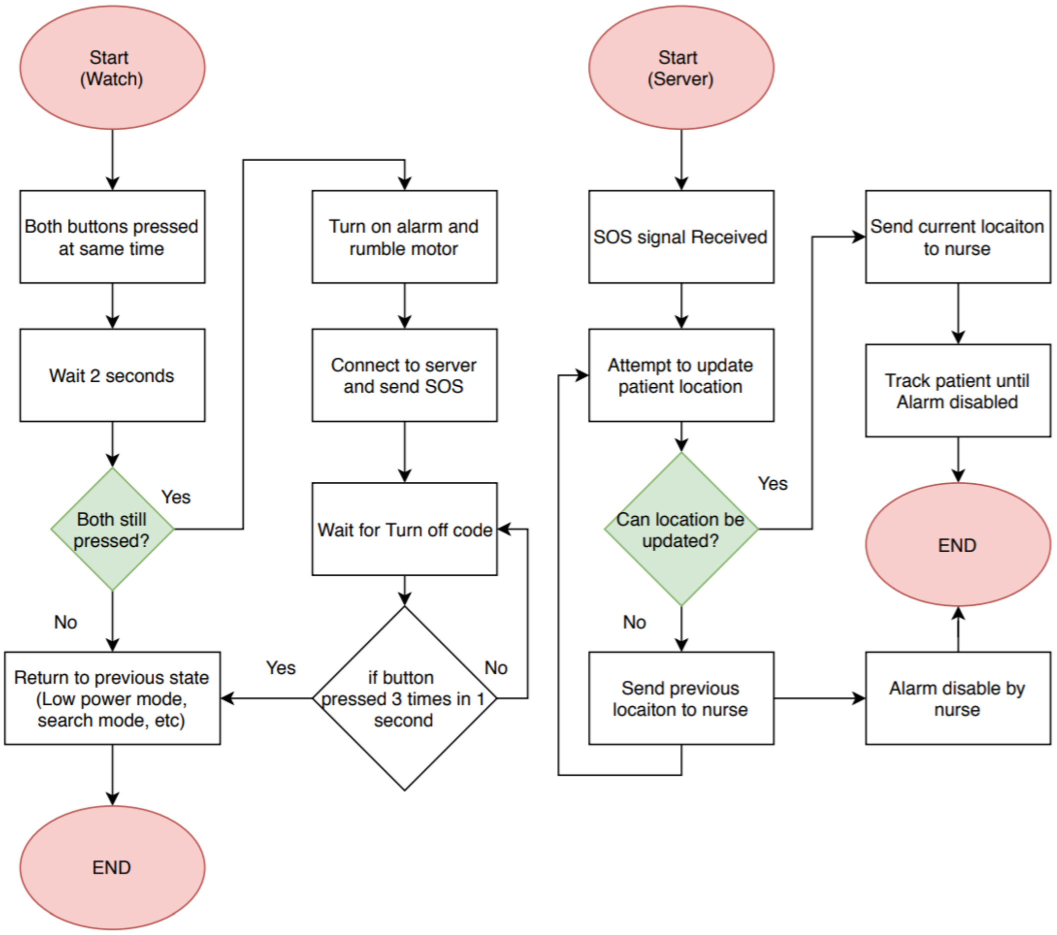

Start (Watch)
Both buttons pressed at same time
Wait 2 seconds
Both still pressed?
Yes
No
Return to previous state (Low power mode, search mode, etc)
END
Turn on alarm and rumble motor
Connect to server and send SOS
Wait for Turn off code
if button pressed 3 times in 1 second
Yes
No
Start (Server)
SOS signal Received
Attempt to update patient location
Can location be updated?
Yes
No
Send previous locaiton to nurse
Send current locaiton to nurse
Track patient until Alarm disabled
END
Alarm disable by nurse

Figure 53. Software flowchart of panic mode

5.15.4 Watch (Real Time Clock)

Arguably the most important part of the smart watch is the ability to display the time. Every single of the discussed MCUs has a low power mode with a 32KHz clock, which allows for a real time clock. This allows for an overflow flag of a 16 bit register every second, which can be used to update the clock from low power mode.

5.15.5 Patient Info on Display

An important point to the watch is the ability to display crucial information easily and clearly. Some of the most important information that must be displayed is the identity of the patient, their allergens, and their heart rate. The character display would provide a simple environment for design, allowing for writing of words to be very simple. The limitations occur due to the very simplistic design of the display not allowing for clear visuals without some creative programming. The black and white OLED screens will allow for more control of what can be drawn on the screen but it will be more complex to program, even using a visual frontend. The bottom images show the design of the menus using the character display with a character resolution of 16 characters over 2 lines, with each character having a resolution of 5 by 8 dots.

L	A	S	T	N	A	M	E		|	B	P
F	I	R	S	T					|1	0	0

Figure 54. Main Screen of Watch

1	2	:	4	5	:	1	2	P	|	B	P
N	O	V	1	4	,	1	7		|1	0	0

Figure 55. Time Screen of Watch

5.15.6 Power Saving Modes

Every mAh used by the design is crucial when designing a mobile system. A great way to minimize the battery usage of the device will be to use the low power modes of the design. The current drawn from the microcontroller when the low power mode os on can reduced to the level of nano amps, while still allowing for interrupts and timers to wake up the device. Similarly, the wireless module can be put into low power modes where the power drawn more than halves, and an ultra power save mode that saves even more. The downside of these power modes is that they often require some deleay to ramp up the system back to operating speeds.

These will be some of the software designs in order to reduce the power consumption of the device. The variables will need to be played with in order to find the happy moddle ground between power efficient and usable.

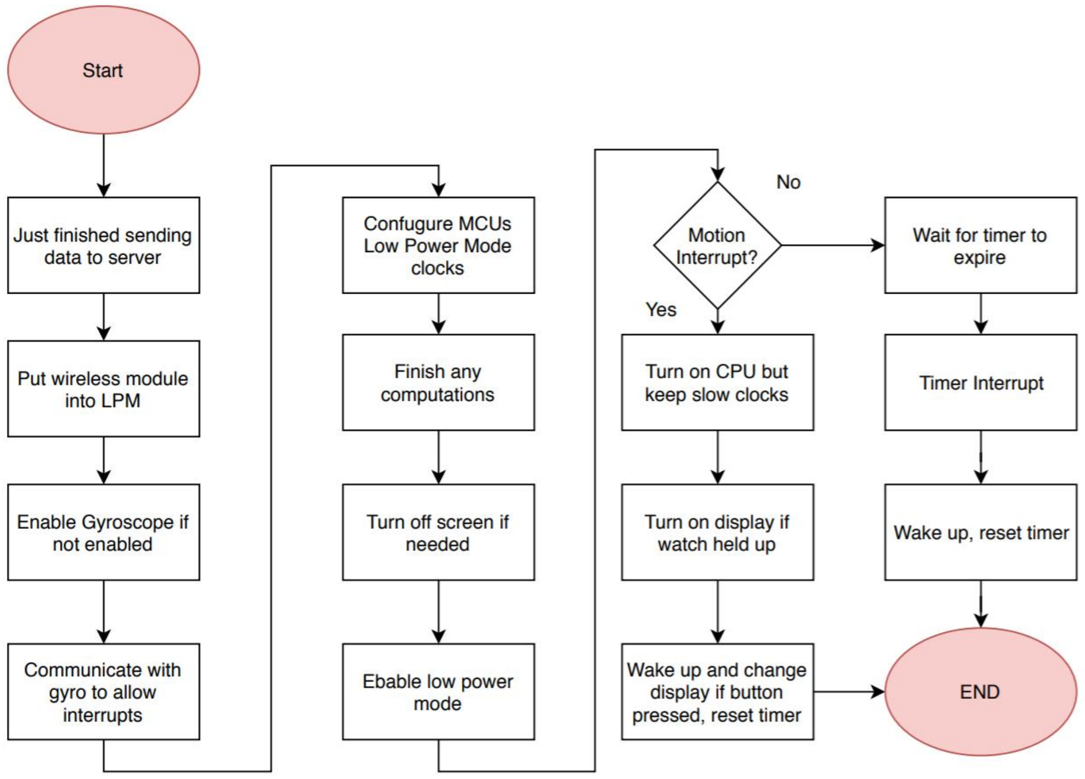

Start
Just finished sending data to server
Put wireless module into LPM
Enable Gyroscope if not enabled
Communicate with gyro to allow interrupts
Confugure MCUs Low Power Mode clocks
Finish any computations
Turn off screen if needed
Ebable low power mode
Motion Interrupt?
No
Yes
Turn on CPU but keep slow clocks
Turn on display if watch held up
Wake up and change display if button pressed, reset timer
Wait for timer to expire
Timer Interrupt
Wake up, reset timer
END

Figure 56. Software flowchart of power consumption strategy

5.16 Summary of Design

All in all, the complete project consists of four major subsystems which deal with the major goals of this project: Medical Sensors, WIFI/BT connectivity, Low power consumption, and MCU technology (enabling it to be portable). The components for these subsystems were chosen such that they could fit in a small compact design, be excellent quality, while being as budget friendly as possible. The main core of the design revolves around the internal main MCU (EXP-MSP430FR4133): similar to a typical computer. This MCU will be the connecting point between the different information peripherals of the system; it will be the main interface for all the inputs and outputs associated with the system; and will drive the internal clock to synchronize the device itself.

The WIFI and potential Bluetooth connectivity will be handled by the ESP32 chip. This chip will be responsible for the communication via to other hospital devices and servers: it is the communication gateway of this device to the outside world. This chip will also be used to expand the applications in which this device be used. Such a chip is key in the design of this product as this chip will allow the device to evolve with new emerging technology and potential newer systems that will possibly be incorporated in a hospital. This is because the current trend in technology is leaning toward the IoT (internet of things) applications which is used to create a more integrated and wireless environment (similar to Home Kit for Apple Technology).

The foundation for all of these chips will be the power distribution systems set in place. This system which is composed mostly of voltage regulators were designed and placed in the vital areas (between the power supply and each corresponding chip) such that maximum power efficiency can be reached. This is a critical part of the design since the energy efficiency will provide the overall device with longer battery life which equivocates to expansion of the potential usage of this technology and of the type and kind of hospital patients it can be used to monitor.

Powering all these hardware chips will of course be the software. The software was designed in such a way that the more energy consuming operations (calculations) where primarily placed on the server side of this project. The software will be working just as hard if not harder than the actual hardware. The software was designed to be able to locate a patient (which uses complex algorithms), place the system in low power modes (which is essential to battery life and efficiency), and monitor the information from the different peripherals and chips.

Overall, this design was very well thought through and geared specifically for applications within a hospital. This is not to say that this design would not work anywhere else. Other applications could be a day care, an elderly nursing home, or even applications that do not deal with the health of someone but simply indoor localization. An interesting thing to pursue would be outdoor localization through LTE and GPS.

6 Project Prototype Construction and Coding

These mark the final steps in the process of the project. Not much can be written but the document will be expanded much in senior design 2.

6.1 Integrated Schematics

An integrated schematic is the design of a circuit to be assembled with electronic components and then fabricated as a single unit. It consists of many components such as microchips, resistors, capacitors, etched in paths, and much more coming together to form an overall device. Our section of the integrated schematics will first be broken down into each subsystem providing a schematic for each portion of the device piece by piece. Then once each individual subsystem is made we will bring all of the schematics together to create the overall schematic in the end, which we will eventually build in senior design two and go into testing each schematic design as well. The integrated schematics is the blue prints to our design.

6.1.1 NFC Schematic

The NFC schematic is quite simple and easy to follow using this diagram. It shows the antenna design of the inductor and capacitor together, where the SPI/I2C may connect to the chip and also where the power input is. There are also some coupling capacitors to reduce noise.

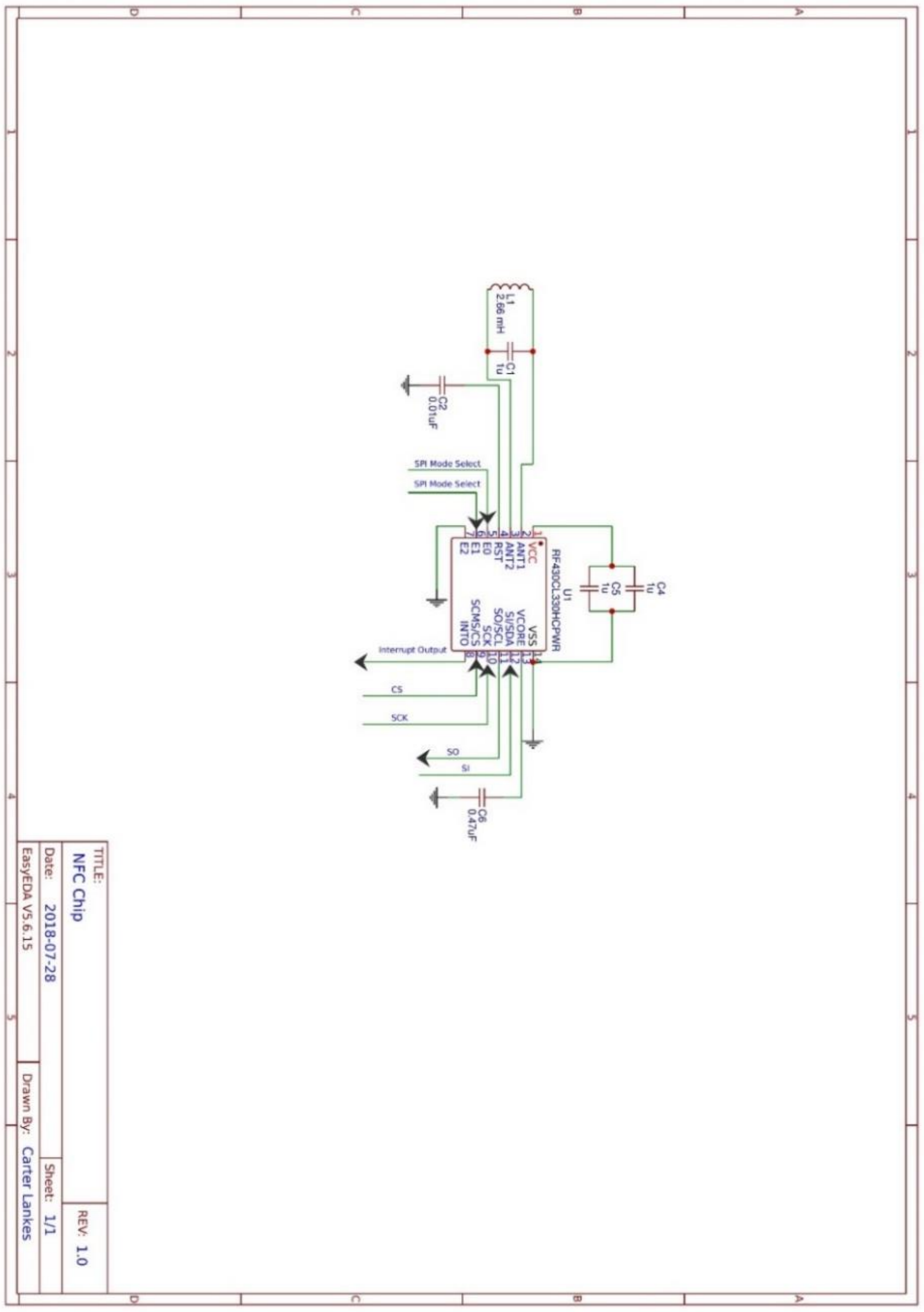


Figure 57. NFC Schematic

6.1.2 Voltage Regulator Schematic

This next schematic shows the layout of voltage regulator LDO REG104FA-3.3 we used that will output a voltage of 3.3 volts. We followed the data sheet exactly when designing this circuit.

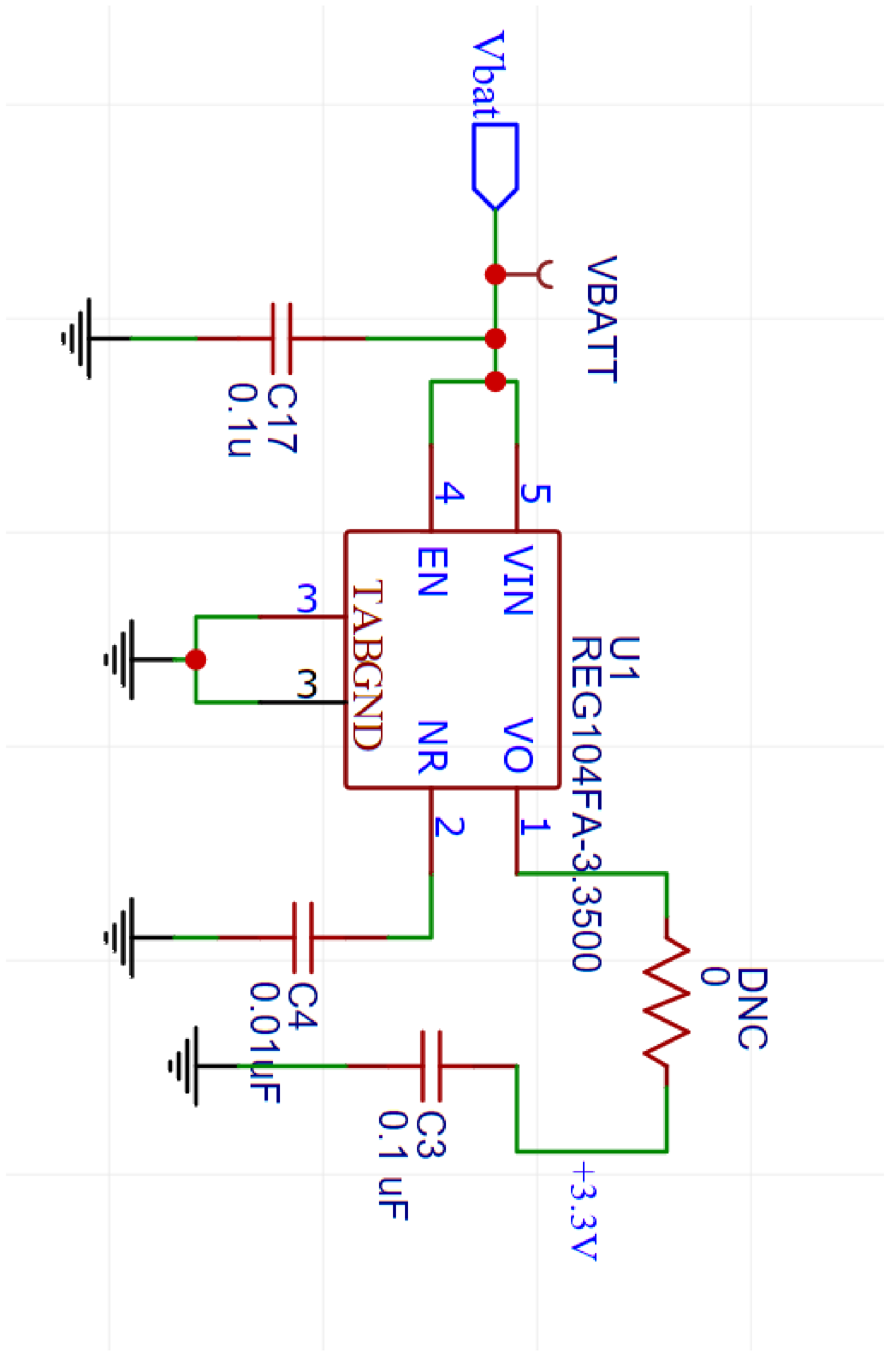


Figure 58. Regulator Schematic

6.1.3 WIFI Schematic

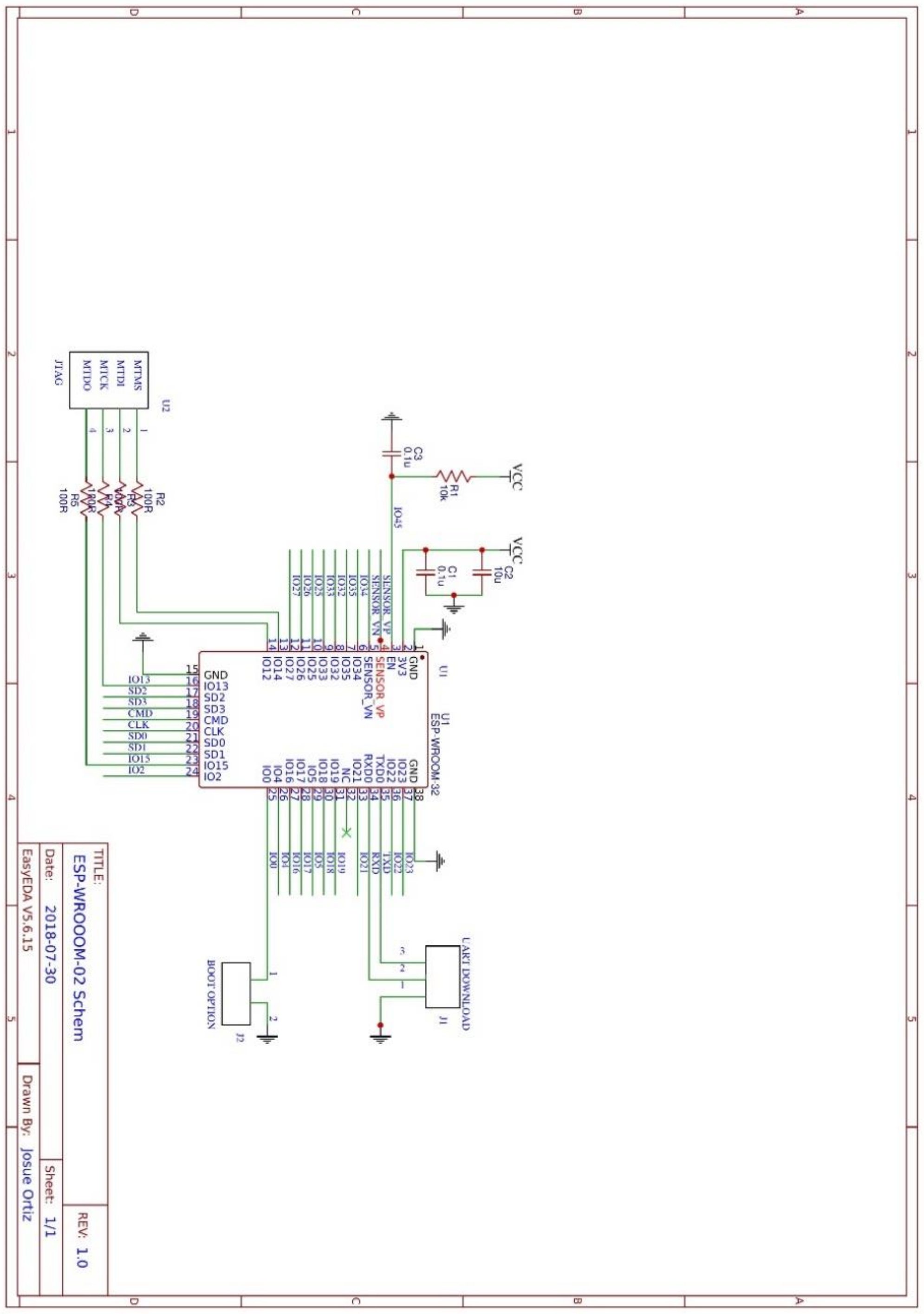


Figure 59. WIFI Schematic

6.1.4 Analog Front End Schematic

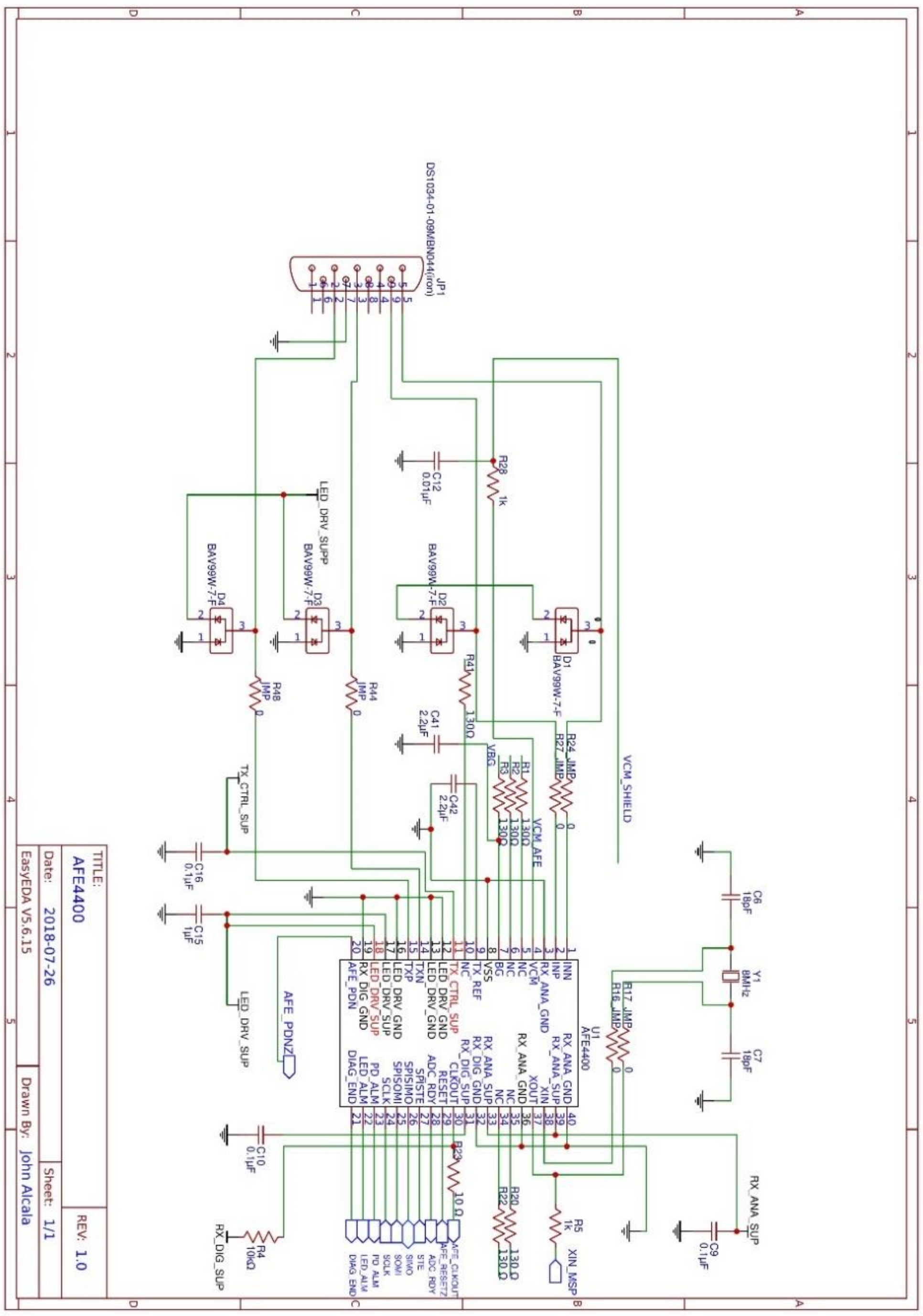


Figure 60. AFE schematic

6.1.5 Microcontroller Schematic

Below is the schematic for the microcontroller. The modules will be connected to the MCU via the interrupt pins and the communication ports, I2C and SPI.

