EE 459 Final Report

Team 6 - Samsung

Alexander Lee, Juliana Echternach, Ebrahim Faisal, Qihong (Jeffrey) Wang

I. Purpose

The purpose of this project was to create a "smart" outdoor recreational device. We worked on a interdisciplinary team with a marketing team that provided market research on what products would sell best and how to sell whatever "smart" product we developed, and a design team that created a 3D rendering of the product. Based on market research provided by our business team members, we decided to create a smart hiking pole under the Samsung brand.

II. Introduction

The idea behind this product was to allow the average hiker to have a safer, more convenient experience while hiking. It would work as an attachment to hiking poles and have a screen on the top of the pole that would display all the features. We were tasked with developing a device that could navigate between many features through a touchscreen. The features included a GPS, Heart Rate Monitor, a Compass, and Emergency Signal. A major theme in this product was safety. For example, we wanted the user to send an emergency signal that did not rely on Wifi or Satellite connection, and for the hiking pole to monitor the users' heart rate, and send an emergency message if the users' pulse reached a dangerous rate.

III. Overview

The *Smart Hiking Pole* is a device we created where the user navigates through the device features through a touchscreen. All of the device's features are displayed on a home page. Every module is accessed by the microcontroller through different means of communications.

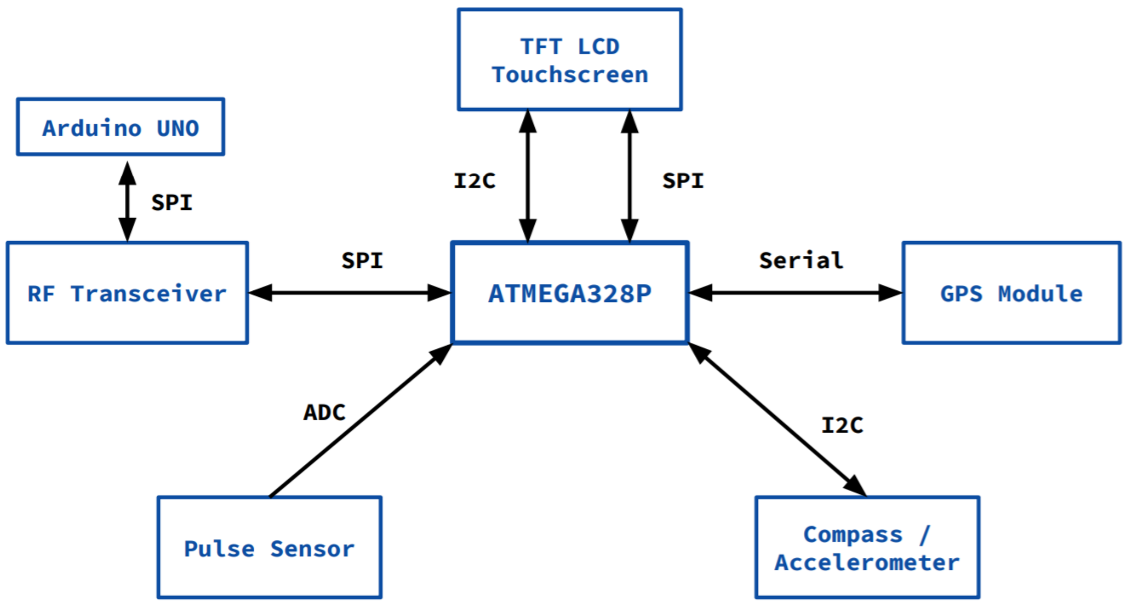

Figure 1. *Design Block Diagram. Every feature in our design is driven by the Atmega328P Microcontroller. The connecting arrows illustrate the communication method used to communicate between the chip and the devices.*

Our project utilized several types of communication, as shown in Figure 1. The GPS received satellite information through serial communication, the LCD displayed information through Serial Peripheral Interface (SPI), and used Integer-Integrated

Circuit (I2C) Protocol to handle the user touch. The Pulse Sensor used Analog to Digital Converter (ADC) pins on the Atmega328P. Finally, the Accelerometer required I2C communication. From *Figure 1* it is apparent we had a lot of different parts with varying communication to juggle for this project.