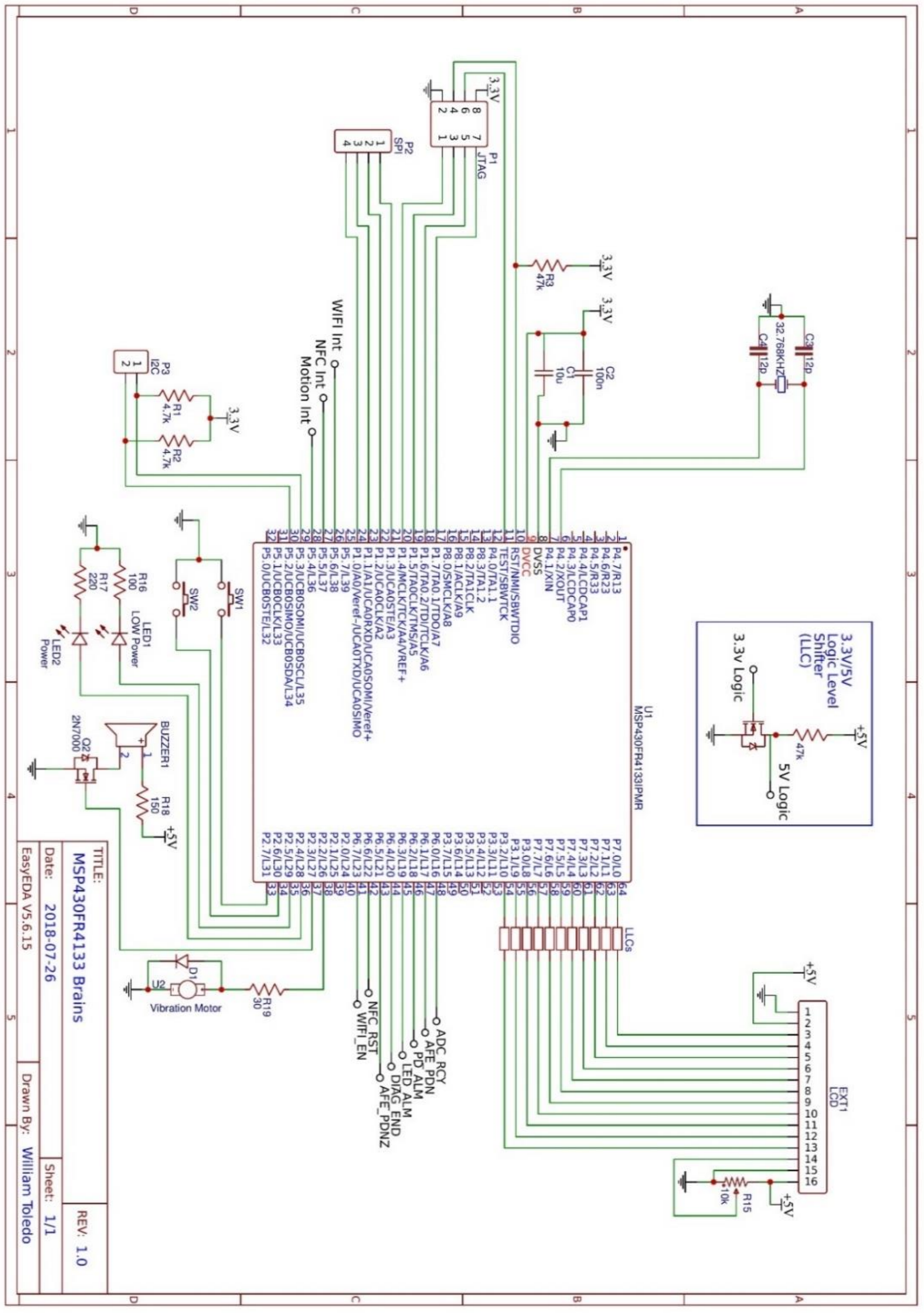


Figure 61. MCU schematic

6.2 PCB Vendor and Assembly

The PCB will mainly be designed and bought in senior design two, but we are going to get a head start on the process and watch tutorial videos on how to create them in Eagle and easyeda. Easyeda offers great schematic and PCB creation, which is great for us beginners. Easyeda also offers great prices on purchasing there PCB boards. It is best to get a head start on this process, so we do not waste too much time in senior design two and can focus on building the prototype. As for assembly there is a local PCB assembly company that can do the difficult placement of the components that we will not be able to do such as baking the parts onto the board. Many of our components are going to be required to be baked onto the board due to their connection being on the underside of them and very fragile to heat based on their size. In the end we should be able to deisgn and assemble a PCB board with not too many problems using these references we have just listed.

6.3 Final Coding Plan

The coding for this project is quite complex and mutli-facet. The whole coding plan for this project can be divided by the separate goals or subsystems that will have anything to do with information gathering, monitoring, transmitting or storing. Overall, the coding plan can be divided into the following general goals or facets: localization, sensor monitoring, and common MCU processes.

6.3.1 Localization

Once the code is completely finished being developed we will go to the next stage of the process and test our code on our device to make sure it is fully functional and there are no glitches or crashes. We will begin by first developing the software needed to scan, record, and transmit the WIFI signals using the watch. This will then be tested to make sure it is functioning properly before moving on to the next step. After the validity of this code is proven, the algorithms for fingerprinting and trilateration will be developed now that the information can be retrieved. These two algorithms will run on a server or separate computer.

Included in developing these two algorithms is also the developing of the code that enables the wearable device to communicate with server and the server to receive the information for processing. Once the algorithms are done, an official test will be made using the device. These are the major coding steps that will be taken in this project. It is important to note that at each minor step along the way, testing will be made to ensure that the minor pieces of code (which will eventually compose the major pieces of code) are all working such that the debugging process at then end can be shorter. The following outline lists the final coding plan steps in chronological order:

1) Developing the Localization program code
    a) WIFI Scan
        i) Scan WIFI signals

        - ii) Retrieve APs MAC addresses
    - b) Write to memory
    - c) Transmit Data
    - d) Incorporate Low Power Mode Coding
        - i) Test Current Consumption
        - ii) Adjust until optimum is found

2) Developing the server program code
    - a) Information Reception
    - b) Trilateration Code
        - i) Test for accuracy
        - ii) Adjust as needed
3) Implement Whole system
    - a) Compile and test the whole code together
    - b) Verify Result and adjust

6.3.2 Sensor Monitoring

The sensor monitoring code will first begin with a code that is able to detect when the sensor pulse oximeter probe is on a patient and when it is not. This is key for running our system in low power mode. After our code is successfully able to detect the signals of a patient, a code will be developed which will take the measurement information coming from the AFE at a specified interval and record this information; because the MCU has limited battery, it will then need to be transmitted over WIFI to a server where the logs can be kept. As with the localization code, at each minor step, testing will occur to help more easily compile and implement the final code. The following outline details the steps that will be taken to code the sensor monitoring code:

1) Patient detection
    - a) Detection function
        - i) Differentiate between false and valid readings
        - ii) Set a signal threshold
    - b) Test and verify that the code works
        - i) Test for different hand sizes
        - ii) Different temperatures
        - iii) Adjust code as needed until acceptable
2) Interval Measurement
    - a) Measurement Recording
        - i) Select an arbitrary starting interval time
        - ii) Test for good mean representation
    - b) Compare measurement recording with power consumption
        - i) Increase or decrease interval time as needed

        ii) Find a good compromise between both

3) Transmit over WIFI
    a) Memory management code
        i) Develop a routine that monitors MCU memory
        ii) Test to see if routine works
    b) Transmission code
        i) Use similar code from localization in order to transmit data to server
        ii) Find the ideal transmitting interval in order to save power

6.3.3 Programming the MCU on PCB

The included JTAG port on the PCB will be used in order to program the MSP430FR4133 on the finished device. The MSP-EXP431FR4133 Development board already includes a USB-to-JTAG programming interface specifically designed for MSP430 devices, called the eZ-FET. Fortunately the dev board has jumpers that can be tapped in order to program the mounted MCU on the medical device, via the MCU_JTAG port.

6.3.4 Common MCU Processes

Left in this category are all the other basic but essentials processes that must be done by the MCU. Such processes are the panic mode code, interfacing with the NFC chip, and interfacing with the LCD screen These processes will be coded and developed based on order of importanct. The first process to be coded for is the panic mode code. The most power saving way of implementing this code is by using simple interrupts. Of course, other fail-safe algorithms will need to be developed so that the triggering of false alarms is kept to a minimum. This code will be coded according to the following plan:

1) Panic mode code
    a) Interface panic buttons with MCU and piezo speaker and vibrator
        i) Use interrupts for power saving considerations
        ii) Verify connection is made
    b) Create a false alarm rejection code
        i) Find ways to minimize false alarms
        ii) Verify that no spontaneous alarms are produced by a glith in the code or other internal means

After the panic mode coding is in place, the next vital process would be the interface with the NFC chip. As stated before, the NFC chip will not hold the patient's actual information but simply an encrypted signal that will be used by the device to "download" the patient's information from hospitals servers. The first step of the doing plan would be to detect when an actual signal is being induced onto the NFC antenna, and be able to reject other kinds of false signals that may arise due to other sources. After this code is tested to be successful, then the following piece of code will need to be able to communicate to the WIFI chip in order to download the relevant information via WIFI. Once the information is received, another function will need to be used to then send it to the NFC chip which will transmit it to the reader.

1) NFC communication
    a) Signal detection function
    b) Noise signal rejection
        i) Find an ideal threshold for this
        ii) Test and verify
    c) Communicate with WIFI chip
        i) Use similar code used in localization to communicate with WIFI chip
        ii) Test to see if download speed is satisfactory
        iii) Adjust if needed
    d) Send information back to NFC chip
        i) Test for information accuracy
        ii) Test for reader efficiency.

Finally, the code for interfacing with the LCD screen can be written. As in most systems, a display, if left on for too long, can consume relatively large amounts of current and therefore power. There will be two basic functions for the LCD: 1) to display the information that we desire on the LCD screen and 2) to turn off when not being used. The second one is the more trickier of the two so it will be the one we focus on since the first goal is pretty standard. In order to code for the second function, the accelerometer will need to be used. A function will need to be coded so that if the sensor detects that the wrist has moved in a certain direction than the screen will turn on similar to the iPhone function that is "rise to wake." Below is a short plan for this coding

1. LCD display interfacing
    a. Create and tune a function to relate accelerometer data to wrist movement
        i. Tune so that it can detect when the LCD is raised to be read
        ii. Test and adjust accordingly
    b. Create functions to turn off LCD display when not being read
    c. Create function to refresh LCD display with new information

Of course, in addition to all these functions and processes to be coded, there are also other standard or trivial code that will also be included in the final code which were not fully explained or even mentioned here because they are standard or trivial. Mentioned here are the main goals and also the toughest assignments that will need to be coded.

7 Project Prototype Testing Plan

The prototype watch testing plan has three stages: Modular, Interconnect, and Final. The first stage is the testing of the components individually in a modular fashion. This stage will focus on the individual performances of each major subsystem, building them individually on breadboards. There will be thorough testing on the power consumption of each system and how efficiently the subsystem performs. This stage will be critical for deciding on whether new components will be needed or if the schematic will need to be updated, affecting the cost of the device.

The Interconnect stage of the prototype testing will focus on how each of the modules behaves together. Because this stage will involve connecting together the different modules, a focus on code will result in this section, especially concerning the communication standards like NFC and WIFI with regards to the MCU. This will begin to finalize the component selection and determine the size of the first PCB.

The Final prototype stage will involve field testing a PCB design for battery consumption, location accuracy, response time, and sensor accuracy. The initial tests will determine how efficient our layout was and whether a major revision in design will be necessary. Extensive field testing will also be necessary in order to verify durability and possibly water and dust resistance.

7.1 Hardware Test Environment

For the Modular stage of the testing, most of the testing will be taking place in the Senior Design lab. These allow us to test our modular devices using the power supplies and use the multimeters and oscilloscope to test different things. However, some tests will take many hours to execute, and might require testing at home. One specific testing requirement will be the ability to modify the settings of a wireless router. Because of the controlled nature of the wireless network on the main UCF campus, any customized wireless network testing will require a different location with accessible settings.

The Interconnect stage of the prototype testing will majority take place on the Texas Instruments lab in the Engineering Atrium on UCF. The benefits of the testing here is that there are many soldering stations inside the lab and with facilitate any hardware modifications to the protoboard, including passive component changes and even some surface mounted help. The focus here will be on the communication between each module and their combined efficiency. There will be some software focus in this stage of testing, but the software testing will be covered in sections 7.3 and 7.4.

Most importantly about the testing environment in this stage particularly s the power consumption of the design as a whole. In this stage, the initial PCB layout can be roughly shaped using homemade PCBs, using etching compounds. This requires a good amount of highly corrosive chemicals which could result in serious injuries. If possible, these etchings will take place inside the chemistry department. If not possible, the alternative would be to do it at home, of course taking the proper safety measures, including skin and eye protect.

The Final prototype testing will begin at the Senior Design labs once again. This will allow us to use the lab equipment's to verify that the design is functional when moving from protoboards to PCB. The Texas Instruments might also be another testing environment that will allow reflowing of components if necessary, especially considering the miniature nature of a watch-sized schematic.

The next environment will be the UCF campus, which will allow us to test the localization capabilities of the watch when connected to a wireless network. At this point, the major concerns will fall on precision of the localization while also being able to take proper measurements. The UCF campus is also a place with a substantial student population, which would allow anonymous testing of the heart rate sensors. We would take extra care to follow HIPPA standards of collecting anonymous data.

The final testing place of the prototype would ideally take place within a hospital setting. More research is necessary before determining which hospital would allow this. However, this testing environment would be ideal as this would be the intended environment of the product when finalized. The tests here will focus on the usability of the device in real-life settings while also getting opinions from the people whom would be the intended audience.

7.2 Hardware Specific Testing

The prototype will have at least five different subsystems that must be specifically tested individually and eventually as a unit product. Some of these subsystems are the wireless transmission of data and location via WIFI, the wireless reading and writing of patient information via NFC, the clarity and power use of the LCD display, the accuracy of the heartrate and oxygen level readings, and the power delivery system of the battery. The specific plan for each of these systems will be discussed in each section bellow.

7.2.1 Testing Wi-Fi Location

Accurate localization using Wi-Fi is critical to our design, properly testing the functionality of this feature will be very important. Ideally that testing will be done in a building with multiple rooms at several meters apart. The Harris Engineering or Engineering I building at UCF will be more than sufficient for these purposes. For our system and device, it is also important to test the fingerprinting and the trilateration positioning algorithm separately so that they can be more effectively combined in the final system. In this way, a good median can be used between the two in order to more closely approximate a patients location than if only one algorithm were being used.

The Wi-Fi Positioning System (WPS) using trilateration is only useful for 2D position and not altitude according to previous studies and documented trials have indicated. Therefore, in order to test the trilateration algorithm, the testing will be done on a singular floor at a time. The PCB board or breadboard with battery attached will be moved throughout the hardware test environment and communicate to a nearby computer that has permission to communicate data with the device. This data as well as the physical positions of the APs relative to the building will have to be imported to an indoor map where it can then be used to place a physical location of a person. For our testing purposes we will have to create an indoor map of a building at UCF to display the location to a nearby computer or android device on the LAN. This can be done using an open source software called indoor location (www.indoorlocation.io). It is important to note on this testing that direct communication from the device to the computer is not required since we are not testing this. The only two things being tested here are the devices capability of receiving and scanning the WIFI signals and the team's ability to properly create an indoor map.

The testing of a WIFI positioning system using fingerprinting is a bit more rigorous. To set up the finger printing testing, the first step is to pre-record RSSI values at different locations inside a building and build what researches have names as a "radio" map. This radio map is nothing more than a collection of APs' RSSI data points for locations inside a building: in a sense, this method, "finger prints" or acquires the characteristic of the building at these locations. Since this method uses pre-recorded values, enough values can be included in the data set for each point such that there can be a differentiation between different floors. To test this software, the same procedure will be used as with the trilateration except the testing for fingerprinting will also include a change of different levels. To keep things simple at first, finger printing will only occur on the physical levels since this mostly simulates how a hospital will be fingerprinted. An interesting stretch goal would be to be able to map even positions that are in between floors (such as stairs wells) in order to have a more complete map.

After testing both algorithms separately and figuring out the benefits and the downsides of each, the next test will involve combining both algorithms and testing the new combination of both such that the localization can be more accurate. The testing of this will also be very similar to that of the fingerprinting and of the trilateration. Several tests will have to be run to tune and effectively combine both of these two algorithms into one.

7.2.2 Testing Wi-Fi Transmission of Data

Sending frames of data from the sensors heart-rate monitor will be vital to achieve the goal of a wireless continuous monitoring system. Ideally the data will be sent to an android device on a LAN. Testing this will be very simple, gathered data from the sensors will be sent via an algorithm. The small difficulty in this testing is having a simple operating system on our board so we can easily synchronize the data. A final choice as to which open source operating system will work best with our MSP430 Family MCU will be made during testing, as it could be subject to change and is not a critical design factor in our project and can be easily flashed on the memory.

7.2.3 WIFI Coexistence

Besides testing the accuracy of location and the data transmission over WIFI, we will also use the standards mentioned in section 4. One of these standards is the test for WIFI coexistence. Even though our WIFI chip module is already pre-tuned and tested by the manufacturer, according to the ANSI C63.27 standard, this cannot be relied upon since this can usually lead to erroneous conclusions about our design. To perform this evaluation or test, our device will be placed inside the testing environment (HEC building) and near an access point that is functional and transmitting various WIFI signals (different SSIDs). Once the device is placed as close as possible to the access point, the device will be turned on and commanded to transmit data over WIFI and perform localization. In this way, the coexistence of this device with other WIFI signals will be tested.

7.2.4 RF Emission Interference On-Site Testing

Besides testing WIFI coexistence of the device, this device will also need to be evaluated to determine whether it can operate correctly even while receiving powerful emissions from other RF working devices. This kind of testing is usually done in a lab where other parameters can be easily controlled and neutralized, and where other kinds of RF equipment can be used. Since we do not have a lab available to us where we can test this, the ANSI C63.18 standard provides a way to evaluate this in a crude but effective manner. For this testing, due to the equipment limitations, various cellphones will be used to emit RF unto the device. The standard details how exactly to set this up and at what distance the cellphones will need to be placed in order for this kind of testing to be effective. Once we make the cellphones transmit the RF signals, the various operations and peripherals of the device will be tested to see if they still operate correctly. For example, the LCD will be checked to make sure it is not display weird characters and the WIFI data transmission and localization will be tested to ensure that these critical operations still maintain accuracy even when being bombarded with RF emissions from other devices. As stated in the actual standard, this kind of testing should not replace lab testing, but it does provide a way to crudely test this kind of operations.

7.2.5 Testing NFC Reading and Writing

Testing the ability of the reading and writing of data on the NFC chip will be essential for storing the patient's information. In order to be able to test this chip we must use a breakout board since the chip leads are too small to test on a breadboard alone. We will verify and validate that our component does in fact meet our desired needs and operates to the standards. In order to test this component of our device we will test for its range of being picked up by an NFC receiver, which would be our mobile device (Refer to Range testing section for more information). We will also test for the amount of data we can store onto the chip that will be transferred to our microcontroller, and possibly the NFC chips Bluetooth capabilities as well. There is a possible app we could use to monitor our sensors and other components from Texas Instruments that we are looking into using and it could be paired using Bluetooth. If that is not the case, then we will have to develop our own simple app to enter necessary patient information. Another aspect to test on the NFC chip will be encrypting the data so it cannot be stolen. We will do this by sending the data encrypted and then downloading it to the device. Also we must make sure the device sending out the data when it needs to be read is also encrypted. We must test to verify that the data is in fact encrypted when being both sent and received because patient's information is very sensitive. Overall for the NFC chip we will be testing to write encrypted data onto the chip and then using it again to scan and receive the encrypted information to verify it operated correctly. Repeating these steps over and over to make sure it works consistently as well.

7.2.6 Testing NFC Range

As for testing the range of the NFC chip this really comes down to the antenna design which involves our inductor and one or more capacitors to adjust the resonance frequency to 13.56 MHz so it can operate at its full potential. The suggested inductor value is about 2.66 microhenry's and the NFC has an internal capacitance of 35 picofarands. For our inductor we will be using a 0.32mm wire and coil it 6 times with a loop diameter to get around our desired value of 2.66 microhenry's, but it will probably not be exact so we will adjust the capacitance to get us to our frequency. To test the actually frequency we will use a spectrum analyzer if we can obtain one, other wise we will use an oscilloscope to see where the frequency peaks and then calculate our actual inductance and adjust our capacity from there. The close to 13.56 MHz we get the more range we have for scanning, which should be to a max of about 10 centimeters.

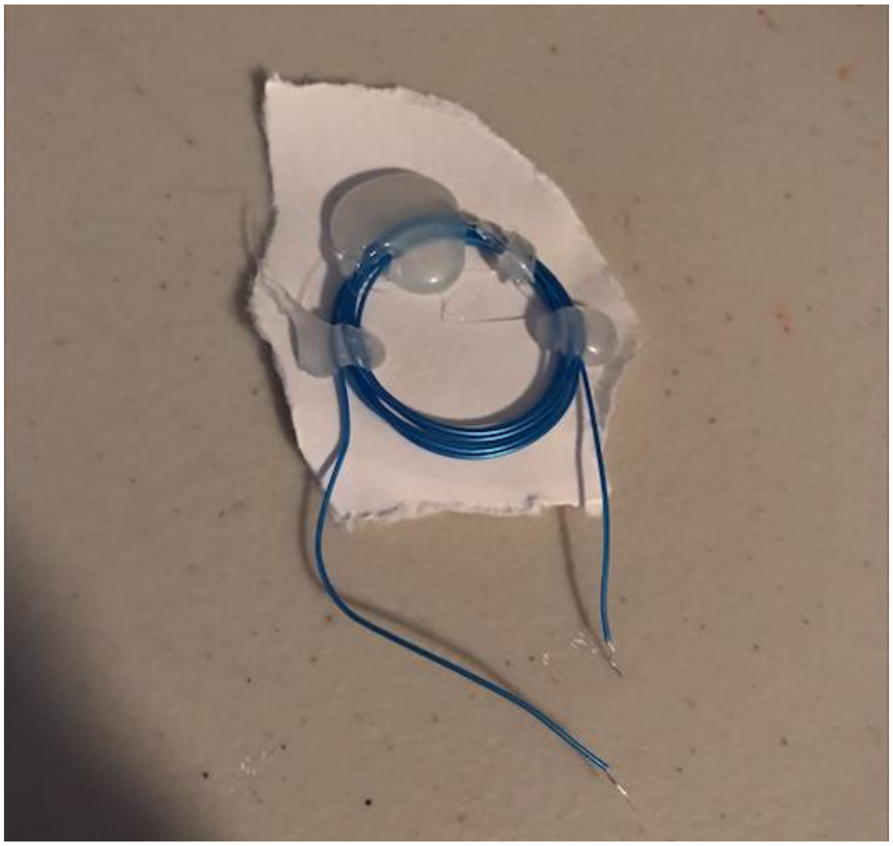


Figure 62. Custom made inductor for Antenna design

7.2.7 Testing Pulse Oximeter

The pulse oximeter is another essential function of our design which measures a subject's heart rate. We will require another good means of measurement to confirm our results are correct. For initial calibrating of the sensor we can use an android based smartphone's pulse oximetry functions and for final calibration we will investigate a way to obtain an ECG based result, perhaps by the UCF Health Clinic. The physical analog probe for obtaining measurements is a DB9 (Serial) based pulse oximeter placed at the subject's fingertip. The output waveforms of the SP02 measurement must be observed and calibrated. If the signal is weak it must be tweaked to the provide the proper output. This could be due to needing higher resistance on the probe or simply by providing more voltage to the LED drivers. The Texas Instruments reference design for the AFE is made in mind with the DB9 sensor connected so that resistance values should be properly set. The adjustment to the LED voltage will most likely be the parameter that needs to be calibrated. Since the soldering of the AFE to the MSP430 is complex, this particular test will not be able to be performed until Senior Design II.