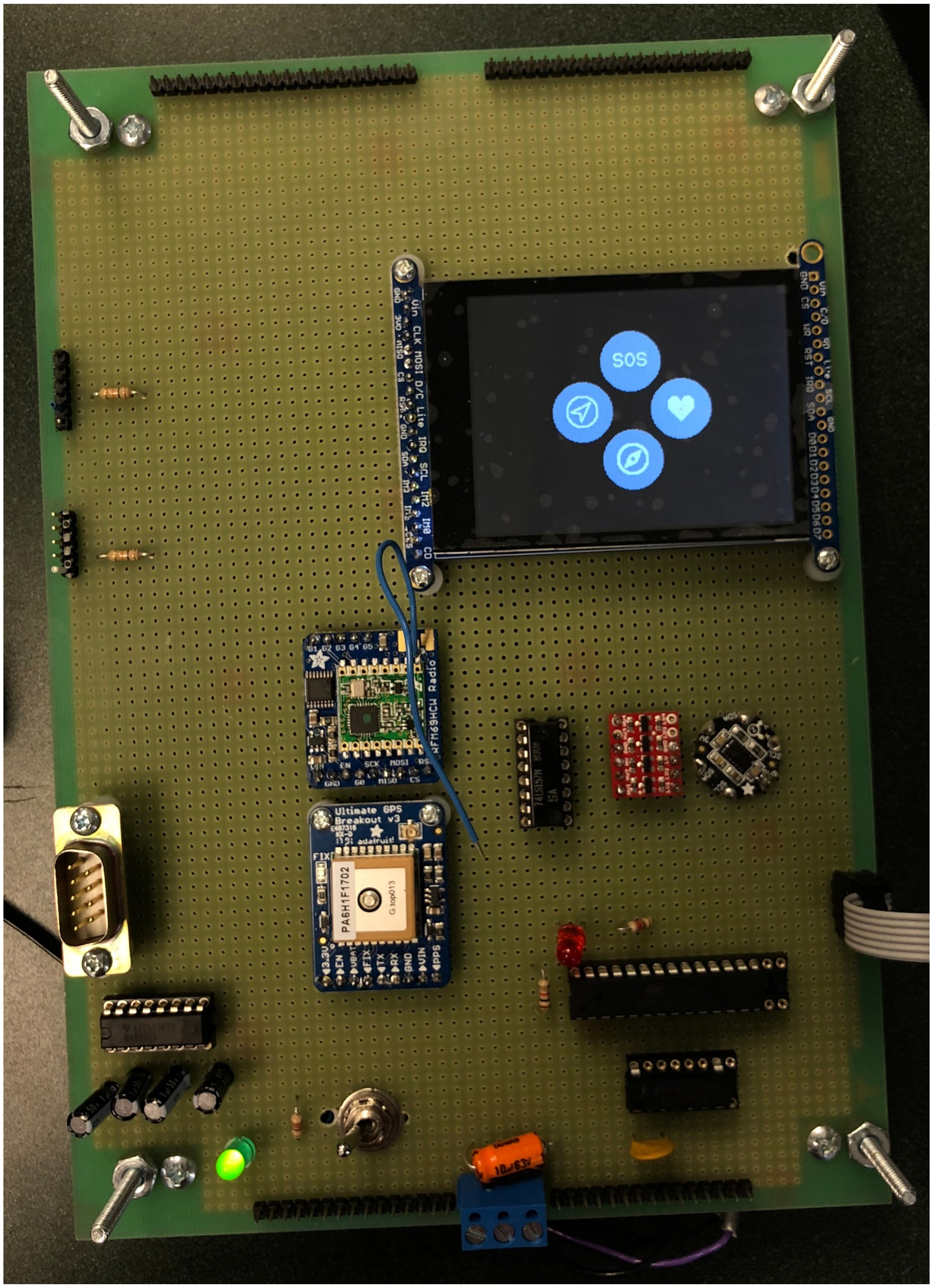

Figure 2. *Finalized Engineering Prototype (Top)*

Inputs	Outputs
User Touch	LCD Display
GPS Information	RF Transmit Signal
Pulse Sensor	Compass Position
Motion / Direction (Accelerometer)	
RF Receive Signal	

Table 1. *The inputs and outputs used in the prototype.*

Part	Cost
GPS	$39.95
TFT LCD Touchscreen	$39.95
Accelerometer / Compass	$14.95
RF Transceiver (x2)	$19.90 (per $9.95)
Pulse Sensor	$24.95
ATmega328p	$2.14
Total:	**$141.84**

Table 2. *The cost analysis of the parts purchased for the Smart Hiking Pole device.*

IV. Project Components

A. Touchscreen LCD

The Touchscreen LCD was the *2.8" TFT LCD with Capacitive Touch* from Adafruit. It comprised of two components, the LCD ILI9341 interface (which communicated in either an 8-bit or SPI mode) and the FT6206 Touch interface (which communicated in I2C).