7.2.8 Testing LCD

The LCD screen in our project does not require any complex graphics just a simple frame that would display simple information (e.g. time, heart rate, battery life). Testing will involve connecting the LCD to the MSP430 MCU on a breadboard with their respective pins. Sending a sample frame will involve using Code Composer Studio to send this frame. Our LCD uses serial communication with the MCU to send and receive data, use of the ready-made SPI library on the MSP430 will be implemented.

We want to send a small frame for testing purposes, the frame size on our LCD is only 8 bits so we can simply send a few words on to the frame.

An unfortunate oversight when choosing the LCD was the operating voltage. Many CMOS devices have a standard of 5V, and this standard is used very often with certain electronics hobbyists. The Arduino Uno (ATmega328) operates at 5V, and so does the LCD that we chose. However, the final microcontroller that we used operates on the TTL standard, which is 3.3V. This meant that the MSP430s could not directly interface with the LCD.

A logic level shifter allows for the TTL voltage to be boosted to CMOS standards by the use of a MOSFET and resistor. The benefit here is that the logic can work with both reading and writing, but unfortunately it also inverses the logic. This is addressed by the code of the MCU. This is could be temporary measure if a Logic Level Shifter IC is not found. This increases the footprint of the LCD, requiring eleven resistors and MOSFETs. The schematic bellow demonstrates the design of the LLS.

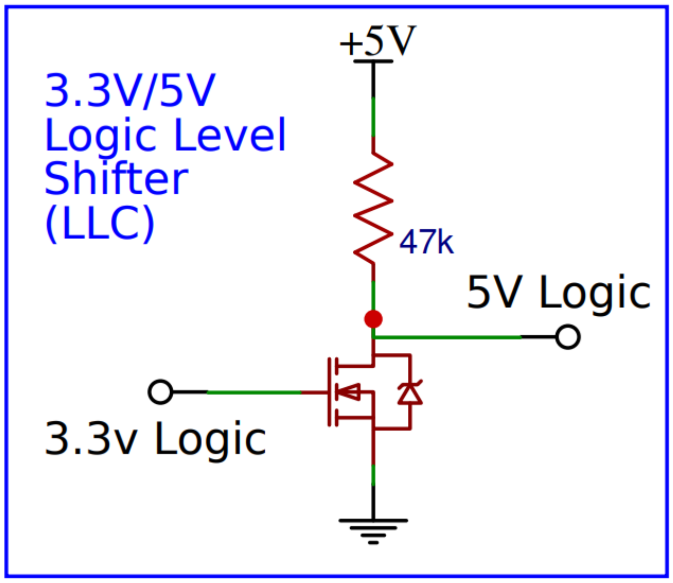


Figure 63. Logic Level Shifter, TTL to CMOS

7.2.9 Testing Battery Life

For testing the power supply we want to verify that it performs how it says it does by actually supplying 3000 mAh and measuring how long it can last under certain test loads. We can measure the battery's capability by using a multimeter or oscilloscope to measure its voltage and current and seeing how long it can sustain before dying and needing to be recharged. We will also need to calculate the total power consumption of our device and test how long the battery lasts under that specific load to get a good general idea of how long it can power the device for. We can also look into testing the recharge capability of the battery to see what the most efficient and quickest way for recharging it is. Another option to test as well is to measure the battery's performance for a low power battery saving system and then comparing it to normal power of the device.

An important point of discharging the batteries is that the battery voltage will drop as the battery becomes less charged. To get the right values, we need to have a constant current drawn from the device. Using an operational amplifier and an N channel power MOSFET allows for such a test. Here we would need to measure the voltage of the battery and construct a voltage curve. The way this circuit operates is that as the battery depletes so will its current and voltage. In order to maintain a constant current draw we set a voltage reference right before the resistor that draws the current. We then use the op amp as a comparator to notify when to boost or cut off the current through the MOSFET to keep the voltage reference constant.

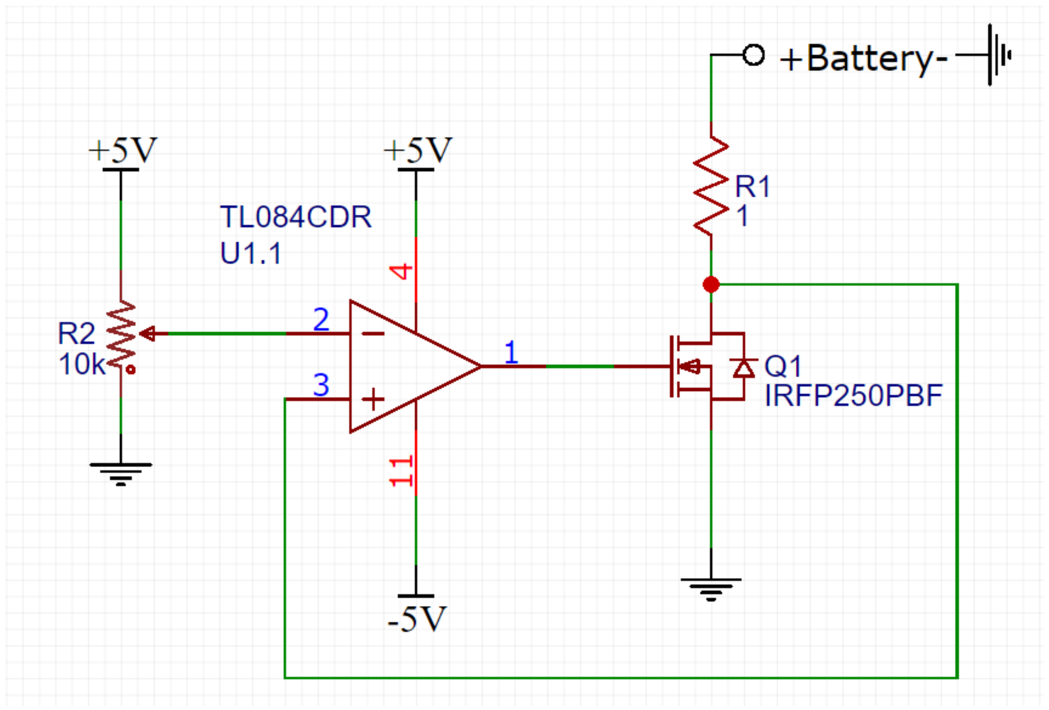


Figure 64. Constant Current Source

A critical part of the testing is getting the op amp to stabilize at the value of the voltage divider. The design is meant to draw 600mA of current from the battery. The voltage divider is set to 600mA

The voltage divider is set to 600 mV, and the op amp would allow for a voltage difference of 600mV across the 1 Ohm resistor, drawing 600mA. The op amp works by switching the MOSFET on and off to average 600mV at the terminal. The initial testing used an LM393 but the switching speed was too slow and would result in the lowest voltage average of 2.6 V. The diagram bellow displays the output voltage oscillation at the bottom of the resistor.

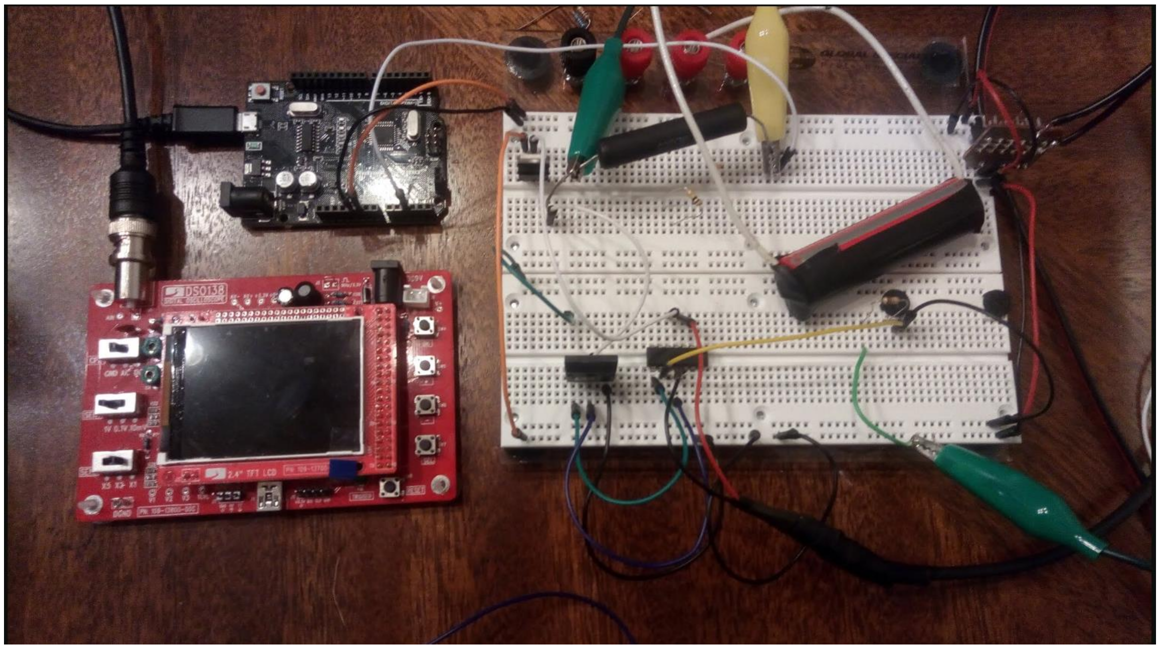


Figure 65. Testing Constant Current Circuit

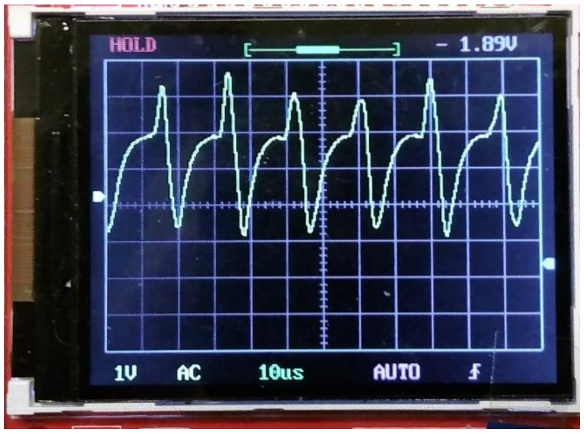


Figure 66. Op Amp Output Driving MOSFET

The voltage across the battery was measured automatically using an Arduino Uno. The test lasted for ninety minutes until the battery voltage dropped below 2.8V for seconds consecutively. Unfortunately the ADC of the Arduino made the measurements of the battery very imprecise and resulted in a very coarse load line. The test will be run again later, but the 90 minutes at 600mA, results in a capacity of at least 900mAh. However the voltage dropped from 3.77 to 3.49V, when measured with a multimeter. Assuming a near linear drop in voltage until 2.8V, the capacity can be approximated to be about 2600 mAh, which is near the expected capacity.

7.2.10 Testing Voltage Regulators

The voltage regulators do not need to be tested, but instead verified. Most of our sensors, chips, microcontroller, and more, operate at either 3.3 volts or 5 volts. So we will adjust our input supply of 3.7V to those desired outputs with our regulators just to make sure they operate correctly.

7.3 Software Test Environment

Most of the software testing will take advantage of the code composer studio's very rich development environment. Because it allows controlling the MCU one step at a time, the programming from the MCU and other peripherals can be debugged easily.

Another software environment that will be taken advantage of are the online data servers like Google Drive and One Drive. Using Googles development environment, some levels of data processing using Google sheets can be automized. This would in theory allow for data acquisition if the API is compatible.

Failing at using these systems, some help can be requested from computer and software engineers at the university of Central Florida. This will be very helpful since all four team members are Electrical Engineering majors.

7.4 Software Specific Testing

Each of the software systems has to be tested individually and in combination to provide thorough data and ideal software. The three most important points of the project are the ability to test the localization of the patient within a building, testing if the server can communicate the information of the patient between the readers and the watches, and making sure to test the power saving modes.

7.4.1 Testing Mapping

The best way to test the mapping function would be to try creating our own map in one of our homes. The use of multiple routers at a home would allow for customized router settings which could prove critical when trying to connect to multiple router in quick succession. This would be for small scale testing primarily because of the small area that most homes are, at least in comparison to large hospital buildings.

The secondary step to testing our mapping would be to move to a larger building that could provide a more realistic goal. The ideal testing location would be within a hospital since that would be end goal. However, there could be issues with having tests done at a hospital while the patients are also there. The best middle ground solution would be to test on the UCF campus, particularly the Harris Engineering Center.

7.4.2 Testing Server Transfer of Patient Data

Making a server is not very difficult when using a home network and will allow us to decide how everything will be transferred. However this too might need some external help from a software engineer. The key point that we will be focusing on will be the download of data to the watch, and the upload of the raw location data.

The next thing that we will have to test is what data will be beneficial to have on the server, available for the watch to download. It will definitely include the patients' names, IDs, allergens, and possibly the upcoming procedures. The watch must also be able to download the time from the internet in order to synchronize all communications.

7.4.3 Testing Power Saving Options

Because of the evolution of the technology in microcontrollers and processors, almost all major chips that can possibly be used in a design have low power mode options: many of them have similar low power modes. In this case, both the main MCU and the two MCUs in the WIFI chip have low power modes. The main testing here will cover the different options for low power modes in conjunction with determining how often to use wireless chip: this is because according to the datasheet, the WIFI chip is the biggest user of current therefore power. Testing will need to be done to find the optimal power savings while keeping the system reliably tracking and responsive.

When it comes the low power modes for MCUs, the primary specs that the group will be comparing and considering is clock speed of the MCU. For the WIFI chip, the low power modes will be compared by response time and current usage.

Some other parameters that are relevant to this project but that do not consume as much current as the ones explained above are the following: the precision of analog to digital conversion, the samples per second, the time sampling, and the time between sampling. Also, something important to notes is that the time in which the wireless card is on will play a crucial difference in power and location accuracy. The final major power save will be determining the screen off time of the display.

8 Device Implementation

After coming up with the whole plan, research, and design, we went ahead and actually implemented the design. During the process of implementation, we encountered some obstacles and had to shift the focus of our device somewhat. The following sections will describe in detail the implementation of our work.

8.10 Subsystems

As explained through this paper, there were four main subsystems to our design; power, wireless communication, main MCU, and peripherals. The following sections will speak about the acutal implementation of each individual subsystem: what worked, and what did not work.

8.10.1 Power

The main power supply in LifeWatch is a single TI REG104 Low-Drop Out Regulator which is connected to a rechargeable 3.7 V Lithium-Ion rated at 2.5 Ah. The type of battery used in LifeWatch was carefully chosen to reduce overall physical size of the device and retain the same size of the PCB. The system needed a battery which would be re-chargeable therefore, disposable battery types were not considered. Several factors for a battery were desired. Firstly, a high energy density was required to prevent the need for constant charging. Secondly, the need to prime some types of batteries on their first charge was not ideal. This led to various lithium-ion batteries being explored, a disadvantage to the lithium-ion selection was the requirement of a protection circuit. Initially on the prototype a cylindrical Lithium-Ion battery EBL 18650 was used. Due to the cylindrical shape of the battery, a new one was chosen to fit flush with the PCB and case enclosure. The regulators were changed from two TI TPS63036 operating at 3.3V and 5V to a single TI REG104 operating at 3.3V discussed in detail within the PCB section. We also used a TP4056 battery charger to resupply our lithium ion battery with ease of using a micro USB.

8.10.2 MCU

8.10.3 Wireless Communication

As discussed previously, the subsystem for wireless communication consists of both an NFC/RFID component and a WIFI component. Both systems will be spoken about next.

8.10.3.1 WIFI

In senior design 1, we discussed about 2 main goals we wanted to accomplish with the WIFI system: localization and patient data communication to server. The first was a stretch goal we set out for ourselves while the second was one of the primary goals. In the end, we did not achieve our goal of loacalization although we were able to harvest the raw data to do it. Due to time constraints and the complexity of the process, we decided to leave it out of the project.

8.10.3.1.1 Data Communication to Server

Despite this, we were able to successfully implement data communication from the WIFI chip to the server. We did indeed stick with the WIFI chip we chose in Senior Design 1: the ESP32. In our final implementation, this chip communicated via SPI with the MCU to obtain a patient ID. After connecting successfully to a WIFI network (more on this later), it would send an HTTP GET request which included the ID number to our server. Our server would then service this request (more on this later), and transmit the patient information to our WIFI chip. Our WIFI chip would then transfer this information to the MCU via SPI. After finishing all transactions, the WIFI chip would return to Light Sleep mode where it retains all the pin configurations and memory but pauses the main processor. It would only wake up when interrupted by the MCU when it would then read a command and execute the appropriate action. It is important to note when using SPI mode, specifically with this chip and in full duplex form, that in order to receive n number of bytes, the chip must also send n number of bytes. This must be kept in mind to avoid any sort of confusion with the block diagram for the software that involve SPI. The code developed for this chip is provided in the appendix. The following is a software flowchart that summarizes the functionality of the WIFI chip.

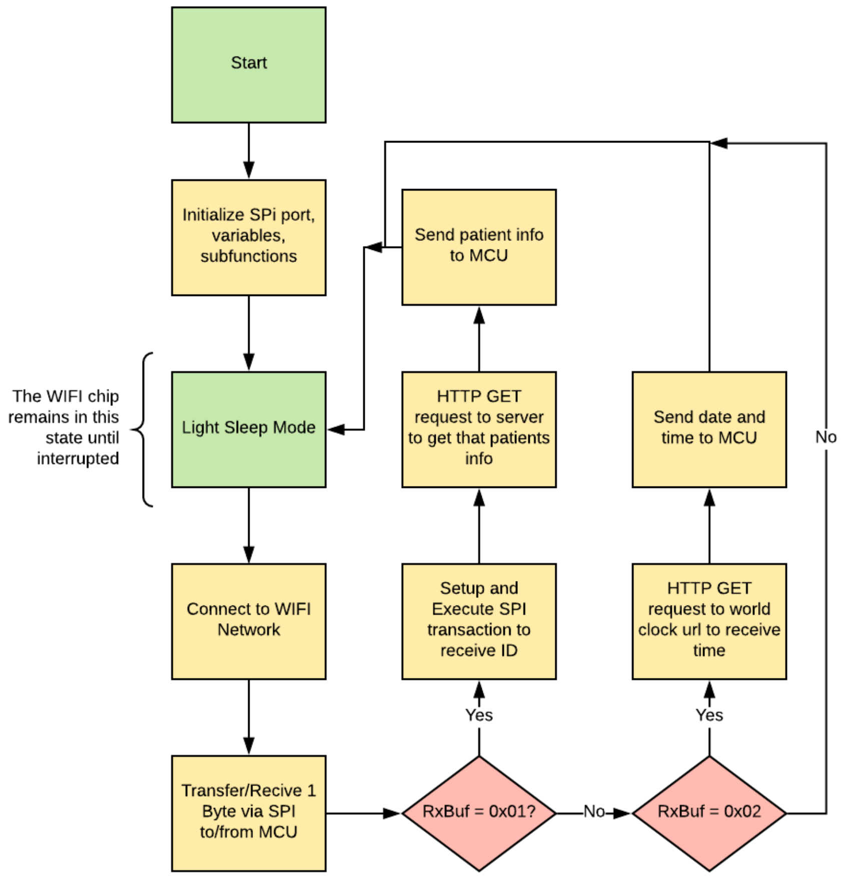

Start
Initialize SPi port, variables, subfunctions
Send patient info to MCU
The WIFI chip remains in this state until interrupted
Light Sleep Mode
HTTP GET request to server to get that patients info
Send date and time to MCU
No
Connect to WIFI Network
Setup and Execute SPI transaction to receive ID
HTTP GET request to world clock url to receive time
Yes
Yes
Transfer/Recive 1 Byte via SPI to/from MCU
RxBuf = 0x01?
No
RxBuf = 0x02

Figure 67. ESP WROOM 32 software flowchart

As can be seen in the flowchart, the main part of the code is the light sleep. This light sleep enabled us to design a device that turned out to be low powered. The following table summarizes the tested current consumption of the chip:

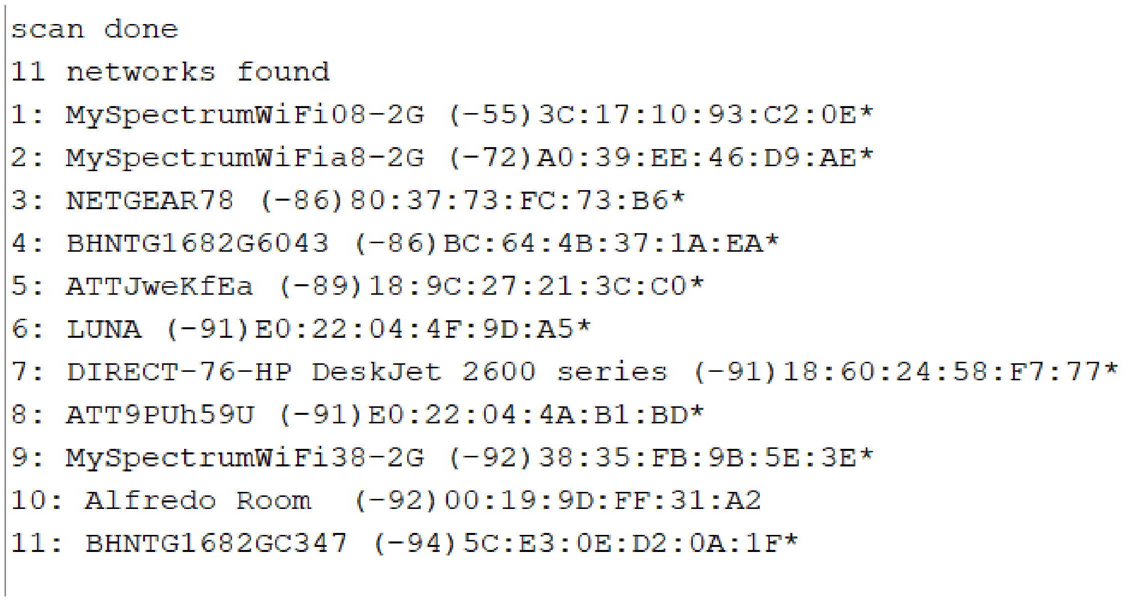
Table 16. WIFI Chip current consumption

Operation	Current Consumption
Light Sleep (MCU paused)	19 mA
Connected to WIFI	100 mA

In our implementation, we attempted to get the time every second but this did not work. So because we were only getting patient information, the average current consumed was a lot closer to 19 mA since ideally, the patient information download via WIFI would only happen when the device is being initialized for a specific patient.

8.10.3.1.2 Localization

As mentioned before, our stretch goal of localization was not met, but we were able to harvest raw data which is the first step in the attempt to localize. Part of the reason we did not achieve this goal was simply due to underestimating the complexity of this localization and our inexperience in writing algorithms. With much more time and experience, this can be developed and successfully included in the features of Life Watch in a future revision. The following is an example of the raw data we were able to harvest using an Arduino ESP32 example called “WiFi Scan.”



```
scan done
11 networks found
1: MySpectrumWiFi08-2G (-55)3C:17:10:93:C2:0E*
2: MySpectrumWiFia8-2G (-72)A0:39:EE:46:D9:AE*
3: NETGEAR78 (-86)80:37:73:FC:73:B6*
4: BHNTG1682G6043 (-86)BC:64:4B:37:1A:EA*
5: ATTJweKfEa (-89)18:9C:27:21:3C:C0*
6: LUNA (-91)E0:22:04:4F:9D:A5*
7: DIRECT-76-HP DeskJet 2600 series (-91)18:60:24:58:F7:77*
8: ATT9PUh59U (-91)E0:22:04:4A:B1:BD*
9: MySpectrumWiFi38-2G (-92)38:35:FB:9B:5E:3E*
10: Alfredo Room  (-92)00:19:9D:FF:31:A2
11: BHNTG1682GC347 (-94)5C:E3:0E:D2:0A:1F*
```

Figure 68. Localization Raw Data

The raw data includes, the network name, signal strength, and MAC address. In theory, the network name is not needed but can be helpful with testing. For more information on how localization is performed, please refer to section 3.2.1.1.

8.10.3.2 NFC

A TI RF430CL330 IC was implemented to provide the Near-Field Communication capability. An external antenna was created to provide a way to communicate with our android smart device. The antenna had to be finely tuned to meet a frequency of 13.56 MHz for good range of pick up. It consisted of an inductor in parallel with a capacitor to meet the datasheet requirements as close as possible. It was crucial to finely tune the antenna in order to pick up the signal through the thick casing and we made many antennas to finally make one that could fit in our case and provide good enough range. This was one of our mistakes that spent more time than necessary, just buy an antenna. The NFC was a major subsystem of our design, tied together several functions. Firstly, the NFC provided patient information from the server when scanned through an android mobile phone running our application that is written to the device and displayed on the LCD. Secondly, the NFC can read current written information on the device and show all the respective patient data held within the server on the mobile phone application.