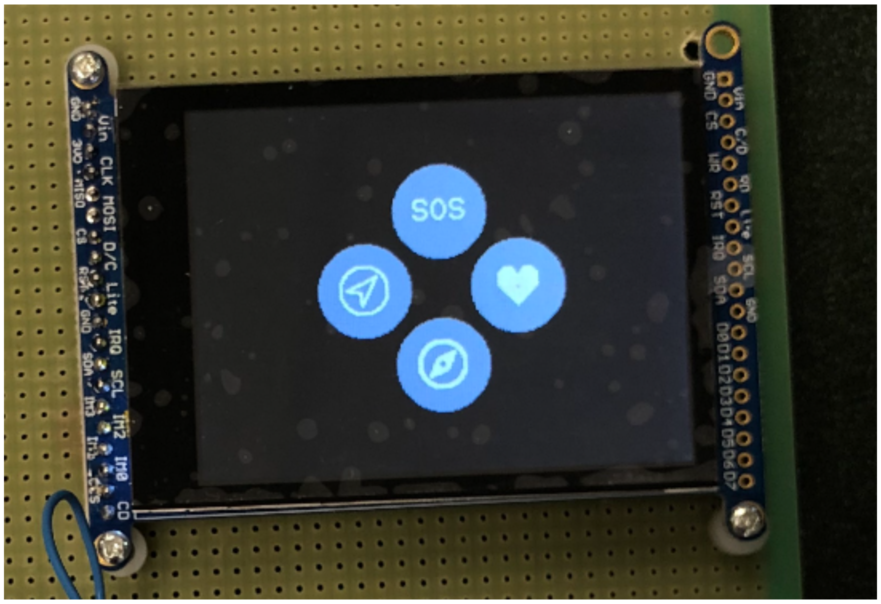

Figure 3. *Home Screen GUI showcasing the TFT LCD*

Communication to the LCD interface was conducted using SPI over 8-Bit. This was because of the limited number of ports on the Atmega328p and latency was not a huge concern (8-Bit mode is known to be faster). Additionally, communication was conducted by 2 modes, Data (D) or Command (C) using the D/C pin. In order to write information onto a register in the interface, we would first send a 8-bit register address with the D/C pin set to command mode, and then send a 8/32-bit data value with the D/C pin set to the data mode. Overall, writing to these registers allows us to output pixels on the display.

For a pixel to be drawn, the address window registers and then send a color to the memory write register. From left to right, in rows, we are able to fill in pixels within the address window. Using this function to draw pixels, we were able to draw lines, circles, text, and icons. Lines and circles were drawn using the Bresenham algorithms. Text and icons were draw using bit manipulation from bitmaps. Bitmaps for the text and icons were stored in header files as arrays of integers, where each bit in an integer represented whether or not a pixel was to be drawn in. From there, we developed a simple algorithm to go through each integer and use bit-shifting operators in order to determine which pixel needed to be written to. Using these drawing algorithms, we were able to set up the GUI shown in *Figure X*.

Touch was communicated via the I2C protocol. Since all of the memory on the FT6206 interface were sequentially next to each other (in terms of addressing), instead of manually addressing each register for information, we were able to address 16 bytes at once from the FT6206 I2C address. Implementing the data retrieved, we created a *point_t* struct in C that allowed us to store the x, y, and pressure of each point. Points were stored globally in an array, and were retrieved by a getter function in the firmware. Using the location of each point, we were able to detect whether a point was in the bounding box of a button.

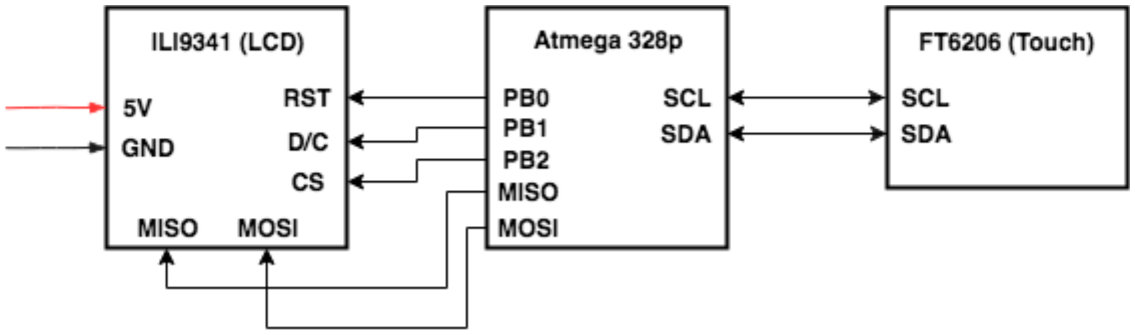

Figure 4. *TFT LCD Block Diagram*

B. GPS

The GPS Module we used was the *Ultimate GPS Breakout v3* from Adafruit. It received its satellite information as a National Marine Electronics Association (NMEA) sentence. NMEA sentences output varying interpreted sentences based on their $GPXXX tags. For this project, we were interested in the $GPGGA formatted sentence, which was the Global Positioning System Fix Data. Our job was to parse the string and display that on the LCD.

One important signal that was necessary in parsing the $GPGGA string was the Fix Quality. If the GPS could not get a fix, then it was not communicating with a satellite properly, and the Fix Quality would output 0. Parsing the data required for there to be a GPS Fix of 1 or 2. It was difficult to get a fix inside the lab, so we utilized an Antenna to connect to the uFL antenna connection on the module. Since inside the lab it took awhile to acquire a fix, we decided to put a limit on the number of tries that the GPS could look for a data fix, so that the GPS was not stalling the device for too long looking for data. We had attempted to use interrupt to collect the data but it conflicted with the LCD, so we decided this was the best way to handle acquiring information. Time was given in UTC time based on position, so we converted it to Pacific Standard Time. The module also had a 3.3V output that supplied 100mA of current, which was used to connect to other devices on the board that required 3.3V instead of 5.

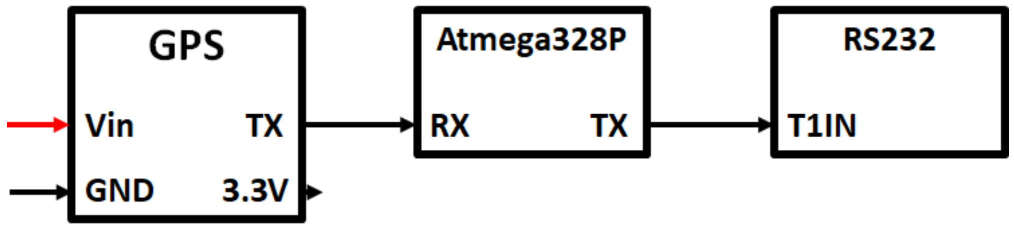

Figure 5. *Diagram of how the GPS was wired on the project board. The GPS Receives satellite information and outputs this through the (TX) pin which goes to the microcontroller RX pin. This is connected to the RS232 T1IN pin.*

Debugging the parsing of the GPS string was mainly done through the Serial connection on the computer. We used PuTTY to output the information and check that the data parsed made sense. Once this was working the GPS data was outputted on the LCD Display.