The read and write ability on the NFC was a key component to providing LifeWatch with the patient data. Below is the LCD screen displaying a patient ID after being written by NFC through the android application.

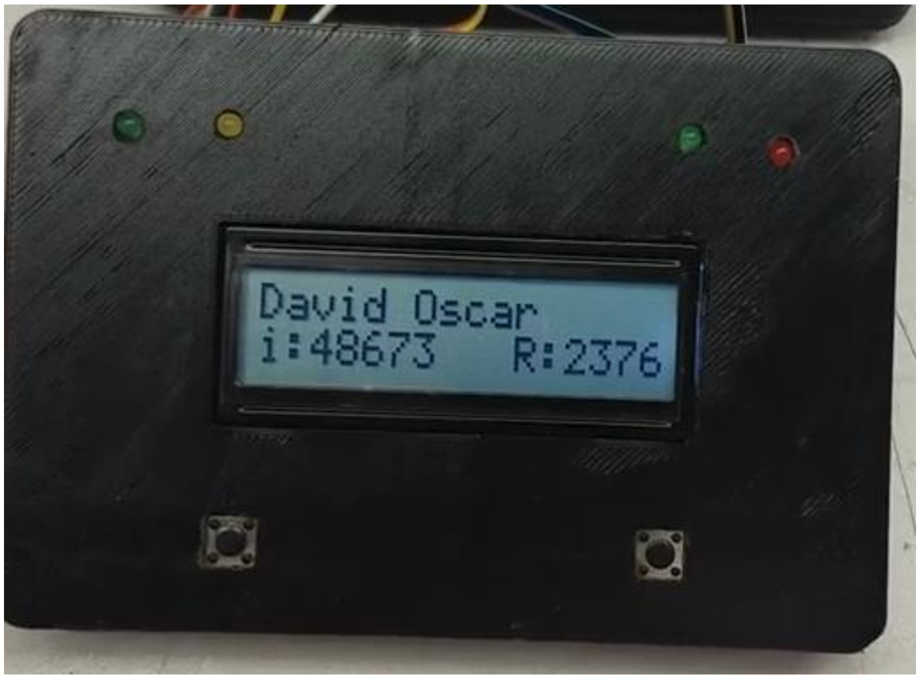


Figure 69. NFC Communication

8.10.4 Peripherals

The final design contained a major peripheral, the LCD Screen. The initial design implemented a 5V LCD Character Display, due to changes in regulator design it was decided that a 3.3V LCD Character Display could be used instead and was chosen as in the final design. The character display contained important patient info that was provided through NFC read and write. The peripherals system includes: a heart rate monitor, LCD screen, and emergency response components.

The LCD screen is responsible for displaying patient information and room number. It also is able to display the date, time, and can be expanded to include allergens, medications, and any other important notes. An important note is that the backlight of the display consumes about 20mA when turned on, which is more that the MCU GPIO pins can supply. A P Channel MOSFET allowed the LCD to be toggled by the MCU in order to save power.

The emergency response system is composed of buttons, vibration motor, and piezo speaker which can provide a method for patients to activate to alert surrounding staff. The chosen buzzer has an IC that provides the PWM wave used to drive the piezo components. This device simplified the emergency response so that the MCU only needs to provide a state to the buzzer as on or off. The important point of the emergency response is that it can be easily accessed when needed, but difficult to do by accident. Pressing both buttons consecutively for a full second, toggles emergency response.

On the other hand, pressing either of the buttons individually will wake the device from low power mode and then update the display. Each button press will scroll the display between the GUIS. An important note is that the button will oscillate when pressed until response settles, and this is due to the mechanical nature of switches. A software debounce is used to disable the button from waking the device for 15ms, which will allow the state to settle.

8.11 PCBs

Overall the PCB remained the same in dimensions from revision 1 to revision 2. There were two major changes in revision 2 of the PCB: The NFC and the regulators. These will be discussed in detail. The initial power regulator used were the TI TPS63036, two of them operated at 3.3V and 5V. During initial testing of the prototype after the first revision had it’s soldering all completed we found the 3.3V regulator to not be turning on despite following the recommended design by TI. Furthermore, the 5V regulators only purpose was for an LCD screen which could be replaced by another screen with the same functionality that runs at 3.3V. Therefore, these regulators were eliminated from the design and a much simpler and reliable regulator was implemented on revision 2; the TI REG104 Low Dropout Regulator. The initial test of this regulator once revision 2 was soldered caused no issues and therefore it was the final power regulator.

The NFC in the first revision contained an area for the antenna, this was removed to create more space for traces and to reduce noise that could be caused by the Wi-Fi or cause noise to the Wi-Fi by the NFC induction itself. The antenna was externally located in the case enclosure with the two ends soldered to the board. The NFC IC itself was also slightly moved to also create more space. Overall, those two major changes were the only ones made besides slightly different trace routes but nothing significant. Below are LifeWatch Rev. 1 and 2 with the changes to the NFC area removal and new larger LDO regulator clearly visible on the lower left hand side of the board.

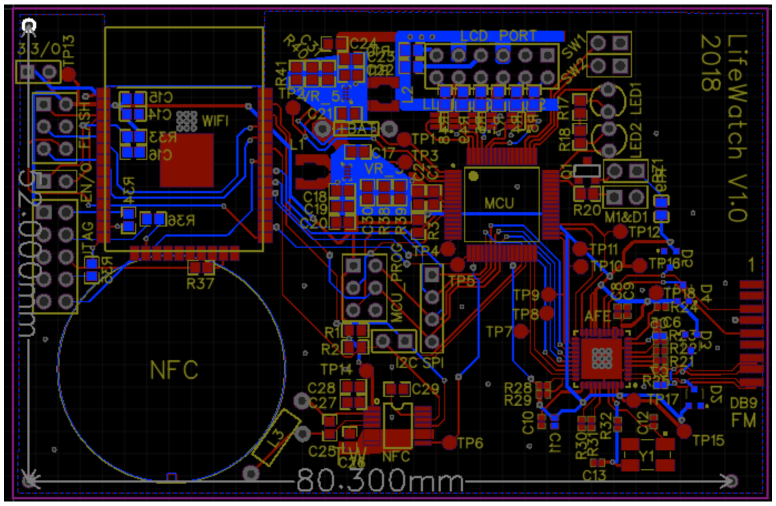


Figure 70. LifeWatch V1.0

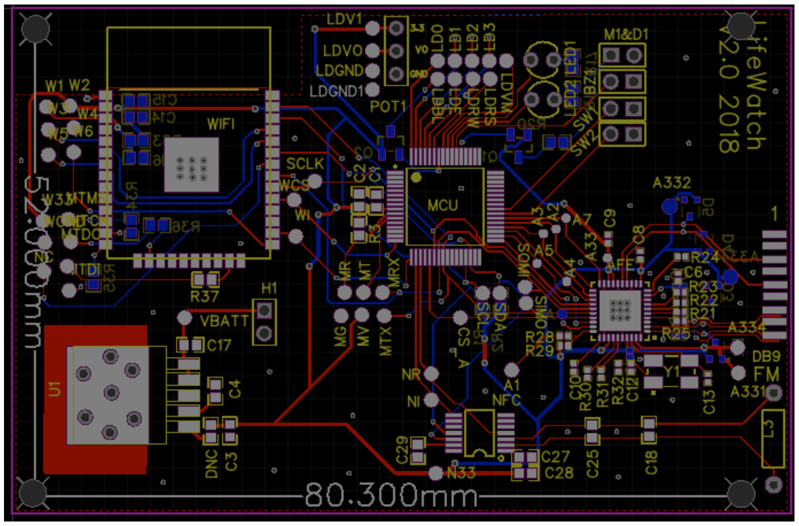


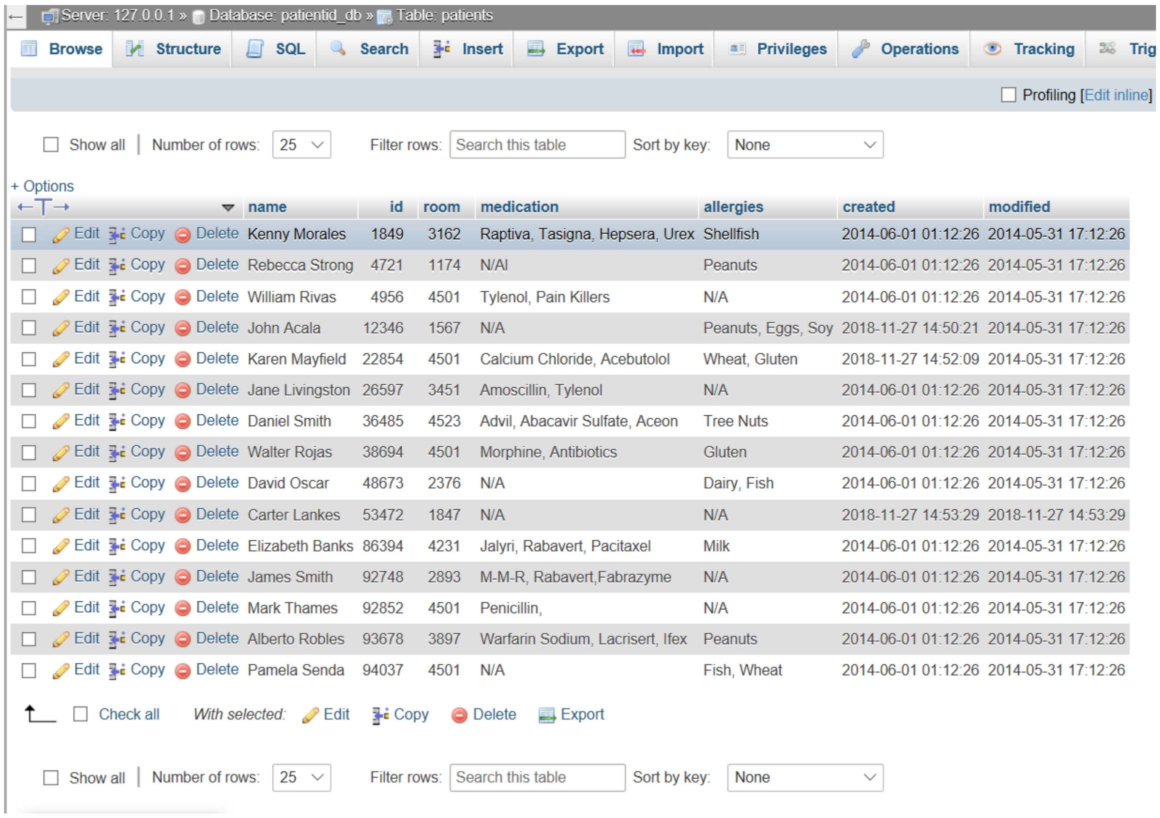
Figure 71. LifeWatch V2.0

8.12 Server

In order to set up our mock server, we used the open source program XAMPP on one of our laptops. We used this server to mimic a hospital setting in which all the patients information and logs are stored in a database within the server. XAMPP, among the things it includes, has the option of running an Apache mock server and a MySQL database.

Once the server was setup, we needed to make an API (Application Programming Interface) in order to store and retrieve data to/from our database in the server using HTTP requests. Doing a quick google search, the most popular API to make for this application was called a REST API which stands for REpresntational State Transfer. We used a step-by-step tutorial found in [39] to create this API. This API basically allowed us to use our WIFI chip to access information stored on our database within our server using HTTP requests, specifically GET requests. This API was written in PHP. For more in depth information, about setting up the API, please refer to the reference given in [39].

The following is a screen shot of the database along with the kind of information that we placed in it:



name	id	room	medication	allergies	created	modified
Kenny Morales	1849	3162	Raptiva, Tasigna, Hepsera, Urex	Shellfish	2014-06-01 01:12:26	2014-05-31 17:12:26
Rebecca Strong	4721	1174	N/Al	Peanuts	2014-06-01 01:12:26	2014-05-31 17:12:26
William Rivas	4956	4501	Tylenol, Pain Killers	N/A	2014-06-01 01:12:26	2014-05-31 17:12:26
John Acala	12346	1567	N/A	Peanuts, Eggs, Soy	2018-11-27 14:50:21	2014-05-31 17:12:26
Karen Mayfield	22854	4501	Calcium Chloride, Acebutolol	Wheat, Gluten	2018-11-27 14:52:09	2014-05-31 17:12:26
Jane Livingston	26597	3451	Amoscillin, Tylenol	N/A	2014-06-01 01:12:26	2014-05-31 17:12:26
Daniel Smith	36485	4523	Advil, Abacavir Sulfate, Aceon	Tree Nuts	2014-06-01 01:12:26	2014-05-31 17:12:26
Walter Rojas	38694	4501	Morphine, Antibiotics	Gluten	2014-06-01 01:12:26	2014-05-31 17:12:26
David Oscar	48673	2376	N/A	Dairy, Fish	2014-06-01 01:12:26	2014-05-31 17:12:26
Carter Lankes	53472	1847	N/A	N/A	2018-11-27 14:53:29	2018-11-27 14:53:29
Elizabeth Banks	86394	4231	Jalyri, Rabavert, Pacitaxel	Milk	2014-06-01 01:12:26	2014-05-31 17:12:26
James Smith	92748	2893	M-M-R, Rabavert,Fabrazyme	N/A	2014-06-01 01:12:26	2014-05-31 17:12:26
Mark Thames	92852	4501	Penicillin,	N/A	2014-06-01 01:12:26	2014-05-31 17:12:26
Alberto Robles	93678	3897	Warfarin Sodium, Lacrisert, Ifex	Peanuts	2014-06-01 01:12:26	2014-05-31 17:12:26
Pamela Senda	94037	4501	N/A	Fish, Wheat	2014-06-01 01:12:26	2014-05-31 17:12:26

Figure 72. Database, patient ID table created in our server

The following is an example of the kind of URL we used to send GET requests to the server:

http://192.168.0.1/patient_id/prodcut/read_one.php?id=12346

This url ending in ".php" accesses a specific PHP file in our REST API which takes an ID number, connects the database table, and retrieves the table row information linked with that ID. An important side note is that this php file "echo's" or prints the data which is how it returns it to the device which requested it. Without any format specifiers, the returned data will simply be in text format. In our application, we included HTML format such that it would be displayed in a more friendly viewer way on our Life Watch App which we will discuss in the next section.

To easily help in the development of our API and database, we used a free program called POSTMAN which is program targeted as an API development software. This allowed us to easily send requests, see the error codes and messages, and smoothly test our API.

8.13 Life Watch App

To develop our Life Watch app, we decided first that it would be developed as an android app. The following table summarizes why we chose Android over IOS:

Table 17. Android vs IOS for App Development

Parameter	Android	IOS
Cost	Free	$99/year
MIT App Inventor Supported?	Yes	No
Ease of developing (n of 10)	9	6

The three reasons stated above is the reason why we chose to develop an app using android. After selecting the operating system, we then had to chose through what development tool we would develop the app. A quick google search boiled down our options to two main development programs: Android Studio and MIT App Inventor 2. Android Studio is a more complete and professional way to develop applications, but the learning curve is more steep. MIT App Inventor is only good when developing simple applications, but the learning curve is a lot more generous.

In the interest of time and because of no prior experience in app development, we decided to use MIT App Inventor 2 to develop our app. Although its capabilities fall short in light of the possibilities in Android Studio, it had all the possible features we wanted to use: these features were mainly NFC read/write capabilities and Web connectivity. Its way of developing apps is also very intuitive; it uses code blocks to build the "actions" of the different interfaces within the app such as buttons, test boxes, etc. Using two example apps provided by MIT App Inventor 2, we successfully developed our app. The following is a screen shot of the "design view" of our application:

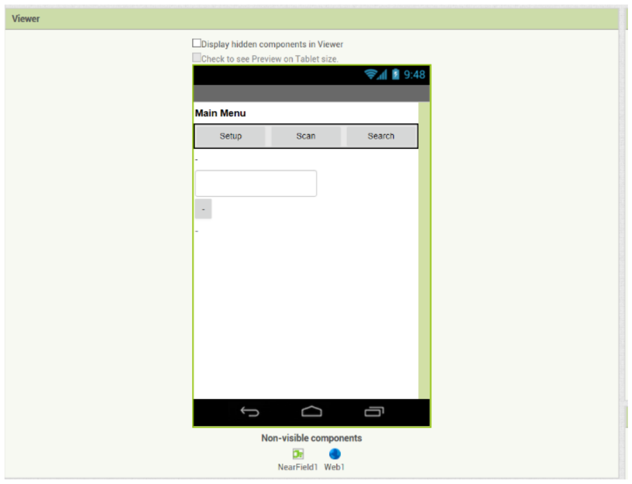

Viewer
Display hidden components in Viewer
Check to see Preview on Tablet size.
9:48
Main Menu
Setup
Scan
Search
Non-visible components
NearField1
Web1

Figure 73. App interface view

As far as the functionality of the app is concerned, this app was made to be an integral part of the actual Life Watch. This app has three main functionalities: Setup, Scan, Search.

 The "Setup" mode was developed to be used to initialize or "set-up" the Life Watch. Initially, the Life Watch only displays a dummy patient information. Using setup mode, the user enters an ID number, which after hitting "submit," can be written via NFC to the NFC tag inside the Life Watch; from there, the MCU takes it and gives it to the WIFI which does what is explained the "WIFI" section of the "Wireless Communication" heading.

The "Scan" mode is what can be thought as the "regular use" mode. After the Life Watch is initialized with an actual patient ID and information, the "Scan" mode can be used to scan the ID stored in the NFC tag. The app will done annex this ID to the end of a specific URL (See previous Server Section for more detail) which will then be used to execute a GET request to the patients database in the server. After successful completion, this information would then displayed in the app for the user or staff to read. The benefit of using the app is that a lot more information can be displayed in the app window than can be displayed on the LCD display of the Life Watch. Finally, the "Search" mode is an added feature to this device. Besides being able to GET patient information by reading an ID via NFC, we also wanted to include a way which di not involve NFC: a direct way to access information on the database by simply typing an ID number into the app.. This functionality was included to give more flexibility to the user since in some circumstances, it could be inconvenient to try to the scan the ID. Such cases would include a hectic scenario or an emergency in which the user or staff, for one reason or another, cannot NFC read the ID from the Life Watch. Essentially, this mode performs the same function as the "Scan" mode except the ID is entered manually by the user and not scanned from the Life Watch. The following flowchart explains the software functionality of the app:

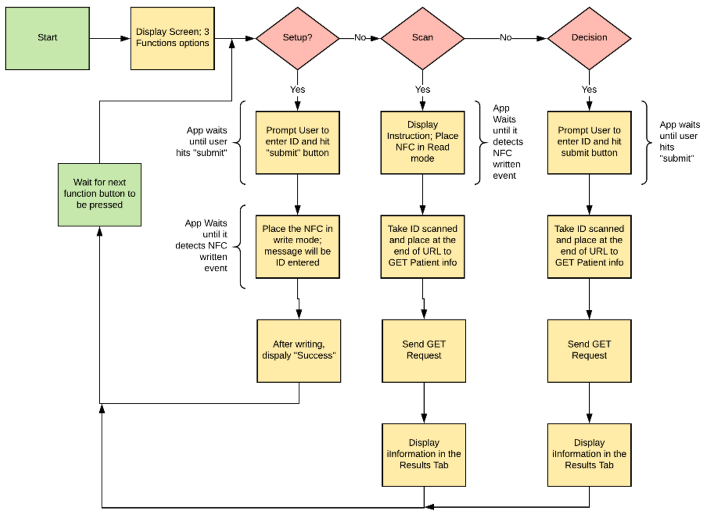

Start
Display Screen; 3 Functions options
Setup?
No
Scan
No
Decision
Yes
Yes
Yes
App waits until user hits "submit"
Prompt User to enter ID and hit "submit" button
Display Instruction; Place NFC in Read mode
App Waits until it detects NFC written event
Prompt User to enter ID and hit submit button
App waits until user hits "submit"
Wait for next function button to be pressed
App Waits until it detects NFC written event
Place the NFC in write mode; message will be ID entered
Take ID scanned and place at the end of URL to GET Patient info
Take ID scanned and place at the end of URL to GET Patient info
After writing, dispaly "Success"
Send GET Request
Send GET Request
Display iInformation in the Results Tab
Display iInformation in the Results Tab

Figure 74. Flowchart of how device works with application

8.14 Printed Housing Case

In order to protect the LifeWatch's valuable circuit board and components a housing case was designed, and 3D printed. Our thick casing makes the LifeWatch more durable, water resistant, and presentable. It is made from PLA and has a thickness of about 1 centimeter. It also has four loop holes attached at the bottom to secure it safely to the patient's arm using durable, recyclable, secure waterproof wristbands. The program Solidworks was used to create the overall design. Then due to small error of sizes we used a file, drill, and solder iron to finely tune it to fit our buttons, LCD screen, and other necessary external features.

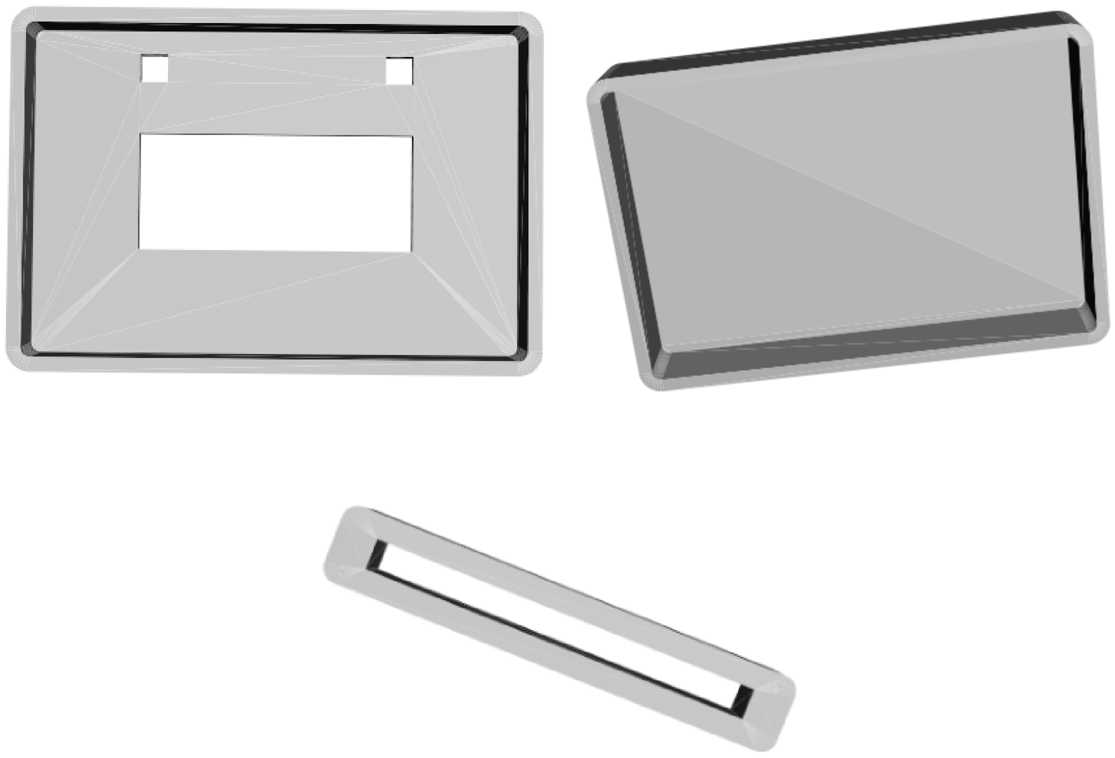


Figure 75. Case components Top, Bottom, Loop holes (left, right, bottom)

9 Administrative Content

This section is about the management of planning and building the device and what our expected milestones and dates to complete them were. In this portion we will also compare our estimated cost with our actual cost and talk about what was used in the end and what was not. Lastly, we will talk about who was assigned to what portion of the project.

9.1 Milestone Discussion

Our Milestones were all met fairly well. Coming up with a project idea was probably one of the most difficult challenges since what we chose was going to be our project we were to work on for the next two semesters. Not only was it difficult because of that commitment of two semesters, but also because the team was trying to find a balance between having fun and learning. We chose it carefully to make sure it was challenging, but still feasible. The group was interested in participating in a project in which the team members would all learn; also, the team members also expressed their desire to make something that would be presentable in a interview for a future job. Our process consisted of writing out many ideas the two weeks we had and over time condensing them down to the best one and figuring out what each idea would take to make it. Also we had more of a push for the smart hospital watch idea due to the fact that members of our group had personal accounts of mistakes they had encountered in hospitals themselves, and this came about as being a way to help solve those issues. Once the team had a clear idea, things began to "flow" better into place: with a clear idea, also comes a clear goal, and with a clear goal comes a sense of purpose. Now that there was a clear sense of purpose, the team could now move forward with the actual paper deadlines.

Then from there, for the divide and conquer, we all decided what we were interested in and chose each aspect of the project that we could work on in depth. The team used this divide and conquer activity to set general goals for the project. This divide and conquer paper was to be used as the foundation for the rest of the paper writing. Once we decided who was working on what we split the research up between each other.

The research that the team began to do right after the divide and conquer fit in perfectly with the next paper objective which was to be the 60 page draft. We focused on our specific portions of the device and explored the possible components to use and also learned how the technology worked, which consisted of the major bulk of our 60 page draft. In these first 60 pages, the group experienced the biggest knowledge growth up to date. The team had to learn things not taught inside the classroom.