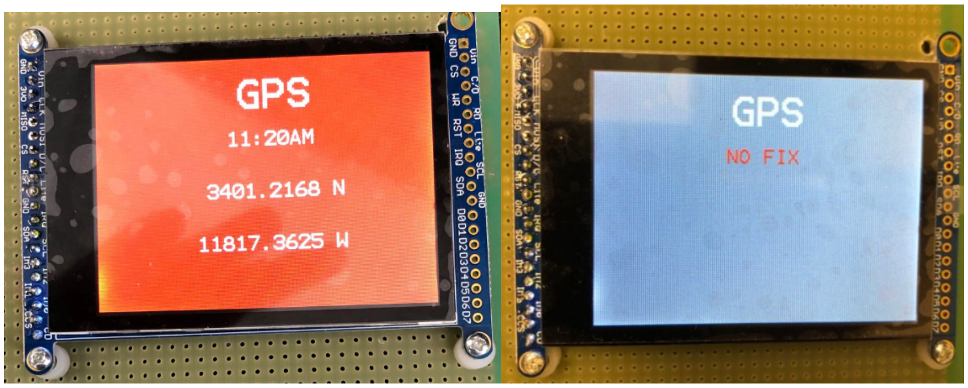

Figure 6. *Image of the GPS output on the LCD Display. The left image shows the GPS working with time, latitude, and longitude displayed on the screen. The right image shows the output if the GPS was unable to get a fix.*

C. Pulse Sensor

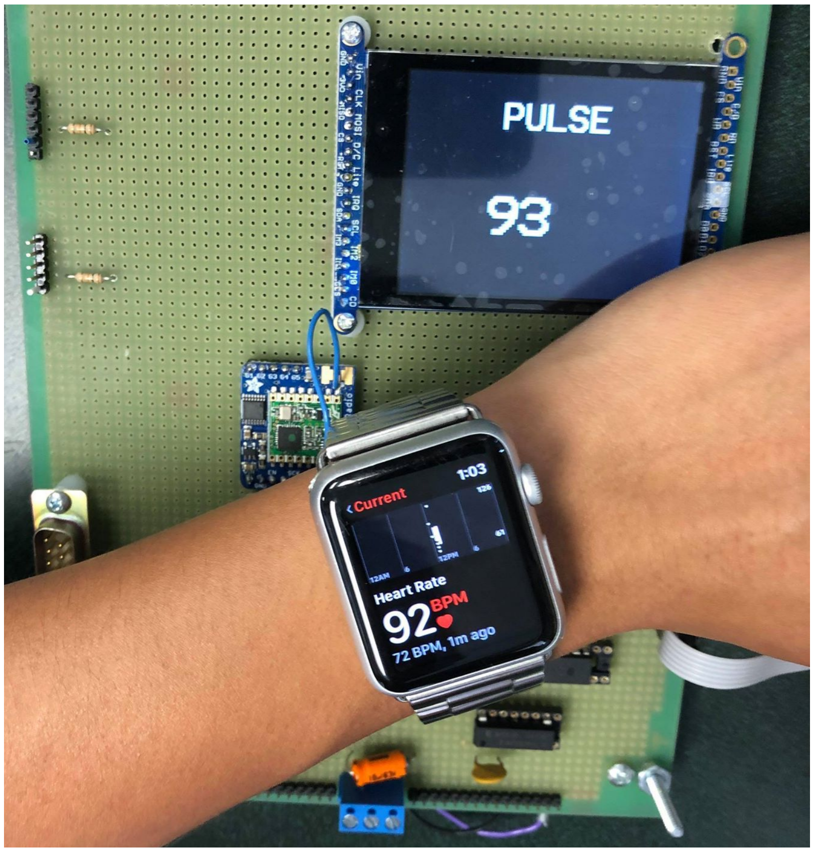

Figure 7. *Pulse Sensor compared to Apple Watch Pulse Reading*

The pulse sensor module used in the project is sold on PulseSensor.com. The module is designed to interface with an Arduino with a set of libraries implemented using the Arduino IDE. However, the module was interfaced with the Atmega328p microcontroller and the available libraries were used for reference only. The pulse sensor has three individual pins: a ground pin was connected to the ground bus on the board; a power pin was connected to the power source bus on the board rated at 5V;

and an analog signal pin connected to the ADC pin2 on the microcontroller with a range of 0V - 5V.

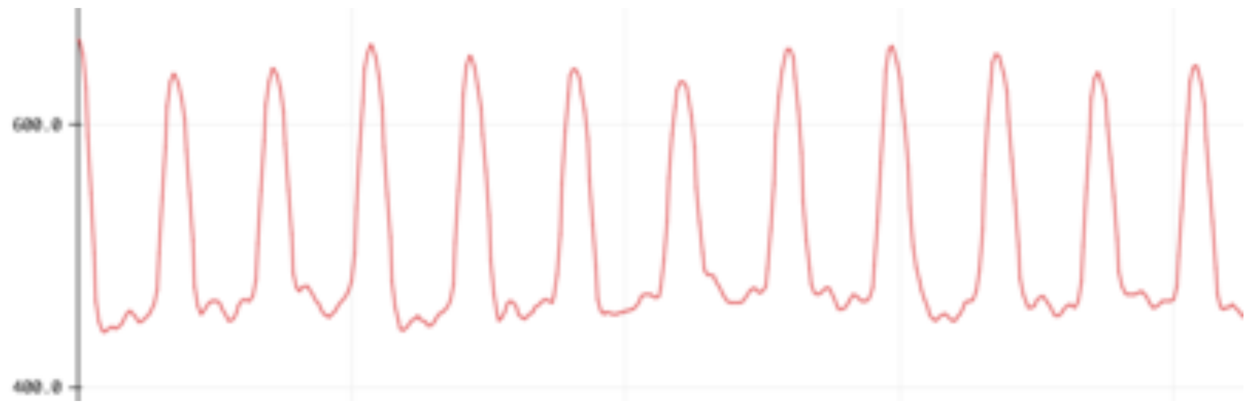

Figure 8. *A graph replicating the pulse sensor's output measured using the oscilloscope. The wave signal ranges between 0V to 5V and is centered at 2.5V.*

The figure above shows the pulse sensor output. When the sensor is not properly placed, the signal stabilizes at 5V then drops to 0V to indicate a reset. Afterwards, the signal stabilizes around 2.5V with negligible noise until a heartbeat is introduced to the sensor. Once a heartbeat is detected, the displayed wave appears with an amplitude proportional to the strength of light reflecting back from targeted arteries.

In order to process the sensor's input, a code must be setup to convert analog signal to digital and capture data periodically. ADCSRA register was set to enable Analog-to-Digital conversion. A separate function was implemented to initiate conversion when the pulse sensor signal needs capturing. ADMUX was assigned which pin the pulse sensor is connected to and ADC conversion is carried out using polling. A 10-bit digital value is then returned given the sensor is designed to provide values ranging between 0 and 1024.

Given the Atmega328p chip has limited memory, variables used to process the sensor's signal are declared as volatile global within the .c file and were updated each time the input signal is processed.

Variable Name	**Size**	**Purpose**
BPM (Beats Per Minute)	2 Bytes	Stores the final BPM value calculated using the latest signal read by ADC pin
IBI (Interval of Beat Irregularity)	2 Bytes	A changing value representing the wait time necessary to consider a pulse in the signal to be heartbeat. It is used to detect the drop in voltage following a pulse. It makes sure the signal is shaped like a heartbeat pulse and not noise
Pulse	1 Byte	A flag set when a pulse in signal is detected. It is used to start detecting a drop in voltage following a pulse
sampleCounter	4 Bytes	A counter that tracks the time between individual pulses
lastBeatTime	4 Bytes	A timestamp of the last measured heartbeat. Used with sample counter to determine the time interval between pulses to calculate BPM
P	2 Bytes	Stores the local maximum value of a peak
T	2 Bytes	Store the local minimum value of a trough

thresh	2 Bytes	Threshold value to determine whether the signal is high or low. It is pivotal in detecting pulses in signal and must be updated according to the middle value the pulse sensor stabilize around. The variable is initially set to the middle value of ADC conversion. Overtime, the local amplitude of the signal is calculated using P and T. The middle value of the amplitude is then stored in thresh.
amp	2 Bytes	Stores the local amplitude of the signal. The wave signal may change its center and range. Updating amp is necessary to update the threshold value used to detect pulses.
firstBeat	1 Byte	A flag to track the first beat. This flag is used to prevent updating BPM when the first beat of the signal is detected given it will result in producing an inaccurate reading
secondBeat	1 Byte	Second beat is used to start calculating BPM after the first beat
rate[10]	20 Bytes	An array storing beat interval time of the past 10 pulses detected. The content of the array is used to calculate the average BPM.

Table 3. *A table of all the variables used in processing the pulse sensor signal.*

When the user selects the heartbeat monitor on the display, the interface initiates a protocol to start reading the pulse sensor signal. The protocol runs for approximately 5-10 seconds to collect enough samples for accurate reading. Once the protocol is finished, the BPM global variable is passed to the display interface. When the BPM value calculated is out of the normal range, the phrase "Bad Data" is displayed, prompting the user to initiate the pulse monitor again.