Next came the 100 page draft, building off the 60 page draft, and we focused on the standards and constraints of our device in the hospital setting, explaining why we chose our components, how to test them, and also explaining our budget and milestones. The writing about standards and constraints gave us a "real world" view of engineering projects. Once we knew what components we were going to choose, we also began to focus on the software aspect of the project. For each major function, we began to explore online and think of the "how" we were going to implement that function on the hardware.

During the 100 page draft we had to order our components very quickly in order to test them within the weeks after to include in our final document. Testing our components was a big learning experience of realizing we needed to order more things than we realized such as breakout boards, heat sink, inductor at specific value, and more which put us in a bit of a crunch for testing. Also when testing we encountered many problems along the way by overlooking small details or improperly setting up the hardware. We learned when performing an experiment always plan it, simulate it, then run it to save time and effort instead of just doing it. Although that may work sometimes the amount of small details for components that can be overlooked is incredible and often happens when you take the "just do it" approach. This portion of the project was definitely a huge challenge and learning experience for us, but in the long run will help us in senior design two.

We worked from the 100 page to the 120 page paper. Although this seemed to be the smallest amount of writing since the 10 page Divide and Conquer paper, this was one of the hardest 20 pages to write. The difficulties mainly grew from the format requirements. Our document for some reason kept accumulating "white space" while we worked on it and no matter how hard we tried to rid the document of whitespace, more and more just kept appearing as we went on. It spread through out our paper like a deadly virus. Another very challenging aspect to getting to the 120 paper marker was the program we used to work together. Our collaboration website we used to make our document and other project related materials was "OneDrive". At first it ran very smoothly, but as time went on and more and more pages were added it began to lag very badly. On top of the lagging that was occurring OneDrive also had a tendency to crash, erase, or add in lots of white space. It almost had a mind of its own at times which made it very frustrating to polish the document and make sure it was good to go for submission. Luckily we were able to press through all those interface challenges and give the finished document.   For these last 20 pages, we mainly focused on fixing the mistakes pointed out by our Senior Design advisor. We also focused on being more specific on our details; we included all of the component schematic drawings and possible PCB vendors and assemblers information that we may use in senior design two. These last 20 pages were just the finishing touch of the document, and the first feel of what Senior Design 2 was going to feel like since we began to develop a crude prototype for our system.

We then used the very crude prototype that we developed as the basis for our building block in Senior Design 2, which helped us in the long run. We had an understanding of how the basics of our device worked, operated, and interacted with it's other components giving us more time in senior design two to focus on fine tuning our device instead of making it from scratch. We also learned which goals would become stretch and which goals were achievable in our given time.

Our first major milestone in senior design two was getting the NFC to successfully work and interact with our smart device. We had troubles connecting it to our MSP430 at first due to figuring out which pins were used for I2C. We also had difficulties due to one of the NFC chips being burnt and not working, causing us to figure out if these was a major issue until using another NFC chip to test with. Frustrating it was, but very rewarding in the end.

The next milestone was to send for our PCB, which we created in easyeda. We had never created a PCB board through out our education at UCF so this was another challenge especially since there are so many minor details to the PCB that can be overwhelming. Luckily we kept it simple with 2 layers and just focused on the general purpose of the PCB: to make our circuits physical. We didn't get caught up too much in the infinite PCB errors that could occur, but we did make sure to separate our AFE from the WIFI chip so the frequencies did not interfere. It was highly suggested. We then ordered the specific parts and took it to QMS to become soldered.

Now we have our board back and are doing initial testing and to no surprise we had errors, but luckily only a few. The main errors were the voltage regulators as stated before. Everything else we were able to verify with tests and they all worked properly. This led us to changing our voltage regulators to the LDO REG104.

As the semester went on we gave our CDR to our peers and also had a midterm meeting with our professor where he was able to allow us to drop two of our goals, heart beat sensor and WIFI location, that would be too difficult to implement in the amount of time we had left. With time closing in on us we decided to focus on perfecting the NFC interacting with the WIFI module and printing to the LCD screen. We also began working on our second PCB design and possibly an Application to use with our device.

The next achievement was sending out our second version of the PCB and this time ordering 2 components of everything so we could make 2 boards to test with. At the same time we focused on printing to the LCD screen, getting the WIFI module and a server to communicate via WIFI, the NFC to interact with the LCD. The WIFI communication led us to using a hot spot on one of member's phone because they both must be on the same network. We also began to work on the design of the casing that would hold the PCB, battery, etc. Also one of our members began developing the application that was a stretch goal.

Our second PCB comes back and we take it to QMS once again to get 2 boards soldered. They came back and we tested the regulator first to make sure it works properly. It does and then we test the rest of the board piece by piece and there are a few differences, but nothing major. Our PCB is good to go and we now work to finish the code and WIFI communication.

Finally, we are able to get everything to work. The WIFI to communicate with the server and MCU, the NFC to send text to the LCD and interact with the smart device with good range, our application to interface between everything, and do it continuously. We were also able to make a panic mode to send out a noise and securely put it into its casing.

Then we presented it to our panel of faculty, and unfortunately during transport something became corrupted within and our demonstration did not fully work due to the LCD not working. They then gave us till the end of the day to fix it, which we did. Thus completing our senior design project.

As a team, we feel that we have successfully completed all the goals we set to meet after the teacher demonstration. We also feel that we have successfully completed the main mission of Senior Design by stretching ourselves to learn new things, work in a group setting effectively to complete a task, and do it in a realistic and professional way. We started with a lot of goals that we later realized could not be done effectively in the time we were given due to our novice skills for real life circuitry production, lack of coding skills, and other academics we had to also focus on. It seems we accomplished a lot with the Lifewatch, but at the same time only scratched the surface of what it could really be when we started planning in senior design 1. There is still a lot of potential with the Lifewatch.

Senior Design I:

Table 18. Administrative

Task	**Duration**	**Date**
Project Ideas	2 weeks	May 14-21
Project Selection	1 week	May 21-28
Divide and Conquer	1.5 weeks	June 8
Research and Documentation	7 weeks	June 8–July 30
60 Page Draft	4 weeks	June 8–July 6
100 Page Draft	7 weeks	June 8–July 20
Final Document	7 weeks	June 8–July 31

Table 19. Technical

Task	Duration	Date
Component Research	4 Weeks	June 8 – July 6
Order Components	2 Weeks	July 6 – July 20
Test Components	1 ½ Weeks	July 20 – July 29
Design Schematics	1 ½ Weeks	July 20 – July 29

Senior Design II

Table 20. Administrative SD2

Task	Duration	Date
Construct prototype design	4-5 weeks	Aug 20 – Sept 17
Test and redesign	2 weeks	Sept 17- Oct 1
Finalize Prototype	2 weeks	Oct 1- Oct 15
Peer Presentation		Sept. 14

Final Presentation		Nov. 27
Final Report		Dec. 3

Table 21. Technical SD2

Task	Duration	Date
Breadboard	2 Weeks	Aug 20 – Aug 31
Design PCB Board	2 Weeks	Aug 20 – Aug 31
Software	4 Weeks	Aug 20 – Sept 17
Attach components to Board	1 Week	Sept 17 – Sept 24
Test and modify	1 Week	Sept 24 – Oct 1
Verify prototype	2 Weeks	Oct 1- Oct 15

9.2 Budget and Finance

One of the most important constraints in any project whether in school or in the real world is budget. It is one of the first constraints that we took into consideration when aiming for goals as far as technology that we wanted to implement. For example, a technology that we were considering in implementing on the hospital watch was wireless charging. In reality, this way of charging is a lot more convenient for a setting such as a hospital, but it is also much more expensive. So not only did budget and economic factors put a physical constraint on the parts we could and could not order, but through that, it also put a constraint on our imagination and what we would have wanted to implement.

Despite this, our team did not perceive the budget as an enemy or as a hindrance to our imagination but rather as a challenge that we wanted to overcome. Our team believed that working with a budget and being able to design a quality product with a limited budget was a skill and challenge were doing. As with every project, it is always best to be safe than sorry We applied this approach to our estimated budget, we wanted to make sure we did not pass our budget during this project: for this reason, we overestimated our budget and our expenses. Our budget was expected to be around $200 total for our components and other necessities to build our device and so far we have spent around $75.00, which is not buying a majority of the components already.

Throughout Senior Design 1, we sought many ways to save money especially in the ordering of our parts. Although the team felt like we overestimated our budget, we still looked for ways to be financially wise. We looked for vendors or websites that included benefits such as free shipping; we also tried to combine our parts into one order. In many cases, shipping takes up a significant amount of what we paid when we ordered parts.

As we went through Senior Design 2 we had to account our PCB boards, the enclosure for our device, and the actual wrist straps that meets the hospital standards of comfort as well. For the strap we looked into a disposable wrist strap or reusable washable ones: something that is inexpensive yet of very good quality. We went with the disposable wrist straps that you would get when entering a club because they are very durable, water proof, and cheap. For PCB boards, we are looked to vendors that were involved in simple design since our design is not very complex: this saved us even more money. We did order additional components along the way and and parts that we messed up during the testing, we had to re order again.

The following is a break down of our estimated cost. This was the very first initial budget our team came up with in one of the earliest team meetings in this course. Looking at some of these numbers now (for example the PCB board cost, and wireless chip), we feel that we overestimated, but the team is alright with it because it works to our benefit anyways. This overestimation was partly due to our inexperience and lack of knowledge. Below the table of cost is our final cost we came to at the end of our senior design two semester.

Table 22. Estimated Cost

Item	**Quantity**	**Price per Unit**
PCB	1	**~$80**
	1	**~$10**

Microcontroller		
Sensors	**2**	**~$40**
LCD Screen	**1**	**~$10**
NFC Tags	**1**	**$0.10**
Enclosure	**1**	**~$20**
Wireless chip	**1**	**~$20**
LI-ION Battery	**1**	**~$10**
Disposal Strap	**1**	**~$3**

Total Cost		**$195**

Table 23. Actual Cost

Item	Quantity	Price per Unit
3.7V 2.5Ah Adafruit Battery Lithium-ion	1	$14.95
RF430CL330H (RFID/NFC Chip)	3	$1.29
PA0033 (NFC Breakout Board)	1	$3.69
REG104FA-3.3500 (Voltage Regulator)	4	$6.52
ESP-WROOM-32	1	$3.80
ESP -WROOM-02	1	$2.70
NHD-0216HZ-FSW-FBW-33V3C	1	$11.60

(Character LCD)		
MSP430FR4133IPMR (MCU)	1	$2.82
MSP-EXP430FR4133	1	$14.49
Nellcor-DS100 Pulse Oximeter Probe	1	$20
TI AFE4400	1	$2.50
TP4056 Li-ion battery charger	1	$1.25
Wrist Strap	100	$0.08
PCB	10	$0.50
TOTAL		$85.61

At the end of our senior design project we still kept our under our budget by a good amount. Only made a few changes for components, which only increased us by a few dollars. Lifewatch is an affordable product overall.

9.3 Assigned Roles

The role assignation on this project was done on a voluntary basis. As explained in a previous section, the different aspects of the project were discussed and the individual team members committed to doing a specific part they were interested in doing. The following is a table of the primary and secondary roles for all of our team members:

	Primary Roles	Secondary Roles
Carter Lankes	• RFID/NFC ○ Deciding which type of wireless communication to use for sending and receiving the patient's personal information • Power Supply ○ Choosing a rechargeable power supply that is able to power the device for a long period of time with its components, sensors, and etc. • Voltage Regulators ○ Choosing an efficient DC to DC converter that is able to meet all the voltage requirements of the sensors, micro controller, and other necessary components	• Localization research
Josue Ortiz	• Indoor Localization ○ Research how indoor localization has been implemented in other systems ○ Research the optimal technology to do localization • WIFI ○ Research and compare various WIFI modules and chose the best one for our design • Bluetooth ○ Research and compare various BLE modules and chose the best one for our design	• Program Coding

William Toledo	• Microcontroller ○ Researching a power efficient microcontroller that would be usable on a wrist worn device • Display ○ Research different LCD displays to find a practical display for a medically enabled smart watch • Communication protocols ○ SPI and I2C communications research and implementation	• Power Supply, NFC, server, general circuit analysis and research, serial communications, localization
John Alcala	• Pulse Oximetry ○ Research how pulse oximetry works and best way to implement	• Microcontroller

Appendix A – Citations

[1] Y. Akao, *The Customer Driven Approach to Quality Planning and Deployment.* Minato, Tokyo: Asian Productivity Organization, 1994

[2] John R. Hauser and Don Clausing, "House of Quality" *Harvard Business Review,* 1988 https://hbr.org/1988/05/the-house-of-quality

[3] "Patient Identification", World Health Organization http://www.who.int/patientsafety/solutions/patientsafety/PS-Solution2.pdf

[4] Bradley Mitchell "What Is a Wireless Access Point?" Lifewire, 2018. https://www.lifewire.com/wireless-access-point-816545

[5] Understanding the Network Terms SSID, BSSID, and ESSID., Juniper. https://www.juniper.net/documentation/en_US/junos-space-apps/network-director2.0/topics/concept/wireless-ssid-bssid-essid.html

[6] Texas Instruments, "CC3220MODx and CC3220MODAx SimpleLink™ Wi-Fi® CERTIFIED™ Wireless MCU Modules", March 2017. http://www.ti.com/lit/ds/symlink/cc3220moda.pdf

[7] Sparkfun, WiFi Module – ESP8266, https://www.sparkfun.com/products/13678

[8] "What is the difference between Bluetooth and Wi-Fi?" Techopedia, April 2017 https://www.techopedia.com/2/27881/networks/wireless/what-is-the-difference-between-bluetooth-and-wi-fi

[9] "Bluetooth Beacons: A Beginners Guide", BlueMaestro "https://www.bluemaestro.com/ultimate-guide-bluetooth-beacons/"

[10] Microchip, "RN4020 Bluetooth Low Energy Module", http://www.mouser.com/ds/2/268/50002279A-515512.pdf

[11] Murata "UHF MAGICSTRAP® Preliminary Data Sheet LXMSJZNCMF-198", https://www.mouser.com/datasheet/2/281/Murata_07102017_LXMSJZNCMF_198(rev0%202)-1186223.pdf.

[12] ST "Dynamic NFC/RFID tag IC with 64-Kbit EEPROM NFC Forum Type 4 Tag and I2C interface" https://www.mouser.com/datasheet/2/389/m24sr64-y-1156150.pdf

[13] Texas Instruments "RF430CL330H Dynamic NFC Interface Transponder", November 2012. http://www.ti.com/lit/ds/symlink/rf430cl330h.pdf

[14] James Thrasher "RFID vs. NFC: What's the Difference?" *RFID Insider*, https://blog.atlasrfidstore.com/rfid-vs-nfc

[15] Near Field Communication. http://chittagongit.com/icon/near-field-communication-icon-2.html

[16] SparkFun Electronics PRT-13851, Digi-Key Electronics. https://www.digikey.com/product-detail/en/sparkfun-electronics/PRT-13851/1568-1493-ND/6605199

[17] PKCELL "Li-Polymer Battery Technology Specification", https://cdn-shop.adafruit.com/product-files/2750/LP552035_350MAH_3.7V_20150906.pdf

[18] Illinois Capacitor RJD3555HPPV30M, Digi-Key Electronics. https://www.digikey.com/product-detail/en/illinois-capacitor/RJD3555HPPV30M/1572-1627-ND/6159145

[19] Texas Instruments LM1084IT-ADJ/NOPB, Mouser Electronics. https://www.mouser.com/ProductDetail/Texas-Instruments/LM1084IT-ADJ-NOPB?qs=X1J7HmVL2ZFnaHQkkplutQ%3D%3D&gclid=CjwKCAjwg_fZBRAoEiwAppvp-fQync2Lrv4UjeP7b-njsK8oaKaICYoSi1nOIiLV5M3JCFI58TFz8hoCIOwQAvD_BwE

[20] Texas Instruments LM2576,LM3420,LP2951,LP2952 Battery Charging. http://www.ti.com/lit/an/snva557/snva557.pdf

[21] Hooseok Lee, Et al. “Reflectance pulse oximetry: Practical issues and limitations” *ICT Express*, 2016, pp. 195-198 https://www.sciencedirect.com/science/article/pii/S2405959516301205#br000005

[22] Measuring heart rate and blood oxygen levels for portable medical and wearable devices, *Embedded Computing Design*. http://www.embedded-computing.com/embedded-computing-design/measuring-heart-rate-and-blood-oxygen-levels-for-portable-medical-and-wearable-devices

[23] PQRST Complexes in the ECG waveform, *ResearchGate*. https://www.researchgate.net/figure/PQRST-complexes-in-the-ECG-waveform_fig1_262384778

[24] A. Arcelus, M. Sardar and A. Mihailidis, "Design of a capacitive ECG sensor for unobtrusive heart rate measurements," *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, Minneapolis, MN, 2013, pp. 407-410.

[25] Maxim Integrated, MAX20112. https://www.maximintegrated.com/en/products/sensors/MAX30112.html

[26] Oliver J. Woodman, “An introduction to inertial navigation” University of Cambridge Computer Laboratory. August 2007. http://www.cl.cam.ac.uk//techreports/UCAM-CL-TR-696.pdf

[27] Yang, C.-C., & Hsu, Y.-L. (2010). A Review of Accelerometry-Based Wearable Motion Detectors for Physical Activity Monitoring. *Sensors (Basel, Switzerland)*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3231187/

[28] Texas Instruments, “Understanding the I^{2}C Bus”, June 2015. http://www.ti.com/lit/an/slva704/slva704.pdf

[29] Sparkfun “Serial Peripheral Interface” https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

[30] Adafruit, Vibrating Mini Motor Disc https://www.adafruit.com/product/1201?gclid=CjwKCAjwg_fZBRAoEiwAppvp-aGabSejIOhC3UTugKxrruCKieMzKHxnghnE7Yr9aK7ILsZelU3e_hoCe3oQAvD_BwE

[31] Sensible Micro Corporation, “MCU Microcontrollers – Costs Climb” https://sensiblemicro.com/mcu/

[32] Retro Nintendo, “NES Motherboard Identification” “http://www.retronintendoreviews.com/identifying-your-nes-cpu-revision/

[33] Practical Components, Thin Quad Flat Pack Dummy Component. http://www.practicalcomponents.com/Dummy-Components/product.cfm?Thin-Quad-Flat-Pack-%28TQFP%29-Dummy-Component-9A06B8579E20A566

[34] White, Flye “Successive Approximation ADC Block Diagram”. https://commons.wikimedia.org/wiki/File:SA_ADC_block_diagram.png

[35] All About Circuits, Logic Signal Voltage Levels. https://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/

[36] NetSec.news “What is the Definition of a HIPAA Covered Entity?” https://www.netsec.news/definition-hipaa-covered-entity/

[37] U.S Department of Health & Human Services “Protected information”, https://www.hhs.gov/hipaa/for-professionals/privacy/special-topics/de-identification/index.html#protected

[38] NFC “Security Concerns with NFC Technology” http://nearfieldcommunication.org/nfc-security.html

[39] Mike Dalisay. “Simple REST API in PHP? Step By Step Guide.” *Code of a Ninja*. 2017. Accessible at: https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html

Appendix B – Code for MSP MCU

Code for the main MCU, MSP4304133, programmed using Code Composer Studio 8.1.0.

Each section is a separate file, either a c or h file.

10 Master.c

```
#include <master.h>
#include <msp430.h>




typedef enum SPI_ModeEnum{
    IDLE_MODE,
    TX_CMD_MODE,
    RX_REG_ADDRESS_MODE,
    TX_DATA_MODE,
    RX_DATA_MODE,
    TIMEOUT_MODE
} SPI_Mode;

SPI_Mode MasterModeWIFI = IDLE_MODE;     //Start in idle mode

SPI_Mode SPI_WIFI_WRITE(uint8_t* id, uint8_t count);
SPI_Mode SPI_WIFI_CMD(uint8_t wifi_cmd, uint8_t count);
SPI_Mode SPI_WIFI_READ(uint8_t count);
void read_patient_info_WIFI(void);

//software flags

int LCD_Screen = 3;
int mcu_to_wifi = 0;
int SW_flag_debounce = 0;
int both_sw_on = 0;
int emergency_mode = 0;
int SPI_CMD_Receive=0;


int wifi_instruction = 0;
int nfc_int = 0;
int wifi_int = 0;
int wifi_cmd = 0;
int wifi_cmd_read = 0;


unsigned char read_data[200];
unsigned char nfc_read[200];

int HighestLCDScreen = 3;

volatile unsigned int test = 0;
```

```
unsigned int flags = 0;
unsigned int temp = 0;

uint8_t TransmitCmd = 0;
uint8_t ReceiveBuffer[200] = {0};
uint8_t RXByteCtr = 0;
uint8_t ReceiveIndex = 0;
uint8_t TransmitBuffer[MAX_BUFFER_SIZE] = {0};
uint8_t TXByteCtr = 0;
uint8_t TransmitIndex = 0;

//unsigned char NDEF_Application_Data[] = RF430_DEFAULT_DATA;

unsigned char test_data[] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
unsigned char patient_name[16] = "Smith, John";
unsigned char patient_id[8] = "12345";
unsigned char patient_room[4] = "101A";
unsigned char patient_allergies[28] = "No known Allergies";
unsigned char patient_more[20];


unsigned char watch_time[8] = "12:05am";
int time_updated =0;
int seconds=50;
int minutes=5;
int hours=12;
int isPM=0;
unsigned char watch_date[11];




unsigned char CRC_Data[] = {1,2,3,4,5,6,7,8,9};

unsigned char Cmd = 0;  //Command byte for SPI mode
unsigned char read_complete = 0;
unsigned char rx_byte_count = 0;
unsigned char tx_byte_count = 0;
unsigned int Results[11] = {0,0,0,0,0,0,0,0,0,0,0};

/****************************************************************************/
/* Code-binary that opens on ETW and re-trims LF oscillator to below 280kHz */
/****************************************************************************/
unsigned char func_retrim_osc[] = {
        0xB2, 0x40, 0x11, 0x96, 0x10, 0x01,
        0xB2, 0x40, 0x60, 0x03, 0x18, 0x01,
        0x30, 0x41
};


int main(void)
{
      WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
```

```
/*
 * initalize clock
 * initialize gpio
 *      switches (possibly for error mode)
 *      all unused pins to power save  mode
 * initialize com (i2c and spi)
 * init peripheral
 *
 *      lcd welcome screen
 *      nfc
 *          control register
 *          setup interupt
 *      wifi
 *          get time?
 *      afe
 *          setup
 *          start reading?
*       beep speaker
*       update display
*
*
*
*
*   interrupts
*   port1_vector
*       wifi_int
*           receive data from wifi
*           spi
*           led?
*
*       nfc_int
*           nfc was written or read
*           -written
*               update screen
*               update patient info via wifi
*           -read
*               turn on screen if off
*               beep
*               led?
*
*   port2_vector
*       Switches
*           debounce code
*           if pass
*               move to next screen
*               check if second button pressed
*               timer to check 10 times in 2 seconds
*                   if pass emergency mode!
*
 *      afe_int
 *          adc ready
 *              spi read data from register
 *              increment counter
 *              when counter reaches certain amount
 *                   update heart rate with formula
*           adc else
*               update screen to error
*               beep?
*
*
*  timers
*            main timer to check wifi location every XX minutes
*
*
*            secondary timer for pwm if needed
*               also for emergency response

*            32khz for clock update
*               every XX min, wifi time get
*
 *
 */


      initClocks();   //initialize to 16Mhz
      static const int ms = _1ms_16MHz;
      initGPIO();     //init gpio

      LCDinit();      //initialize LCD to 4bit mode
    LCDsetCursor(0,0);
    LCDprint("LifeWatch  v2.0");
    LCDsetCursor(1,1);
    LCDprint("Initializing");

    __delay_cycles(40000000);

    initComm();     //init i2c and spi
    initNFC();

    LCDclear();
    LCDsetCursor(0,0);
    LCDprint("Connect to WIFI");
    LCDsetCursor(0,1);
    LCDprint("Get Time & Date");

    wifi_cmd = 0x01;    //get time
    wakeup_wifi();


    TA0R = 0;
    TA0CCTL0 &= ~CCIFG;    //Clear Flag
    TA0CCTL0 |= CCIE;      //Enable interrupts



    while (1){
        __bis_SR_register(LPM3_bits+GIE); //LMP with int
        __no_operation();
        //if both switches were held down for a while
        if(both_sw_on){
            LCDclear();
            LCDsetCursor(1,0);
            LCDprint("Emergency Mode");

            if (emergency_mode){
                LCDsetCursor(4,1);
                LCDprint("ENABLED");    // go into emergency mode
                BUZZ_OUT |= BUZZ;       // turn on buzzer
                //VIB_OUT |= VIB;       // turn on motor, needs pwm?


            }else{
                LCDsetCursor(3,1);
                LCDprint("DISABLED");   // Exit emergency mode
                BUZZ_OUT &= ~BUZZ;      // turn off buzzer
                //VIB_OUT &= ~VIB;       // turn off motor, turn off
pwm/timer?
            }
            both_sw_on = 0;
        }

        //WIFI woke up device
        if(wifi_int){

            if(wifi_cmd == 0x01){       //get time

                LCDclear();
                LCDprint("Wait for wifi to wake");    // Exit emergency mode

                SPI_WIFI_CMD(0x01, 0);  //send command
                P1IE   |=  WIFI_INT;     //Enable interrupt from wifi
                __bis_SR_register(LPM3_bits+GIE); //LMP3 w int
                __no_operation();

                LCDclear();
                LCDprint("wifi woke, get time");    // Exit emergency mode
                P1IFG  &=  ~WIFI_INT;    //clear wifi int
                P1IE   &=  ~WIFI_INT;    //Disable interrupt from wifi
                SPI_WIFI_READ(19);        //get time

                int i;

                for(i=0;i<10;i++){
                    watch_date[i]=ReceiveBuffer[i+1];
                }
                watch_date[10] = 0;



                for(i=11;i<18;i++){
                    watch_time[i-11]=ReceiveBuffer[i+1]; //put into string
format "HH:MMpm"
                }
                watch_time[7] = 0;

                hours = (watch_time[0]-0x30)*10 + (watch_time[1]-0x30);
                minutes = (watch_time[3]-0x30)*10 + (watch_time[4]-0x30);
                if(watch_time[5]=='p'){
                    isPM=1;
                }
                else{
                    isPM =0;
                }
                seconds = 0;

                LCD_Screen = 1;
                updateLCD();

                __delay_cycles(400000);

                LCD_Screen = 3;
                updateLCD();

                SW_IE   |= (SW1|SW2);                // pin interrupt enabled
                TA0R = 0;
                TA0CCTL0 &= ~CCIFG;     //Clear Flag
                TA0CCTL0 |= CCIE;       //Enable interrupts

            }


            if(wifi_cmd == 0x02){           //get patient id
                LCDclear();
                LCDprint("CMD 2");

                SPI_WIFI_CMD(0x02, 0);
                P1IE   |=  WIFI_INT;     //Enable interrupt from wifi
                __bis_SR_register(LPM3_bits+GIE); //LMP3 w int
                __no_operation();
                P1IFG  &=  ~WIFI_INT;    //clear wifi int
                P1IE   &=  ~WIFI_INT;    //Disable interrupt from wifi

                //Update display

                LCDclear();
                LCDprint("Connect to WIFI");
                LCDsetCursor(0,1);
                LCDprint("ID: ");
                LCDsetCursor(4,1);
                LCDprint(patient_id);
                SPI_WIFI_WRITE(patient_id, 5); //write the patient ID to the
wifi
                P1IE   |=  WIFI_INT;     //Enable interrupt from wifi
                __bis_SR_register(LPM3_bits+GIE); //LMP3 w int
                __no_operation();
                P1IFG  &=  ~WIFI_INT;   //clear wifi int
                P1IE  &=  ~WIFI_INT;    //Disable interrupt from wifi

                LCDclear();
                LCDprint("Get Chunks");

                //SPI_WIFI_READ(1);          //know how many chunks

                LCDclear();
                LCDprint("READ INFO");
                LCDsetCursor(0,1);
                LCDprint("Chunk ");
                LCDsetCursor(6,1);


                int chunk=0;
                while(chunk<2){

                LCDsetCursor(6,1);
                //LCDprint(chunk+0x30);
                __delay_cycles(1000);
                SPI_WIFI_READ(30);          //wait for wifi to send patient
info

                int i;
                    for(i=0;i<30;i++){

                        read_data[chunk*30+i] = ReceiveBuffer[i];
                    }
                    chunk++;
                }
                int i;
                for(i=0;i<200;i++){
                    ReceiveBuffer[i] = read_data[i];
                }
                read_patient_info_WIFI();

                LCDclear();
                LCDprint("SUCCESS!");

                LCD_Screen = 0;
                updateLCD();
            }



//              LCDclear();
//              LCDsetCursor(0,0);
//              LCDprint(ReceiveBuffer);
//              LCDsetCursor(0,0);
//              LCDprint(" ");
            //SPI_WIFI_CMD(0x01, 8);
            wifi_cmd = 0;
            wifi_int = 0;
            P1IFG &= ~WIFI_INT;      //Clear pending flag from wifi
            P1IE  |=  WIFI_INT;      //Enable interrupt from wifi
            __no_operation();

        }

        //NFC woke up device
        if(nfc_int){
            //Disable RF for safety
               Write_Register(CONTROL_REG, INT_ENABLE + INTO_DRIVE); //clear
control reg to disable RF
               __delay_cycles(75.0*ms);
               int flags = Read_Register(INT_FLAG_REG); //read the flag
register to check if a read or write occurred
               int temp = 0;
               Write_Register(INT_FLAG_REG, EOW_INT_FLAG + EOR_INT_FLAG);
//ACK the flags to clear

               Write_Register(INT_ENABLE_REG, 0);


               LED_OUT &= ~RED_LED;      //Turn off red led

               if(flags & EOW_INT_FLAG) //check if the tag was written
               {
                   //tag was updated, so we should read out the new data
                   //read out the data
                   //Read_Continuous(0, read_data, 200);
                   __no_operation(); //breakpoint here to examine the data

                   //show that tag was written by blinking LED 3 times
                   for(temp = 0; temp < 3; temp++){
                       LED_OUT |= RED_LED;
                       __delay_cycles(30.0*ms);
                       LED_OUT &= RED_LED;//clear LED
                       __delay_cycles(30.0*ms);

                   }
                   LCDclear();
                   LCDsetCursor(0,0);
                   LCDprint("NFC Written: Get");
                   LCDsetCursor(0,1);
                   LCDprint("info from RF430");

                   read_patient_info_NFC();               //move info from RF
memory to variables
```

```
               }
               else if(flags & EOR_INT_FLAG) //check if the tag was read
               {
                   __no_operation();

                   //show that tag was read with LED by holding it on for 1
second
                   LED_OUT |= RED_LED;
                   __delay_cycles(80.0*ms);
                   LED_OUT &= ~RED_LED;//clear LED
                   LCDclear();
                   LCDsetCursor(0,0);
                   LCDprint("NFC Read");
                   LCDsetCursor(11,0);
                   LCDprint(patient_id);
                   LCDsetCursor(1,0);
                   LCDprint(patient_name);
               }

               flags = 0;
               nfc_int = 0; //we have serviced INTO

               //Enable interrupts for End of Read and End of Write
                Write_Register(INT_ENABLE_REG, EOW_INT_ENABLE +
EOR_INT_ENABLE);
                int test = Read_Register(INT_ENABLE_REG);

               //Configure INTO pin for active low and re-enable RF
               Write_Register(CONTROL_REG, INT_ENABLE + INTO_DRIVE +
RF_ENABLE);

               //re-enable INTO
               P1IFG &= NFC_INT;
               P1IE |= NFC_INT;
               if(wifi_cmd==0x02){
                   wakeup_wifi();
               }

               __no_operation();

        }
    }

}

void initClocks(void){
    // Configure one FRAM waitstate as required by the device datasheet for
MCLK
    // operation beyond 8MHz _before_ configuring the clock system.
    FRCTL0 = FRCTLPW | NWAITS_1;

    __bis_SR_register(SCG0);    // disable FLL
    CSCTL3 |= SELREF__REFOCLK;  // Set REFO as FLL reference source
    CSCTL0 = 0;                 // clear DCO and MOD registers
    CSCTL1 &= ~(DCORSEL_7);     // Clear DCO frequency select bits first
    CSCTL1 |= DCORSEL_5;        // Set DCO = 16MHz
    CSCTL2 = FLLD_0 + 487;      // set to fDCOCLKDIV = (FLLN +
1)*(fFLLREFCLK/n)
                                //                    = (487 + 1)*(32.768
kHz/1)
                                //                    = 16 MHz

    CSCTL4 |= SELA;             // set ACLK = 32kHz internal REFO
    TA0CTL |= TASSEL__ACLK | MC__CONTINOUS;      // ACLK, continuous mode, div
by 4
    TA1CTL |= TASSEL__ACLK | MC__CONTINOUS;      // ACLK, continuous mode
    //RTCCTL |= (RTCSS0|RTCSS1);                   // VLOCLK

    __delay_cycles(3);
    __bic_SR_register(SCG0);                        // enable FLL
    while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1));      // FLL locked
}

void initGPIO(void){

    //Setup pins

    //WIFI
    WIFI_CS_DIR |=  WIFI_CS;       //pin output direction
    WIFI_CS_OUT |=  WIFI_CS;       //output high

    WIFI_INT_DIR &= ~WIFI_INT;               // pin input
    WIFI_INT_OUT |= WIFI_INT;                // pin pull up
    WIFI_INT_REN |= WIFI_INT;                // pin pull up/down resistor
enable
    WIFI_INT_IES |= WIFI_INT;                // pin falling edge
    WIFI_INT_IFG &= ~WIFI_INT;               // pin IFG cleared
    WIFI_INT_IE  |= WIFI_INT;                // pin interrupt enabled

    //NFC
    P1SEL0 &= ~NFC_INT;
    NFC_INT_DIR &= ~NFC_INT;               // pin input
    NFC_INT_OUT |= NFC_INT;                // pin pull up
    NFC_INT_REN |= NFC_INT;                // pin pull up/down resistor enable
    NFC_INT_IES |= NFC_INT;               // pin falling edge
    NFC_INT_IFG &= ~NFC_INT;               // pin IFG cleared
    //NFC_INT_IE  |= NFC_INT;                // pin interrupt enabled

    P1SEL0 &= ~NFC_R;
    NFC_R_DIR |=  NFC_R;       //pin output direction
    NFC_R_OUT &=  ~NFC_R;       //output low
    __delay_cycles(1600000);
    NFC_R_OUT |=  NFC_R;       //output high
    __delay_cycles(1600000);
```

```
//    PORT_RST_SEL0 &= ~RST;                    // Setting as GPIO pin
functionality
// //   PORT_RST_SEL1 &= ~RST;
//    PORT_RST_OUT &= ~RST;
//    PORT_RST_DIR |= RST;                      // RF430CL330H device in Reset
//    __delay_cycles(100000);
//    PORT_RST_OUT |= RST;                      // Release the RF430CL330H from
Reset
//    __delay_cycles(100000);
//

    //AFE

     AFE_DIR |= (AFE_RESETZ + AFE_PDNZ);    // pin as output
     AFE_OUT |= (AFE_RESETZ + AFE_PDNZ);    // output high
     AFE_DIR &= ~(AFE_ADC_READY + AFE_PD_ALM + AFE_LED_ALM + AFE_DIAG_END);
                                            // pin input for all flags

    //SPI
     SPI_SEL = (SPI_SIMO + SPI_SOMI + SPI_SCLK);
                               // secondary pin function, SPI

     AFE_CS_DIR |= AFE_CS;       //Set the CS of the AFE to high
     AFE_CS_OUT |= AFE_CS;

     WIFI_CS_DIR |= WIFI_CS;     //Set the CS of the WIFI to high
     WIFI_CS_OUT |= WIFI_CS;

    //I2C
     I2C_SEL |= (I2C_SDA|I2C_SCL);  // secondary pin function, I2C

     //Switches
     SW_DIR &= ~(SW1|SW2);              // pin input
     SW_OUT |= (SW1|SW2);               // pin pull up
     SW_REN |= (SW1|SW2);               // pin pull up/down resistor enable
     SW_IES |= (SW1|SW2);               // pin falling edge
     SW_IFG &= ~(SW1|SW2);              // pin IFG cleared
     SW_IE  |= (SW1|SW2);              // pin interrupt enabled until later!

     // LCD
     LCD_OUT = LCD_BL;      //All bits as low except backlight, pchannel
     LCD_DIR |= 0xFF;       //All bits as output

     //Peripherals
     LED_OUT |= (RED_LED|GREEN_LED); // output high
     LED_DIR |= (RED_LED|GREEN_LED); // pin output direction

     BUZZ_OUT &= ~BUZZ; // output LOW
     BUZZ_DIR |= BUZZ;  // output DIRECTION

     VIB_OUT &= ~VIB; // output LOW
     VIB_DIR |= VIB;  // output DIRECTION

    //Unused pins should be set to output per documentation
    //"Px.0 to Px.7 Open Switched to port function, output direction (PxDIR.n
= 1)"
    P8DIR |= ~(P8_USED_PINS);
    P7DIR |= ~(P7_USED_PINS);
    P6DIR |= ~(P6_USED_PINS);
    P5DIR |= ~(P5_USED_PINS);
    P4DIR |= ~(P4_USED_PINS);
    P3DIR |= ~(P3_USED_PINS);
    P2DIR |= ~(P2_USED_PINS);
    P1DIR |= ~(P1_USED_PINS);

    //Unused pins output low
    P8OUT &= (P8_USED_PINS);
    P7OUT &= (P7_USED_PINS);
    P6OUT &= (P6_USED_PINS);
    P5OUT &= (P5_USED_PINS);
    P4OUT &= (P4_USED_PINS);
    P3OUT &= (P3_USED_PINS);
    P2OUT &= (P2_USED_PINS);
    P1OUT &= (P1_USED_PINS);


    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

}

void initComm(void){
    //initialize the serial communication registers and settings


    //USCI_A0 - SPI
    //Clock Polarity: The inactive state is high
    //MSB First, 8-bit, Master, 3-pin mode, Synchronous
    UCA0CTLW0 = UCSWRST;                        // **Put state machine in
reset**
    UCA0CTLW0 |= UCCKPL | UCCKPH | UCMSB | UCSYNC
                 | UCMST | UCSSEL__SMCLK;      // 3-pin, 8-bit SPI Slave, mode3
    UCA0BRW = 0x20;                             //clock divided by 0x20 = 32
    //UCA0MCTLW = 0;
    UCA0CTLW0 &= ~UCSWRST;                      // **Initialize USCI state
machine**
    UCA0IE |= UCRXIE;                           // Enable USCI0 RX interrupt


    //USCI_B0 - I2C
   UCB0CTLW0 |= UCSWRST;                    // Software reset enabled
   UCB0CTLW0 |= UCMODE_3 | UCMST | UCSYNC; // I2C mode, Master mode, sync
   UCB0CTLW1 |= UCASTP_2;                   // Automatic stop generated
                                            // after UCB0TBCNT is reached
   UCB0BRW = 0x00A5;                        // baudrate = SMCLK / 165
   UCB0I2CSA = 0x0028;                      // Slave address, RF430 NFC
   UCB0CTL1 &= ~UCSWRST;
   UCB0IE |= UCRXIE | UCNACKIE | UCBCNTIE;
}


void read_patient_info_WIFI(void){
    unsigned int i;
    int reached_semi=0;       //flag of semicolon reached


    int index = 1;                                 //starting position in the
1st item in array


   //Get name, up to 16 letter
   for(i = 0; i<16; i++){
       if(ReceiveBuffer[index]==59){             //if reach semicolon,
           reached_semi = 1;
           while(i<16){
               patient_name[i] = 0x00;      //fill rest of string with null
               i++;
           }
       }
       else{
           patient_name[i] = ReceiveBuffer[index];   // else move character
over
           index++;                               // update index
        }

   }
   if(!reached_semi){
       while(ReceiveBuffer[index]!=59){
           index++;     //increment index unil semicolon
       }
   }
   index++;                //skip semicolon
   reached_semi = 0;    //reset flag for semi


   //Get id, up to 8 letter
   for(i = 0; i<8; i++){
       if(ReceiveBuffer[index]==59){             //if reach semicolon,
           while(i<8){
               patient_id[i]= 0x00;         //fill rest of string with null
               i++;
           }
       }
       else{
           patient_id[i] = ReceiveBuffer[index]; //else move character over
           index++;
        }

   }
   index++;//skip semicolon
```

```
//Get room, up to 4 letter
for(i = 0; i<4; i++){
    if(ReceiveBuffer[index]==59){             //if reach semicolon,
        while(i<4){
            patient_room[i] = 0x00;      //fill rest of string with null
            i++;
        }
    }
    else{
        patient_room[i] = ReceiveBuffer[index]; //move character over
        index++;
     }

}
index++;//skip semicolon

//Get allergies, up to 28 letter
for(i = 0; i<28; i++){
    if(ReceiveBuffer[index]==59){      //if reach semicolon,
        while(i<28){
            patient_allergies[i] = 0x00;   //fill rest of string with null
            i++;
        }
    }
    else{
        patient_allergies[i] = ReceiveBuffer[index]; //move character over
        index++;
     }

}
index++;//skip semicolon

//Get more, up to 20 letter
for(i = 0; i<20; i++){
    if(ReceiveBuffer[index]==59){         //if reach semicolon,
        while(i<20){
            patient_more[i] = 0x00;  //fill rest of string with null
            i++;
        }
        index++;
    }
    else{
        patient_more[i] = ReceiveBuffer[index]; //move character over
        index++;


     }

}

LCD_Screen = 0;          //display patient info
updateLCD();
```

```
}

void read_patient_info_NFC(void){
    int i;
    int reached_semi=0;       //flag of semicolon reached

    Read_Continuous(0, nfc_read, 200);              //get contents of RF430
    //int nfc_str_len = nfc_read[30] +30 -4;        //length of data written
    int index = 35;                                 //starting position in the
35th item in array

   if((nfc_read[index]>0x39)|((nfc_read[index])<0x30)){ // if wrote Name,
       //Get name, up to 16 letter
       for(i = 0; i<16; i++){
           if(nfc_read[index]==59){             //if reach semicolon,
               reached_semi = 1;
               while(i<16){
                   patient_name[i] = 0x00;      //fill rest of string with
null
                   i++;
               }
           }
           else{
               patient_name[i] = nfc_read[index];   // else move character
over
               index++;                             // update index
            }

       }
       if(!reached_semi){
           while(nfc_read[index]!=59){
               index++;     //increment index unil semicolon
           }
       }
       index++;              //skip semicolon
       reached_semi = 0;     //reset flag for semi


       //Get id, up to 8 letter
       for(i = 0; i<8; i++){
           if(nfc_read[index]==59){             //if reach semicolon,
               while(i<8){
                   patient_id[i]= 0x00;         //fill rest of string with
null
                   i++;
               }
           }
           else{
               patient_id[i] = nfc_read[index]; //else move character over
               index++;
            }

       }
       index++;//skip semicolon
```

```
        //Get room, up to 4 letter
        for(i = 0; i<4; i++){
            if(nfc_read[index]==59){             //if reach semicolon,
                while(i<4){
                    patient_room[i] = 0x00;      //fill rest of string with
null
                    i++;
                }
            }
            else{
                patient_room[i] = nfc_read[index]; //move character over
                index++;
             }

        }
        index++;//skip semicolon

        //Get allergies, up to 28 letter
        for(i = 0; i<28; i++){
            if(nfc_read[index]==59){      //if reach semicolon,
                while(i<28){
                    patient_allergies[i] = 0x00;    //fill rest of string with
null
                    i++;
                }
            }
            else{
                patient_allergies[i] = nfc_read[index]; //move character over
                index++;
             }

        }
        index++;//skip semicolon

        //Get more, up to 20 letter
        for(i = 0; i<20; i++){
            if(nfc_read[index]==59){         //if reach semicolon,
                while(i<20){
                    patient_more[i] = 0x00;  //fill rest of string with null
                    i++;
                }
                index++;
            }
            else{
                patient_more[i] = nfc_read[index]; //move character over
                index++;
             }

        }
        LCD_Screen = 0;          //display patient info
        updateLCD();
        }
```

```
   else{             //If ID written;
   //Get id, up to 8 letter
          for(i = 0; i<8; i++){
              if((nfc_read[index]==59)|
                      (nfc_read[index]<0x30)|
                      (nfc_read[index]>0x39))
              {//if reach semicolon, or non number
                  while(i<8){
                      patient_id[i]= 0x00;         //fill rest of string with
null

                      i++;
                  }
              }
              else{
                  patient_id[i] = nfc_read[index]; //else move character over
                  index++;
               }

          }
   }

   wifi_cmd=0x02;       //get ready to read patient info from wifi



}

void wakeup_wifi(void){
    mcu_to_wifi = 1;    //Flag that shows MCU is sending cmd first

    WIFI_INT_IE  &= ~WIFI_INT;               // disable interrupts
    //pulse wifi_int to wake up WIFI chip
    WIFI_INT_OUT &= ~WIFI_INT;             // button output low
    __delay_cycles(1000);
    WIFI_INT_OUT |= WIFI_INT;               // button output high

    WIFI_INT_IFG &= ~WIFI_INT;              // button IFG cleared
    WIFI_INT_IE  |= WIFI_INT;               // enable interrupts

}

//SPI_Mode SPI_Master_WriteReg(uint8_t reg_addr, uint8_t *reg_data, uint8_t
count);
//
//SPI_Mode SPI_Master_WriteReg(uint8_t reg_addr, uint8_t *reg_data, uint8_t
count)
//{
//    MasterModeWIFI = TX_CMD_MODE;
//    TransmitCmd = reg_addr;
//
//    //Copy register data to TransmitBuffer
//    CopyArray(reg_data, TransmitBuffer, count);
//
```

```
//    TXByteCtr = count;
//    RXByteCtr = 0;
//    ReceiveIndex = 0;
//    TransmitIndex = 0;
//
//    WIFI_CS_OUT &= ~(WIFI_CS);
//    SendUCA0Data(TransmitCmd);
//
//    __bis_SR_register(CPUOFF + GIE);              // Enter LPM0 w/
interrupts
//
//    WIFI_CS_OUT |= WIFI_CS;
//    return MasterModeWIFI;
//}



SPI_Mode SPI_WIFI_CMD(uint8_t wifi_cmd, uint8_t count)
{
    MasterModeWIFI = RX_DATA_MODE;
    TransmitCmd = wifi_cmd;
    RXByteCtr = count;
    TXByteCtr = 0;
    ReceiveIndex = 0;
    TransmitIndex = 0;

    WIFI_CS_OUT &= ~(WIFI_CS);          // let wifi know we are ready to
transmit
    SendUCA0Data(TransmitCmd);              // send command
    __bis_SR_register(CPUOFF + GIE);        // Enter LPM0 w/ interrupts

    WIFI_CS_OUT |= WIFI_CS;             // pull wifi_CS back up
    return MasterModeWIFI;
}


SPI_Mode SPI_WIFI_WRITE(uint8_t* id, uint8_t count)
{
    CopyArray(id, TransmitBuffer, count);

    MasterModeWIFI = TX_DATA_MODE;
    TXByteCtr = count;
    RXByteCtr = 0;
    ReceiveIndex = 0;
    TransmitIndex = 0;

    WIFI_CS_OUT &= ~(WIFI_CS);          // let wifi know we are ready to
transmit

    SendUCA0Data(DUMMY);              // send command
    __bis_SR_register(CPUOFF + GIE);        // Enter LPM0 w/ interrupts

    WIFI_CS_OUT |= WIFI_CS;             // pull wifi_CS back up
    return MasterModeWIFI;
```

```
}


SPI_Mode SPI_WIFI_READ(uint8_t count)
{
    MasterModeWIFI = RX_DATA_MODE;
    RXByteCtr = count;
    TXByteCtr = 0;
    ReceiveIndex = 0;
    TransmitIndex = 0;

    WIFI_CS_OUT &= ~(WIFI_CS);          // let wifi know we are ready to
transmit
    //SendUCA0Data(TransmitCmd);              // send command
    UCA0IFG |= UCRXIFG;

    __bis_SR_register(CPUOFF + GIE);    // Enter LPM0 w/ interrupts

    WIFI_CS_OUT |= WIFI_CS;             // pull wifi_CS back up

    return MasterModeWIFI;
}

void SendUCA0Data(uint8_t val)
{
    while (!(UCA0IFG & UCTXIFG));              // USCI_B0 TX buffer ready?
    UCA0TXBUF = val;
    SPI_CMD_Receive=UCA0RXBUF;
//    while(!(UCA0IFG & UCRXIFG));
//    wifi_cmd_read = UCA0RXBUF;
}

void CopyArray(uint8_t *source, uint8_t *dest, uint8_t count)
{
    uint8_t copyIndex = 0;
    for (copyIndex = 0; copyIndex < count; copyIndex++)
    {
        dest[copyIndex] = source[copyIndex];
    }
}



void updateLCD(void){
    switch(LCD_Screen){
        case 0:                         //Display Patient name, id, and room
            LCDclear();
            LCDsetCursor(0,0);
            LCDprint(patient_name);
            LCDsetCursor(0,1);
            LCDprint("i:");
            LCDsetCursor(2,1);
            LCDprint(patient_id);
            LCDsetCursor(10,1);
            LCDprint("R:");
            LCDsetCursor(12,1);
            LCDprint(patient_room);
            break;

        case 1:                         //Display Patient and Clock
            LCDclear();
            LCDsetCursor(4,0);
            LCDprint(watch_time);
            LCDsetCursor(3,1);
            LCDprint(watch_date);
            break;


        case 2:
            LCDclear();
            LCDsetCursor(0,0);
            LCDprint("Hold buttons for");
            LCDsetCursor(0,1);
            LCDprint("[EMERGENCY MODE]");
            break;

        case 3:
            LCDclear();
            LCDsetCursor(0,0);
            LCDprint("Please input ID");
            LCDsetCursor(2,1);
            LCDprint("info via app");
            break;

        case -1:
            LCDclear();
            LCDsetCursor(0,0);
            LCDprint("![ EMERGENCY MODE ]!");
            LCDsetCursor(0,1);
            LCDprint("disable-hold buttons");
            break;

        default:
            LCDclear();
            LCDsetCursor(0,0);
            LCDprint("Please input patient");
            LCDsetCursor(0,1);
            LCDprint("info via NFC");
            break;
    }
}

/* +--------------------------------------------------------\\
   |                       Interrupts                        |
   \\--------------------------------------------------------+

   Priority of used interrupts:
   System Interrupts: Reset, watchdog, POR
```

```
    Timer0_A0 (TA0CCR0)
    Timer0_A1 else (TA0IV)
    Timer1_A0 (TA1CCR0)
    Timer1_A1 else (TA1IV)
    RTC
    eUSCI_A0  SPI    (UCA0IV)
    eUSCI_B0  I2c    (UCB0IV)
    P1   P1IFG.0 to .7 (P1IV)
    P2   P2IFG.0 to .7 (P2IV)

*/


#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void){
    TA0CCTL0 &= ~CCIFG;     //Clear Flag
    TA0CCTL0 &= ~CCIE;      //Disable interrupts
    int time_updated = 0;


    seconds += 1;           //increase seconds
    LED_OUT ^= GREEN_LED;   //  toggle green led

    if(seconds >= 60){
        time_updated = 1;
        seconds = 0;
        minutes++;
    }
    if (minutes==60){
            minutes=0;
            hours++;
    }
    if (hours==13){
        if(isPM){
            isPM=0;
        }
        else{
            isPM = 1;
        }
        hours = 1;
    }
    if(time_updated){
        //put into string format "HH:MMpm"

        watch_time[0] = 0x30 + (hours/10);     //get int division 10, value in
tens digit
        watch_time[1] = 0x30 + (hours%10);     //get int modulo 10, value in
ones digit
        //colon
        watch_time[3] = 0x30 + (minutes/10);     //get int division 10, value
in tens digit
        watch_time[4] = 0x30 + (minutes%10);     //get int modulo 10, value in
ones digit
        if(isPM){
            watch_time[5] =  'p';   //am or pm
        }else{
            watch_time[5] =  'a';

        }

        time_updated = 0;       //clear flag for next run
        if(LCD_Screen == 1){     //if showing time, update
            updateLCD();
        }
    }
    TA0CCTL0 &= ~CCIFG;     //Clear Flag
    TA0CCTL0 |= CCIE;      //Enable interrupts


}

#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void){

        //available for PWM

}

#pragma vector=TIMER1_A0_VECTOR
__interrupt void TIMER1_A0_ISR(void){

    if(both_sw_on){                 //if at least both switches pressed once
        //LCDclear();
        //LCDprint("Both SW");
        if((SW_IN & (SW1|SW2)) == 0){//and still pressed
               both_sw_on++;                          //increment counter
               if(both_sw_on > 25){        //if both switches pressed for 1s
constantly
                   both_sw_on = 1 ;          // flag for both switches
                   emergency_mode ^= 1;     // toggle emergency mode flag
                   TA1CCTL0 &= ~CCIFG;     //Clear Flag
                   TA1CCTL0 &= ~CCIE;      //Disable interrupts


                   __bic_SR_register_on_exit(LPM3_bits); //wake up to handle
interrupt
               }
               else{                        //if both switches pressed
consecutively
                   TA1CCR0 += 100*_1ms_32kHz;//time to check next time for
both switches
                   TA1CCTL0 &= ~CCIFG;     //Clear Flag
                   TA1CCTL0 |= CCIE;       //Enable interrupts

               }
            }
        else{                        //if not both pressed,
                both_sw_on = 0;              //reset counter
                TA1CCTL0 &= ~CCIFG;     //Clear Flag
                TA1CCTL0 &= ~CCIE;      //Disable interrupts
            }


    }

}

#pragma vector=TIMER1_A1_VECTOR
__interrupt void TIMER1_A1_ISR(void){

    if ((TA1CCTL1 & (CCIFG|CCIE)) == (CCIFG|CCIE)){ //if flag enabled and set
        //received 15ms delay for sw1
        if(SW_flag_debounce & SW1){

            //switch 1 action, only do if button is still pressed after 15ms
            if((SW_IN & SW1) == 0){


            }
            else{ //SW1 Action to do if released

            }

            //Switch Debounced, reenable interrupts
            SW_flag_debounce &= ~SW1;   //turn off flag for debounce
            P2IFG &= ~SW1;              //clear pending interrupt
            P2IE |= SW1;                //enable interrupt after 15ms

        }

        if((SW_IN & (SW1|SW2)) == 0){//and still pressed
            both_sw_on++;                       //increment counter
                TA1CCR0 += 100*_1ms_32kHz;//time to check next time for both
switches
                TA1CCTL0 &= ~CCIFG;     //Clear Flag
                TA1CCTL0 |= CCIE;       //Enable interrupts
        }
        else{                          //if not both pressed,
            both_sw_on = 0;             //reset counter
            TA1CCTL0 &= ~CCIFG;     //Clear Flag
            TA1CCTL0 &= ~CCIE;      //Disable interrupts
        }

        TA1CCTL1 &= ~CCIFG;     //Clear Flag
        TA1CCTL1 &= ~CCIE;      //Disable interrupts
    }

    if ((TA1CCTL2 & (CCIFG|CCIE)) == (CCIFG|CCIE)){   //if flag enabled and
set
        //received 15ms delay for sw2

        if(SW_flag_debounce & SW2){
            //switch 2 action, only do if button is still pressed after 25ms
            if((SW_IN & SW2) == 0){


            }
            else{ //SW2 Action to do if released

            }

            //Switch Debounced, reenable interrupts
            SW_flag_debounce &= ~SW2;    //turn off flag for debounce
            P2IFG &= ~SW2;               //clear pending interrupt
            P2IE |= SW2;                 //enable interrupt after 15ms

        }

        if((SW_IN & (SW1|SW2)) == 0){//and still pressed
            both_sw_on++;                        //increment counter
                TA1CCR0 += 100*_1ms_32kHz;//time to check next time for both
switches
                TA1CCTL0 &= ~CCIFG;      //Clear Flag
                TA1CCTL0 |= CCIE;        //Enable interrupts
        }
        else{                          //if not both pressed,
            both_sw_on = 0;              //reset counter
            TA1CCTL0 &= ~CCIFG;      //Clear Flag
            TA1CCTL0 &= ~CCIE;       //Disable interrupts
        }

        TA1CCTL2 &= ~CCIFG;      //Clear Flag
        TA1CCTL2 &= ~CCIE;       //Disable interrupts
    }

}

//rtc


//SPI
#pragma vector = USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    uint8_t UCA0_rx_val = 0;
    switch(__even_in_range(UCA0IV, USCI_SPI_UCTXIFG))
    {
        case USCI_NONE: break;
        case USCI_SPI_UCRXIFG:
            UCA0_rx_val = UCA0RXBUF;
            UCA0IFG &= ~UCRXIFG;
            switch (MasterModeWIFI)
            {
                case TX_CMD_MODE:
                    if (RXByteCtr)
                    {
                        MasterModeWIFI = RX_DATA_MODE;    // Need to start
receiving now
                        //Send Dummy To Start
                        __delay_cycles(2000000);
                        SendUCA0Data(DUMMY);
                    }
                    else
                    {
                        MasterModeWIFI = TX_DATA_MODE;        // Continue to
transmision with the data in Transmit Buffer
                        //Send First
                        SendUCA0Data(TransmitBuffer[TransmitIndex++]);
                        TXByteCtr--;
                    }
                    break;

                case TX_DATA_MODE:
                    if (TXByteCtr)
                    {
                      SendUCA0Data(TransmitBuffer[TransmitIndex++]);
                      TXByteCtr--;
                    }
                    else
                    {
                      //Done with transmission
                      MasterModeWIFI = IDLE_MODE;
                      __bic_SR_register_on_exit(CPUOFF);      // Exit LPM0
                    }
                    break;

                case RX_DATA_MODE:
                    if (RXByteCtr)
                    {
                        ReceiveBuffer[ReceiveIndex++] = UCA0_rx_val;

                        //Transmit a dummy
                        RXByteCtr--;
//                        if(ReceiveBuffer[ReceiveIndex]==0x00){
//                            RXByteCtr=0;
//                        }
                    }
                    if (RXByteCtr == 0)
                    {
                        MasterModeWIFI = IDLE_MODE;
                        __bic_SR_register_on_exit(CPUOFF);      // Exit LPM0
                    }
                    else
                    {
                        SendUCA0Data(DUMMY);
                    }
                    break;

                default:
                    __no_operation();
                    break;
            }
            __delay_cycles(1000);
            break;
        case USCI_SPI_UCTXIFG:
            break;
        default: break;
    }
}

#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void){


    if(P1IFG & WIFI_INT){       //interrupt from wifi

        P1IE &=  ~WIFI_INT;     //disable interrupt from wifi
        P1IFG &= ~WIFI_INT;     //Clear flag from wifi

        if(mcu_to_wifi){        // if mcu sent command

        }
        else{                   //if wifi sent command

        }

        wifi_int = 1;

        __bic_SR_register_on_exit(LPM3_bits); //wake up to handle interrupt
    }
    else if(P1IFG & NFC_INT){   //interrupt from nfc

        P1IE &= ~NFC_INT;       //disable intetrrupt from NFC
        P1IFG &= ~NFC_INT;      //Clear flag from NFC

        LED_OUT ^= RED_LED;     //toggle red led

        nfc_int = 1;            //set NFC flag
        __bic_SR_register_on_exit(LPM3_bits); //wake up to handle interrupt

    }


}

#pragma vector=PORT2_VECTOR
__interrupt void PORT2_ISR(void){

    //switches
    if(P2IFG & SW1){      //switch 1

        //software debounce with timer, turn on IE after
        SW_flag_debounce |= SW1; //flag for switch interrupt, enabled later
        P2IFG &= ~SW1;  //clear interrupt until later
        P2IE &= ~SW1;   //disable sw1 interrupt for 15 ms

        //SW1 Action
        //next LCD Screen
        if(LCD_Screen != 0){
            LCD_Screen--; //move to prev lcd screen display
        }
        else{LCD_Screen = HighestLCDScreen;} //Start at last screen
        updateLCD();

        TA1CCR1 = TA0R + 15*_1ms_32kHz;   //delay 15ms
        TA1CCTL1 &= ~CCIFG; //Clear pending flag
        TA1CCTL1 |= CCIE;   //enable flag interrupt

    }
    if(P2IFG & SW2){      //Ready to read HR

       //software debounce with timer, turn on IE after

        SW_flag_debounce |= SW2;     //flag for switch interrupt, enabled
later
        P2IFG &= ~SW2;  //clear interrupt until later
        P2IE &= ~SW2;   //disable SW2 interrupt for 15 ms

        //next LCD Screen
        if(LCD_Screen != HighestLCDScreen){
            LCD_Screen++; //move to next lcd screen display
        }
        else{LCD_Screen = 0;} //Start at first screen
        updateLCD();

        TA1CCR2 = TA0R + 15*_1ms_32kHz;   //delay 15ms
        TA1CCTL2 &= ~CCIFG; //Clear pending flag
        TA1CCTL2 |= CCIE;   //enable flag interrupt
    }

    //AFE INTERRUPTS
    if(P2IFG & AFE_ADC_READY){      //Ready to read HR


        P2IFG &= ~AFE_ADC_READY;
    }
    if(P2IFG & AFE_DIAG_END){       //Diagnostic completion

        P2IFG &= ~AFE_DIAG_END;
    }
    if(P2IFG & AFE_DIAG_END){       //interrupt from nfc

        P2IFG &= ~AFE_ADC_READY;
    }
    if(P2IFG & AFE_DIAG_END){       //interrupt from nfc
```

```
        P2IFG &= ~AFE_ADC_READY;
    }
    if(P2IFG & AFE_DIAG_END){       //interrupt from nfc

        P2IFG &= ~AFE_ADC_READY;
    }
    if(P2IFG & AFE_DIAG_END){       //interrupt from nfc

        P2IFG &= ~AFE_ADC_READY;
    }
    if(P2IFG & AFE_DIAG_END){       //interrupt from nfc

        P2IFG &= ~AFE_ADC_READY;
    }


}
```