D. Compass

For the compass, we used the 3-axis Adafruit *LSM303* accelerometer and magnetometer. The accelerometer gave us our relative rotation on all three euler rotational axes, while the magnetometer gave us relative strength of the magnetic field (which helped us determine where true north was).

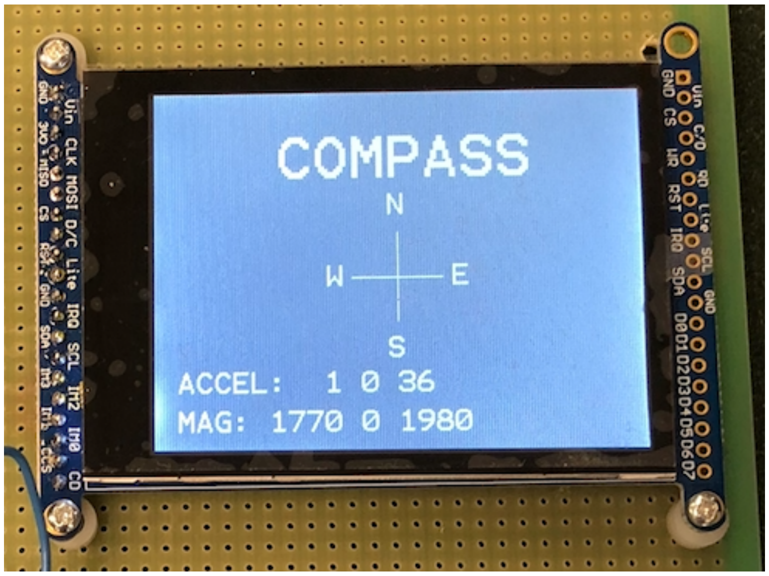

Figure 9. *Compass Interface.*

Due to inaccuracies of the device and timing, we were unable to calculate the angles of true north; however, shown in *Figure X*, we were able to display the values obtained from the LSM303 interface. Raw acceleration data was in terms of *g*'s (or multiples of 9.81), and the raw magnetic information was in terms of *gauss*.

Communication was done by I2C (similar to the Touch Interface). Using information from both sensors, we attempted to calculate the angle (ignoring the z-axis) by taking the *arctan* of the y and x values (*arctan(y,x)* to be precise) and adding the acceleration angle to the magnetometer angle in order to determine relative rotation from true north. We added the acceleration angle to the magnetometer angle since the acceleration angle was the additional offset from the true north value from the magnetometer.

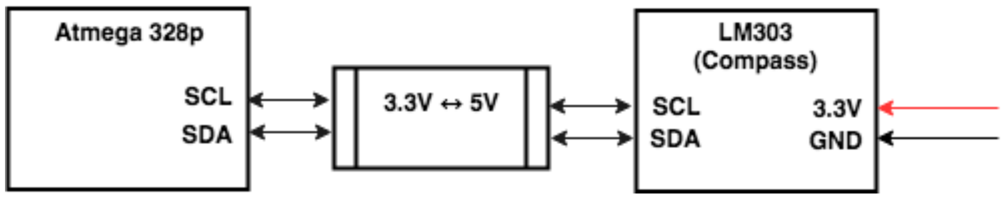

Figure 10. *Compass Block Diagram (LM303 should be LSM303)*

E. RF Transceiver

For the RF Transceiver we used the RFM69 Radio by Adafruit. For this aspect of our project we were trying to send an "Emergency Signal" to another hiking pole device. Because of this we used two radio modules and attempted to get them to communicate with one another. To do this we had one RFM69 on our main board and a second RFM69 connected to an Arduino UNO. Unfortunately, this was an aspect of the project that we were not able to finish. We were able to get the radio on the Arduino to send and receive signals, and the radio on the board to send signals, but could not get the two to communicate with one another.