11 master.h

```
/*
 * main.h
 *
 * Defines values for pins, ports, registers and commands used in LifeWatch
v2.0
 *
 *  Created on: Nov 19, 2018
 *      Author: William Toledo
 */

#ifndef MASTER_H_
#define MASTER_H_

//Headers
#include <msp430.h>
#include <RF430.h>
//#include <AFE4400.h>
#include <LCDMSP.h>
#include <stdint.h>





//function declarations
void initClocks();
void initGPIO();
void initComm();
void read_patient_info_NFC();
void updateLCD(void);
void wakeup_wifi(void);
void CopyArray(uint8_t *source, uint8_t *dest, uint8_t count);
void SendUCA0Data(uint8_t val);
//SPI_Mode SPI_Master_WriteReg(uint8_t reg_addr, uint8_t *reg_data, uint8_t
count);
//SPI_Mode SPI_WIFI_CMD(uint8_t wifi_cmd, uint8_t count);
//SPI_Mode SPI_WIFI_READ(uint8_t count);



// Pin Definitions by sections

//WIFI
#define WIFI_CS         BIT0
#define WIFI_CS_OUT     P8OUT
#define WIFI_CS_DIR     P8DIR

#define WIFI_INT        BIT7
#define WIFI_INT_IN     P1IN
#define WIFI_INT_DIR    P1DIR
#define WIFI_INT_OUT    P1OUT
#define WIFI_INT_REN    P1REN
#define WIFI_INT_IES    P1IES
#define WIFI_INT_IFG    P1IFG
#define WIFI_INT_IE     P1IE

//NFC
#define NFC_R           BIT5
#define NFC_R_OUT       P1OUT
#define NFC_R_DIR       P1DIR
#define NFC_R_SEL       P1SEL0

#define NFC_INT         BIT6
#define NFC_INT_OUT     P1OUT
#define NFC_INT_DIR     P1DIR
#define NFC_INT_IN      P1IN
#define NFC_INT_REN     P1REN
#define NFC_INT_IES     P1IES
#define NFC_INT_IFG     P1IFG
#define NFC_INT_IE      P1IE
#define NFC_INT_SEL     P1SEL0

//AFE
#define AFE_OUT     P2OUT
#define AFE_DIR     P2DIR
#define AFE_IN      P2IN
#define AFE_REN     P2REN
#define AFE_IES     P2IES
#define AFE_IFG     P2IFG
#define AFE_IE      P2IE
```

```
#define AFE_PDNZ        BIT2
#define AFE_DIAG_END    BIT3
#define AFE_LED_ALM     BIT4
#define AFE_PD_ALM      BIT5
#define AFE_ADC_READY   BIT6
#define AFE_RESETZ      BIT7

#define AFE_CS         BIT3
#define AFE_CS_OUT     P1OUT
#define AFE_CS_DIR     P1DIR

//LCD
#define LCD_OUT     P7OUT
#define LCD_DIR     P7DIR

#define LCD_DB0     BIT0
#define LCD_DB1     BIT1
#define LCD_DB2     BIT2
#define LCD_DB3     BIT3
#define LCD_E       BIT4
#define LCD_RS      BIT5
#define LCD_BL      BIT6 //P CHANNEL MOSFET
#define LCD_VM      BIT7


//Peripherals
#define LED_OUT     P3OUT
#define LED_DIR     P3DIR
#define RED_LED     BIT0
#define GREEN_LED   BIT1

#define BUZZ_OUT     P3OUT
#define BUZZ_DIR     P3DIR
#define BUZZ    BIT2

#define VIB_OUT     P3OUT
#define VIB_DIR     P3DIR
#define VIB    BIT3       //P CHANNEL MOSFET

//Switches
#define SW_OUT     P2OUT
#define SW_DIR     P2DIR
#define SW_IN      P2IN
#define SW_REN     P2REN
#define SW_IES     P2IES
#define SW_IFG     P2IFG
#define SW_IE      P2IE

#define SW1        BIT0
#define SW2        BIT1
```

```
//Communication

//SPI
#define SPI_SEL     P1SEL0
#define SPI_SIMO    BIT0
#define SPI_SOMI    BIT1
#define SPI_SCLK    BIT2

//I2C
#define I2C_SEL     P5SEL0
#define I2C_OUT     P5OUT
#define I2C_DIR     P5DIR
#define I2C_SDA     BIT2
#define I2C_SCL     BIT3


//Used pins
#define P8_USED_PINS    WIFI_CS
#define P7_USED_PINS    LCD_DB0|LCD_DB1 |LCD_DB2|LCD_DB3|LCD_E|LCD_RS|LCD_BL
|LCD_VM
#define P6_USED_PINS    0
#define P5_USED_PINS    I2C_SDA|I2C_SCL
#define P4_USED_PINS    0
#define P3_USED_PINS    GREEN_LED|RED_LED|VIB|BUZZ
#define P2_USED_PINS
SW1|SW2|AFE_PDNZ|AFE_DIAG_END|AFE_LED_ALM|AFE_PD_ALM|AFE_ADC_READY|AFE_RESETZ
#define P1_USED_PINS
WIFI_INT|NFC_R|NFC_INT|AFE_CS|SPI_SIMO|SPI_SOMI|SPI_SCLK




//LCD Screen
//typedef enum LCD_ScreenEnum{
//    NAME_ROOM,
//    TIME_DATE,
//    ID_NAME
//} LCD_Screen_Mode;
//
//LCD_Screen_Mode LCD_Screen = NAME_ROOM;

// variables used in functions
#define _1ms_16MHz    16000    //cycles to have 1ms at 16MHz
#define _1ms_8MHz     8000     //cycles to have 1ms at 8MHz
#define _1ms_1MHz     1000     //cycles to have 1ms at 1MHz
#define _1ms_32kHz    32       //cycles to have 1ms at 32kHz
#define _1ms_1kHz     1        //cycles to have 1ms at 1kHz


////wifi commands
//#define WIFI_TIME       1
//#define WIFI_PATIENT    2
//
#define DUMMY   0x41
```

```
#define MAX_BUFFER_SIZE     100



#endif /* MASTER_H_ */
```

12 RF430.c

```
/*
 * {RF430_example.c}
 *
 * {Functions}
 *
 * Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/
 *
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions
 *  are met:
 *
 *    Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *    Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the
 *    distribution.
 *
 *    Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
*/
/*
 * RF430_example.c
 *
 *  Created on: Feb 6, 2013
 *  Author: JD Crutchfield
 */
#include <master.h>
#include <RF430.h>
#include "msp430.h"

unsigned char RxData[2] = {0,0};
unsigned char TxData[2] = {0,0};
unsigned char TxAddr[2] = {0,0};


unsigned char NDEF_Application_Data[] = RF430_DEFAULT_DATA;

void initNFC(void){
    __delay_cycles(4000000);                // Leave time for the RF430CL33H
to get itself initialized; should be 20ms or greater

      while(!(Read_Register(STATUS_REG) & READY)); // Wait until READY bit has
been set

      //write NDEF memory with Capability Container + NDEF message
      Write_Continuous(0, NDEF_Application_Data, 48);

      //Enable interrupts for End of Read and End of Write
      Write_Register(INT_ENABLE_REG, EOW_INT_ENABLE + EOR_INT_ENABLE);

      //Configure INTO pin for active low and enable RF
      Write_Register(CONTROL_REG, INT_ENABLE + INTO_DRIVE + RF_ENABLE);



      __delay_cycles(4000000);                // Leave time for the RF430CL33H
to get itself initialized; should be 20ms or greater

      NFC_INT_IFG &= ~NFC_INT;                // pin IFG cleared
      NFC_INT_IE  |= NFC_INT;                 // pin interrupt enabled

}

unsigned int Read_Register(unsigned int reg_addr)
{
TxAddr[0] = reg_addr >> 8;      //MSB of address
    TxAddr[1] = reg_addr & 0xFF;    //LSB of address

    UCB0CTL1  &= ~UCSWRST;
    UCB0CTL1 |= UCTXSTT + UCTR;     //start i2c write operation
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[1];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0CTL1 &= ~UCTR;              //i2c read operation
    UCB0CTL1 |= UCTXSTT;            //repeated start
    while(!(UCB0IFG & UCRXIFG0));
    RxData[0] = UCB0RXBUF;
    UCB0CTL1 |= UCTXSTP;            //send stop after next RX
    while(!(UCB0IFG & UCRXIFG0));
    RxData[1] = UCB0RXBUF;
    while((UCB0STAT & UCBBUSY));    // Ensure stop condition got sent
    UCB0CTL1  |= UCSWRST;

    return RxData[1] << 8 | RxData[0];
}

//reads the register at reg_addr, returns the result
unsigned int Read_Register_BIP8(unsigned int reg_addr)
{
    unsigned char BIP8 = 0;
    TxAddr[0] = reg_addr >> 8;      //MSB of address
    TxAddr[1] = reg_addr & 0xFF;    //LSB of address

    UCB0CTL1  &= ~UCSWRST;
    UCB0CTL1 |= UCTXSTT + UCTR;     //start i2c write operation

    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[0];
    BIP8 ^= TxAddr[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[1];
    BIP8 ^= TxAddr[1];

    while(!(UCB0IFG & UCTXIFG0));     // Waiting for TX to finish on bus
    UCB0CTL1 &= ~UCTR;              //i2c read operation
    UCB0CTL1 |= UCTXSTT;            //repeated start

    while(!(UCB0IFG & UCRXIFG0));
    RxData[0] = UCB0RXBUF;
    BIP8 ^= RxData[0];
    while(!(UCB0IFG & UCRXIFG0));
    RxData[1] = UCB0RXBUF;
    BIP8 ^= RxData[1];

    UCB0CTL1 |= UCTXSTP;            //send stop after next RX
    while(!(UCB0IFG & UCRXIFG0));
    if(BIP8 != UCB0RXBUF){
        __no_operation();
    }

    while((UCB0STAT & UCBBUSY));     // Ensure stop condition got sent
    UCB0CTL1  |= UCSWRST;

    return RxData[0] << 8 | RxData[1];
}

void Read_Continuous(unsigned int reg_addr, unsigned char* read_data, unsigned
int data_length)
{
    unsigned int i;
    TxAddr[0] = reg_addr >> 8;       //MSB of address
    TxAddr[1] = reg_addr & 0xFF;     //LSB of address

    UCB0CTL1  &= ~UCSWRST;
    UCB0CTL1 |= UCTXSTT + UCTR;      //start i2c write operation.  Sending
Slave address

    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[1];
    while(!(UCB0IFG & UCTXIFG0));      // Waiting for TX to finish on bus
    UCB0CTL1 &= ~UCTR;               //i2c read operation
    UCB0CTL1 |= UCTXSTT;             //repeated start
    while(!(UCB0IFG & UCRXIFG0));

    for(i = 0; i < data_length-1; i++)
    {
        while(!(UCB0IFG & UCRXIFG0));
        read_data[i] = UCB0RXBUF;
    }

    UCB0CTL1 |= UCTXSTP;             //send stop after next RX
    while(!(UCB0IFG & UCRXIFG0));
    read_data[i] = UCB0RXBUF;
    while((UCB0STAT & UCBBUSY));     // Ensure stop condition got sent
    UCB0CTL1  |= UCSWRST;
}

//writes the register at reg_addr with value
void Write_Register(unsigned int reg_addr, unsigned int value)
{
    TxAddr[0] = reg_addr >> 8;       //MSB of address
    TxAddr[1] = reg_addr & 0xFF;     //LSB of address
    TxData[0] = value >> 8;
    TxData[1] = value & 0xFF;

    UCB0CTL1  &= ~UCSWRST;
    UCB0CTL1 |= UCTXSTT + UCTR;      //start i2c write operation
    //write the address
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[1];
    //write the data
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxData[1];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxData[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0CTL1 |= UCTXSTP;
    while((UCB0STAT & UCBBUSY));      // Ensure stop condition got sent
    UCB0CTL1  |= UCSWRST;
```

```
}

//writes the register at reg_addr with value
void Write_Register_BIP8(unsigned int reg_addr, unsigned int value)
{
    unsigned char BIP8 = 0;

    TxAddr[0] = reg_addr >> 8;      //MSB of address
    TxAddr[1] = reg_addr & 0xFF;    //LSB of address
    TxData[0] = value >> 8;
    TxData[1] = value & 0xFF;

    UCB0CTL1  &= ~UCSWRST;
    UCB0CTL1 |= UCTXSTT + UCTR;     //start i2c write operation

    //write the address
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[0];
    BIP8 ^= TxAddr[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[1];
    BIP8 ^= TxAddr[1];

    //write the data
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxData[0];
    BIP8 ^= TxData[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxData[1];
    BIP8 ^= TxData[1];

    //send BIP8 byte
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = BIP8;

    while(!(UCB0IFG & UCTXIFG0));
    UCB0CTL1 |= UCTXSTP;
    while((UCB0STAT & UCBBUSY));;   // Ensure stop condition got sent
    UCB0CTL1  |= UCSWRST;

}

//0, NDEF_Application_Data, 48

//writes the register at reg_addr and incrementing addresses with the data at
"write_data" of length data_length
void Write_Continuous(unsigned int reg_addr, unsigned char* write_data,
unsigned int data_length)
{
    unsigned int i;

    TxAddr[0] = reg_addr >> 8;      //MSB of address
    TxAddr[1] = reg_addr & 0xFF;    //LSB of address
    UCB0CTL1  &= ~UCSWRST;
    UCB0CTL1 |= UCTXSTT + UCTR;     //start i2c write operation
    //write the address
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[0];
    while(!(UCB0IFG & UCTXIFG0));
    UCB0TXBUF = TxAddr[1];

    for(i = 0; i < data_length; i++)
    {
        while(!(UCB0IFG & UCTXIFG0));
        UCB0TXBUF = write_data[i];
    }

    while(!(UCB0IFG & UCTXIFG0));
    UCB0CTL1 |= UCTXSTP;
    while((UCB0STAT & UCBBUSY));    // Ensure stop condition got sent
    UCB0CTL1  |= UCSWRST;

}
```

13 RF430.h

```
/*
 * {RF430_example.h}
 *
 * {RF430 header}
 *
 * Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/
 *
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions
 *  are met:
 *
 *    Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *    Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the
 *    distribution.
 *
 *    Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
*/
/*
 * RF430_example.h
 *
 *  Created on: Feb 6, 2013
 *      Author: JD Crutchfield
 */

#ifndef RF430_H_
#define RF430_H_

//#include "RF430_example.c"

void initNFC(void);
unsigned int Read_Register(unsigned int reg_addr);
unsigned int Read_Register_BIP8(unsigned int reg_addr);
void Read_Continuous(unsigned int reg_addr, unsigned char* read_data, unsigned
int data_length);

void Write_Register(unsigned int reg_addr, unsigned int value);
void Write_Continuous(unsigned int reg_addr, unsigned char* write_data,
unsigned int data_length);
void Write_Register_BIP8(unsigned int reg_addr, unsigned int value);



//define the values for Granite's registers we want to access
#define CONTROL_REG         0xFFFE
#define STATUS_REG          0xFFFC
#define INT_ENABLE_REG      0xFFFA
#define INT_FLAG_REG        0xFFF8
#define CRC_RESULT_REG      0xFFF6
#define CRC_LENGTH_REG      0xFFF4
#define CRC_START_ADDR_REG  0xFFF2
#define COMM_WD_CTRL_REG    0xFFF0
#define VERSION_REG         0xFFEE //contains the software version of the ROM
#define TEST_FUNCTION_REG   0xFFE2
#define TEST_MODE_REG       0xFFE0

//define the different virtual register bits
//CONTROL_REG bits
#define SW_RESET        BIT0
#define RF_ENABLE       BIT1
#define INT_ENABLE      BIT2
#define INTO_HIGH       BIT3
```

```
#define INTO_DRIVE      BIT4
#define BIP8_ENABLE     BIT5
#define STANDBY_ENABLE  BIT6
#define TEST430_ENABLE  BIT7
//STATUS_REG bits
#define READY           BIT0
#define CRC_ACTIVE      BIT1
#define RF_BUSY         BIT2
//INT_ENABLE_REG bits
#define EOR_INT_ENABLE      BIT1
#define EOW_INT_ENABLE      BIT2
#define CRC_INT_ENABLE      BIT3
#define BIP8_ERROR_INT_ENABLE       BIT4
#define NDEF_ERROR_INT_ENABLE   BIT5
#define GENERIC_ERROR_INT_ENABLE    BIT7
//INT_FLAG_REG bits
#define EOR_INT_FLAG    BIT1
#define EOW_INT_FLAG    BIT2
#define CRC_INT_FLAG    BIT3
#define BIP8_ERROR_INT_FLAG BIT4
#define NDEF_ERROR_INT_FLAG BIT5
#define GENERIC_ERROR_INT_FLAG  BIT7
//COMM_WD_CTRL_REG bits
#define WD_ENABLE   BIT0
#define TIMEOUT_PERIOD_2_SEC    0
#define TIMEOUT_PERIOD_32_SEC   BIT1
#define TIMEOUT_PERIOD_8_5_MIN  BIT2
#define TIMEOUT_PERIOD_MASK     BIT1 + BIT2 + BIT3

#define TEST_MODE_KEY 0x004E

#define RF430_DEFAULT_DATA      {
\
/*NDEF Tag Application Name*/
\
0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01,
\

\
/*Capability Container ID*/
\
0xE1, 0x03,
\
0x00, 0x0F, /* CCLEN */
\
0x20,       /* Mapping version 2.0 */
\
0x00, 0xF9, /* MLe (49 bytes); Maximum R-APDU data size */
\
0x00, 0xF6, /* MLc (52 bytes); Maximum C-APDU data size */
\
0x04,       /* Tag, File Control TLV (4 = NDEF file) */
\
0x06,       /* Length, File Control TLV (6 = 6 bytes of data for this tag) */
\
0xE1, 0x04, /* File Identifier */
\
0x0B, 0xDF, /* Max NDEF size (3037 bytes of useable memory) */
\
0x00,       /* NDEF file read access condition, read access without any
security */     \
0x00,       /* NDEF file write access condition; write access without any
security */   \

\
/* NDEF File ID */
\
0xE1, 0x04,
\

\
/* NDEF File for Hello World */
\
0x00, 0x14, /* NLEN: NDEF length (20 byte long message, max. length for
RF430CL) */     \

\
/* NDEF Record (refer to NFC Data Exchange Format specifications)*/
\
0xD1,       /*MB(Message Begin), SR(Short Record) flags set, ME(Message End),
IL(ID length field present) flags cleared; TNF(3bits) = 1; */ \
0x01, 0x10, /*Type Length = 0x01; Payload Length = 0x10 */
\
0x54,       /* Type = T (text) */
\
0x02,       /* 1st payload byte: "Start of Text", as specified in ASCII Tables
*/       \
0x65, 0x6E, /* 'e', 'n', (2nd, 3rd payload bytes*/
\

\
/* 'Hello, world!' NDEF data*/
\
0x48, 0x65, 0x6C, 0x6C, 0x6f, 0x2c, 0x20, 0x77, 0x6f, 0x72, 0x6c, 0x64, 0x21
\
    } /* End of data */

#endif /* RF430_H_ */
```

14 LCDMSP.c

```
//#include "LCDMSP.h"
//#include "msp430.h"
#include <stdio.h>
#include <string.h>
```

```
#include <inttypes.h>
#include <master.h>
//#include "WProgram.h"

#define cyc_per_millisec 16000    //if 1mHz clk, 1 cyc = 1us, 1ms=1000cyc
int m =0;


void LCDinit()
{
    _displayfunction = LCD_4BITMODE | LCD_2LINE | LCD_5x8DOTS;
    _numlines = 2;
    _currline = 0;

    //initialize LCD Pins
    LCD_OUT &= ~(LCD_BL|LCD_E|LCD_RS|LCD_DB2|LCD_DB3|LCD_DB0|LCD_DB1);

    //delay after power up
    LCD_OUT |= (LCD_RS|LCD_E);
    for(m=0; m<50;m++){
        __delay_cycles(cyc_per_millisec); // wait 50 ms
    }

   LCD_OUT &= ~(LCD_RS|LCD_E);


    //LCD takes times to initialize
   LCDwrite4bits(0x03);
   for(m=0; m<5;m++){
           __delay_cycles(cyc_per_millisec); // wait 5 ms
       }

    LCDwrite4bits(0x03);
    for(m=0; m<7;m++){
            __delay_cycles(cyc_per_millisec); // wait 7 ms
    }

    //enter 4bit mode
    LCDwrite4bits(0x02);

    //LCD setup
    LCDcmd(LCD_FUNCTIONSET |_displayfunction);

    // turn the display on with no cursor or blinking default
    _displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
     LCDdisplay();

    //LCD Clear
    LCDclear();

    // Initialize to default text direction (for romance languages)
    _displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
    // set the entry mode
    LCDcmd(LCD_ENTRYMODESET | _displaymode);
```

```
}


//void LCDupdate(uint8_t value)
//{       //update the screen to given value in screen
//    LCDclear();
//    LCDhome();
//    switch(value){
//        case 0: //Patient Info
//            LCDprint(patient_Name);
//    }
//
//
//}

/********** high level commands, for the user! */
void LCDclear()
{
  LCDcmd(LCD_CLEARDISPLAY);  // clear display, set cursor position to zero
  __delay_cycles(2*cyc_per_millisec);
}

void LCDhome()
{
  LCDcmd(LCD_RETURNHOME);  // set cursor position to zero
  __delay_cycles(2*cyc_per_millisec);
}

void LCDsetCursor(uint8_t col, uint8_t row){
  int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
  if ( row > _numlines ) {
    row = _numlines-1;    // we count rows starting w/0
  }

  LCDcmd(LCD_SETDDRAMADDR | (col + row_offsets[row]));
}

// Turn the display on/off (quickly)
void LCDnoDisplay() {
   _displaycontrol &= ~LCD_DISPLAYON;
   LCDcmd(LCD_DISPLAYCONTROL | _displaycontrol);
}
void LCDdisplay() {
  _displaycontrol |= LCD_DISPLAYON;
  LCDcmd(LCD_DISPLAYCONTROL | _displaycontrol);
}

// Turns the underline cursor on/off
void LCDnoCursor() {
  _displaycontrol &= ~LCD_CURSORON;
  LCDcmd(LCD_DISPLAYCONTROL | _displaycontrol);
}
void LCDcursor() {
  _displaycontrol |= LCD_CURSORON;
  LCDcmd(LCD_DISPLAYCONTROL | _displaycontrol);
}

// Turn on and off the blinking cursor
void LCDnoBlink() {
  _displaycontrol &= ~LCD_BLINKON;
  LCDcmd(LCD_DISPLAYCONTROL | _displaycontrol);
}
void LCDblink() {
  _displaycontrol |= LCD_BLINKON;
  LCDcmd(LCD_DISPLAYCONTROL | _displaycontrol);
}

// These commands scroll the display without changing the RAM
void LCDscrollDisplayLeft(void) {
  LCDcmd(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
}
void LCDscrollDisplayRight(void) {
  LCDcmd(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
}

// This is for text that flows Left to Right
void LCDleftToRight(void) {
  _displaymode |= LCD_ENTRYLEFT;
  LCDcmd(LCD_ENTRYMODESET | _displaymode);
}

// This is for text that flows Right to Left
void LCDrightToLeft(void) {
  _displaymode &= ~LCD_ENTRYLEFT;
  LCDcmd(LCD_ENTRYMODESET | _displaymode);
}

// This will 'right justify' text from the cursor
void LCDautoscroll(void) {
  _displaymode |= LCD_ENTRYSHIFTINCREMENT;
  LCDcmd(LCD_ENTRYMODESET | _displaymode);
}

// This will 'left justify' text from the cursor
void LCDnoAutoscroll(void) {
  _displaymode &= ~LCD_ENTRYSHIFTINCREMENT;
  LCDcmd(LCD_ENTRYMODESET | _displaymode);
}

// Allows us to fill the first 8 CGRAM locations
// with custom characters
void LCDcreateChar(uint8_t location, uint8_t charmap[]) {
  location &= 0x7; // we only have 8 locations 0-7
  LCDcmd(LCD_SETCGRAMADDR | (location << 3));
  int i;
  for (i=0; i<8; i++) {
    LCDwrite(charmap[i]);
  }
```

```
}

/*********** mid level commands, for sending data/cmds */

void LCDprint(uint8_t string[]){
    int i =0;
    char letter;
    int length = 16;
      //for(i=0; i<length; i++) {
          letter = string[i];
          while(letter != '\0'){
              LCDwrite(letter);
              i++;
              letter = string[i];
              if(i>length){
                  break;
              }
          }
}

inline void LCDcmd(uint8_t value) {
  LCDsend(value, 0);
}

inline void LCDwrite(uint8_t value) {
  LCDsend(value, 1);
}

/************ low level data pushing commands **********/

// write either command or data, with automatic 4/8-bit selection
void LCDsend(uint8_t value, uint8_t mode) {
    if(mode){//writing
       LCD_OUT |= LCD_RS;
    } else
    {
       LCD_OUT &= ~LCD_RS;
    }

      LCDwrite4bits(value>>4);
      //__delay_cycles(31*cyc_per_millisec);
      LCDwrite4bits(value);
    }

void LCDpulseEnable(void) {

    LCD_OUT &= ~LCD_E;
    __delay_cycles(100);
    LCD_OUT |= LCD_E;
    __delay_cycles(100);
    LCD_OUT &= ~LCD_E;
    __delay_cycles(10000);
   }
```

```
void LCDwrite4bits(uint8_t value) {

  value = (0x0F & value);   //take lower nibble
  LCD_OUT &= ~0x0F;        //xxxx 0000
  LCD_OUT |= value;        //xxxx 1234

  LCDpulseEnable();
}
```

15 LCDMSP.h

```
#ifndef LCDMSP_H_
#define LCDMSP_H_

#include <inttypes.h>
//#include "Print.h"

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
```

```
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00



  void LCDinit();
  void LCDclear();
  void LCDhome();
  void LCDupdate(uint8_t value);

  void LCDnoDisplay();
  void LCDdisplay();
  void LCDnoBlink();
  void LCDblink();
  void LCDnoCursor();
  void LCDcursor();
  void LCDscrollDisplayLeft();
  void LCDscrollDisplayRight();
  void LCDleftToRight();
  void LCDrightToLeft();
  void LCDautoscroll();
  void LCDnoAutoscroll();

  void LCDcreateChar(uint8_t, uint8_t[]);
  void LCDsetCursor(uint8_t, uint8_t);
  void LCDwrite(uint8_t);
  void LCDcmd(uint8_t);

  void LCDsend(uint8_t, uint8_t);
  void LCDwrite4bits(uint8_t);
  void LCDpulseEnable();
  void LCDprint(uint8_t string[]);

  uint8_t _displayfunction;
  uint8_t _displaycontrol;
  uint8_t _displaymode;

  uint8_t _initialized;

  uint8_t _numlines,_currline;


#endif
```

Appendix C – Code for WIFI MCU

The following is the code for the WIFI chip; it includes SPI, WIFI connectivity, and HTTP requests.

16 WIFI.c

```
#include <Arduino.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"
#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"
#include "lwip/igmp.h"
#include "esp_wifi.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_event_loop.h"
#include "nvs_flash.h"
#include "soc/rtc_cntl_reg.h"
#include "rom/cache.h"
#include "driver/spi_slave.h"
#include "esp_log.h"
#include "esp_spi_flash.h"
#include "String.h"


#define GPIO_HANDSHAKE 4
#define GPIO_MOSI 23
#define GPIO_MISO 19
#define GPIO_SCLK 18
#define GPIO_CS 5

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiMulti.h>
#include <HTTPClient.h>

const char* ssid = "Life watch";
const char* password =  "";

//Declare Subfunctions
void WIFI_conn();
char* http_get(char* url, int length1);
void spi_transaction( int t_length, char* tx_data);
void spi_init();
char* get_time();
char* get_info();
```

```
//Called after a transaction is queued and ready for pickup by master. We use
this to set the handshake line high.
void my_post_setup_cb(spi_slave_transaction_t *trans) {
//    WRITE_PERI_REG(GPIO_OUT_W1TS_REG, (0<<GPIO_HANDSHAKE));
Serial.println("This is right before pulling WIFI interrupt low");
 pinMode(GPIO_NUM_4,OUTPUT);
    digitalWrite(GPIO_NUM_4, 0);              //Fire WIFI int
//      digitalWrite(GPIO_NUM_4, 1);
// pinMode(GPIO_NUM_4,INPUT);
}

//Called after transaction is sent/received. We use this to set the handshake
line low.
void my_post_trans_cb(spi_slave_transaction_t *trans) {
 //   WRITE_PERI_REG(GPIO_OUT_W1TC_REG, (1<<GPIO_HANDSHAKE));

Serial.println("This is right after transaction is done");
   digitalWrite(GPIO_NUM_4, 1);
   delay(10);
 pinMode(GPIO_NUM_4,INPUT);
}

spi_slave_transaction_t t;                       //Declare a transaction instant
esp_err_t ret;                                   //Initialize return
char* sendbuf;                                  //Initialize transmit buffer
char recvbuf[10]={0,0,0,0,0,0,0,0,0,0};
//Initialize receive buffer
int test;                                        //Test variable for interrupt
pin
char* time_result;
char* dummy = "J";
char* id_fake="Hello";

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  delay(10);
  pinMode(GPIO_NUM_4,INPUT);
  //digitalWrite(GPIO_NUM_4, 1);
  delay(500);
  spi_init();

/*//Configuration for the SPI slave interface
    spi_slave_interface_config_t slvcfg;
        slvcfg.mode=3;                          //SPI Mode (0-3)
        slvcfg.spics_io_num=GPIO_CS;            //Chip Select Pin assigned
        slvcfg.queue_size=10;                    //Size of Queue
        slvcfg.flags=0;                         //Bitwise or of SPI_SLAVE_*
flags
       slvcfg.post_setup_cb=my_post_setup_cb; //Function executed after new
data is loaded
        slvcfg.post_trans_cb=my_post_trans_cb;//Routine executed after
transaction is done
```

```
//Configuration for the SPI bus
    spi_bus_config_t buscfg;
        buscfg.mosi_io_num=GPIO_MOSI;         //MOSI pin selected
        buscfg.miso_io_num=GPIO_MISO;         //MISO pin selected
        buscfg.sclk_io_num=GPIO_SCLK;         //SCLK pin selected

//SPI slave interface initialization
ret=spi_slave_initialize(VSPI_HOST, &buscfg, &slvcfg, 0); //Initialize SPI
interface
                                                          //Interface, pins,
configuration,
                                                          //DMA channel: used
for large
                                                          //number of bytes
   assert(ret==ESP_OK);                                   //Is initialization
okay?*/
   Serial.println("This is the return code");
   Serial.println(ret);                                     //Print error code
   //memset(recvbuf, 0, 33);                             //Initiazlie receive
buffer to 0
   memset(&t, 0, sizeof(t));                              //Initialize all
traits of the
                                                          //Transaction to 0
    t.rx_buffer=recvbuf;                                  //Set address for
receive buffer
                                                          //We would like to
never change this address
}

void loop() {
  int test;
//Disable WIFI in order to enter low power mode

//Enable interrupts and go to sleep
esp_sleep_enable_ext0_wakeup(GPIO_NUM_4,0);             //Set GPIO wake up
test = digitalRead(GPIO_NUM_4);
Serial.println("This is the value of the interrupt");
Serial.println(test);
esp_light_sleep_start();                                  //Go to sleep
Serial.println("This is the first thing after waking up");


//Enable WIFI & then connect
 WIFI_conn();

  Serial.println("Wakeup reason was external GPIO");
//First get instruction code
  spi_transaction( 1, dummy);
//Now check instruction code whether 0 or 1: 0 = get time; 1 = get patient
info
      if(recvbuf[0]==0x01){
   Serial.println("The receive buffer value was equal to 0x01");
   delay(500);
        //Declare a char array to get result
        char* time_result;
//Try time result array vs pointer to character
        time_result = get_time();
        Serial.println("This is the time result after jumping back to the main
loop");
        Serial.println(time_result);
        Serial.println(*time_result);

        //Set up spi transaction to send time
        spi_transaction(strlen(time_result)+1, time_result);
      }
      else if (recvbuf[0]==0x02){
   Serial.println("The receive buffer value was equal to 0x02");

   //Now I need to receive ID from MCU
   spi_transaction(6, id_fake);

   //Now ID is in the recvbuf: add it to the end of the request
   //Declare a char array to get result
        char* full_result;
        char chunk_result1[31];
        char chunk_result2[31];
        full_result = get_info();
        int length1 = strlen(full_result);
        for(int i =0; i<30;i++){
          chunk_result1[i] = full_result[i];
        }
        //Then set up SPI to transmit info
        spi_transaction(strlen(chunk_result1), chunk_result1);

        //Send remaining bytes
        for(int i=30; i<60; i++){
          chunk_result2[i-30] = full_result[i];
        }
        Serial.println("This is chunk_result2");
        Serial.println(chunk_result2);
        //spi_transaction(strlen(chunk_result2), chunk_result2);
      }
      else{
        Serial.println("None of the codes were met");
        Serial.println(recvbuf[0]);
      }
      Serial.println("Disable WIFI");
      esp_wifi_stop();
      delay(500);
      Serial.println("Repeat code and go back to sleep");
}


//---------------------------Sub Functions-----------------------------------
-
//------------------WIFI Connect--------------------
void WIFI_conn() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }
  Serial.println("Connected to the WiFi network");
}

//------------------SPI Transaction------------------------------------
void spi_transaction( int t_length, char* tx_data){
 t.length=t_length*8 ;                                        //Length of
transmission
 t.tx_buffer=tx_data;                                         //Write to the
actual buffer
 Serial.println("This is before queing");
 Serial.println("This is the send buffer value");
 Serial.println(tx_data);
 Serial.println("This is the receive buffer value");
 Serial.println(recvbuf);
 spi_slave_transmit(VSPI_HOST, &t, portMAX_DELAY);
 Serial.println("This is the return code");
 Serial.println(ret);
 Serial.println("This is the value of the receive buffer after executing");
  Serial.println(*recvbuf, HEX);
 for(int i=0; i<strlen(recvbuf)+1;i++){
  recvbuf[i]=recvbuf[i]>>1;
 }
 Serial.println(*recvbuf, HEX);


}
//-------------------SPI Initialize-----------------------------------
void spi_init(){
  //Configuration for the SPI slave interface
    spi_slave_interface_config_t slvcfg;
        slvcfg.mode=3;                          //SPI Mode (0-3)
        slvcfg.spics_io_num=GPIO_CS;            //Chip Select Pin assigned
        slvcfg.queue_size=12;                    //Size of Queue
        slvcfg.flags=0;                         //Bitwise or of SPI_SLAVE_*
flags
       slvcfg.post_setup_cb=my_post_setup_cb; //Function executed after new
data is loaded
        slvcfg.post_trans_cb=my_post_trans_cb;//Routine executed after
transaction is done

//Configuration for the SPI bus
    spi_bus_config_t buscfg;
        buscfg.mosi_io_num=GPIO_MOSI;           //MOSI pin selected
        buscfg.miso_io_num=GPIO_MISO;           //MISO pin selected
        buscfg.sclk_io_num=GPIO_SCLK;           //SCLK pin selected

//SPI slave interface initialization
ret=spi_slave_initialize(VSPI_HOST, &buscfg, &slvcfg, 0); //Initialize SPI
interface
```

```
                                                          //Interface, pins,
configuration,
   assert(ret==ESP_OK);                                   //Is initialization
okay?
}

//---------------------HTTP Request for time & time conversion----------------
---------
char* get_time(){
  String payload;
  static char result[258];
  static char edited_result[20]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  int length1;
  if ((WiFi.status() == WL_CONNECTED)) { //Check the current connection status
    HTTPClient http;
    http.begin("http://worldclockapi.com/api/json/est/now"); //Specify the URL
    int httpCode = http.GET();
    if (httpCode > 0) { //Check for the returning code
        payload = http.getString();
        Serial.println(httpCode);
        Serial.println(payload);
        length1 = payload.length() +1;
        payload.toCharArray(result,length1);
        Serial.println("This is the original string");
        Serial.println(result);
        Serial.println("This is what we will return");
        for(int i =0; i<16;i++){
          edited_result[i] = result[30+i];
        }
        Serial.println(edited_result);
        Serial.println("This is converting from 24 to 12");
        //Add pm or am to the end of the string
        if((edited_result[11]==0x31 && edited_result[12]>0x31) ||
edited_result[11]==2){
        edited_result[16] = 'p';
        edited_result[17] = 'm' ;
        }
        else{
        edited_result[16] = 'a';
        edited_result[17] = 'm';
        }
        //Convert numbers from 13-19
         if(edited_result[11]==0x31 && edited_result[12]>0x32){
          edited_result[11] = '0';
          edited_result[12] = edited_result[12]-0x02;
          Serial.println(edited_result);
        }
        //Convert numbers from 20-21
         else if(edited_result[11]==0x32 && edited_result[12]<0x32){
          edited_result[11] = '0';
          edited_result[12] = edited_result[1]+0x08;
          Serial.println(edited_result);
        }
        //Convert numbers from 22-24
        else if(edited_result[11]==0x32 && edited_result[12]>0x31){
          edited_result[11] = '1';
          edited_result[12] = edited_result[1]-0x02;
          Serial.println(edited_result);
        }
        else{
          Serial.println("Time is already in standard 12 hour mode");
        }
      }

    else {
      Serial.println("Error on HTTP request");
    }
    http.end(); //Free the resources
  }
  Serial.println(edited_result);
  return edited_result;
}

//----------------------HTTP Request for patient info--------------------
char* get_info(){
 String payload;
 static char result[15];
 char url[]="http://192.168.43.52/patient_id/product/read_one.php?id=";
 char* temp;
 int j,k,i;
 int length1;

//First add ID to the end of the url
//char* temp = recvbuf;

length1 = strlen(url)+1;
Serial.println(length1);
char edited_url[length1+5];

for(i=0;i<strlen(url);i++){
  edited_url[i] = url[i];
  Serial.println(edited_url[i]);
}
j =strlen(url);

for(i=0; i<5;i++){
  edited_url[j+i] = recvbuf[1+i];
  Serial.println(edited_url[j+i]);
}
 edited_url[j+5] = 0x00;
Serial.println(edited_url);
Serial.println(strlen(edited_url));

  if ((WiFi.status() == WL_CONNECTED)) { //Check the current connection status
    HTTPClient http;
    http.begin(edited_url); //Specify the URL
    int httpCode = http.GET();                                         //Make
the request
    if (httpCode > 0) { //Check for the returning code
        payload = http.getString();
        Serial.println(httpCode);
        Serial.println(payload);
        length1 = payload.length() +1;
        payload.toCharArray(result,length1);
        Serial.println("This is what we will return");
        Serial.println(result);
        delay(500);
      }

    else {
      Serial.println("Error on HTTP request");
    }
    delay(500);
    http.end(); //Free the resources
  }
  return result;
}
```