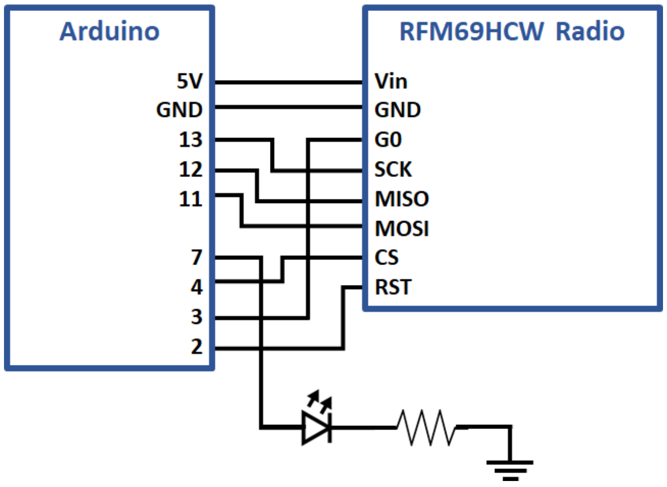

Figure 11. *Wiring diagram of the seconds RFM69HCW Radio connected to the Arduino UNO. We included an LED circuit to blink to indicate when information was sent and received.*

The radio uses SPI communication. With this type of communication there requires a Master and Slave device. On the second board we set up the Arduino as the Master and the radio as the Slave, and on the main board the Microcontroller was the Master and the radio was the slave. Since the LCD also used SPI we navigated through these devices using the Chip Select. We were not able to get the radio and the LCD to work in conjunction, since the Radio used interrupts and disturbed the function of the display.

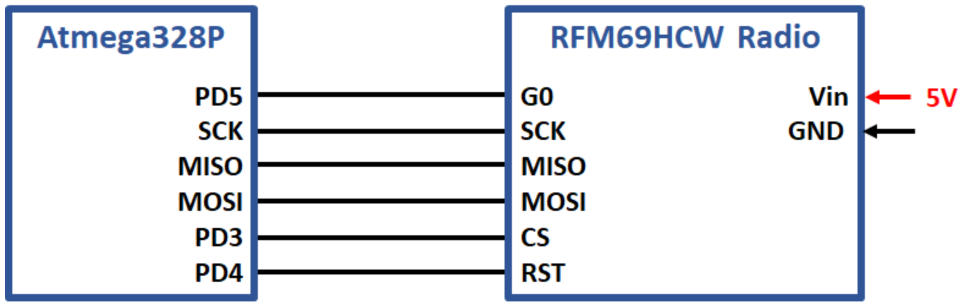

Figure 12. *Wiring diagram of the transmitting radio on the main board connected to the Microcontroller.*

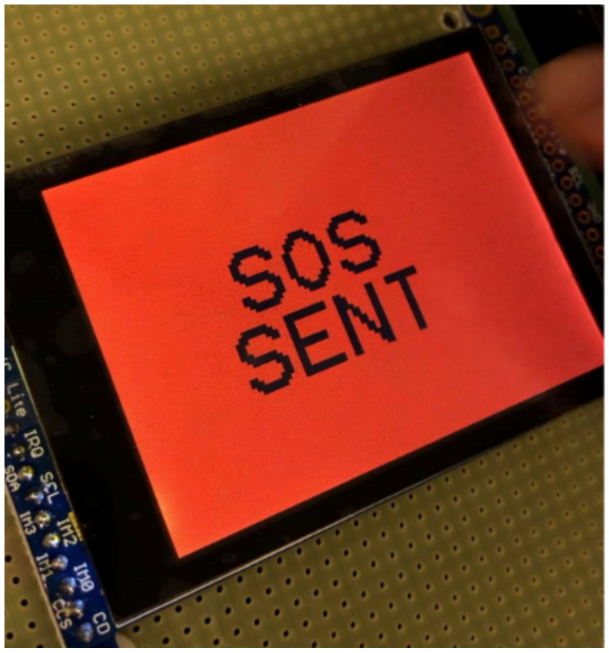

Figure 13. *Outputted message when the user touches the SOS icon on the LCD.*

V. Implementation and Design

A. State Diagram

As illustrated in the figure below, a user can navigate through the different features by tapping the icon on the screen. Tapping on this calls the features specific function. As stated earlier, we decided not to use interrupts since it would disable the LCD functionality for reasons we could not fix. The way we worked around this was to evoke each device only when they are called on by the user. We found this worked well and allowed us to test the device easily.

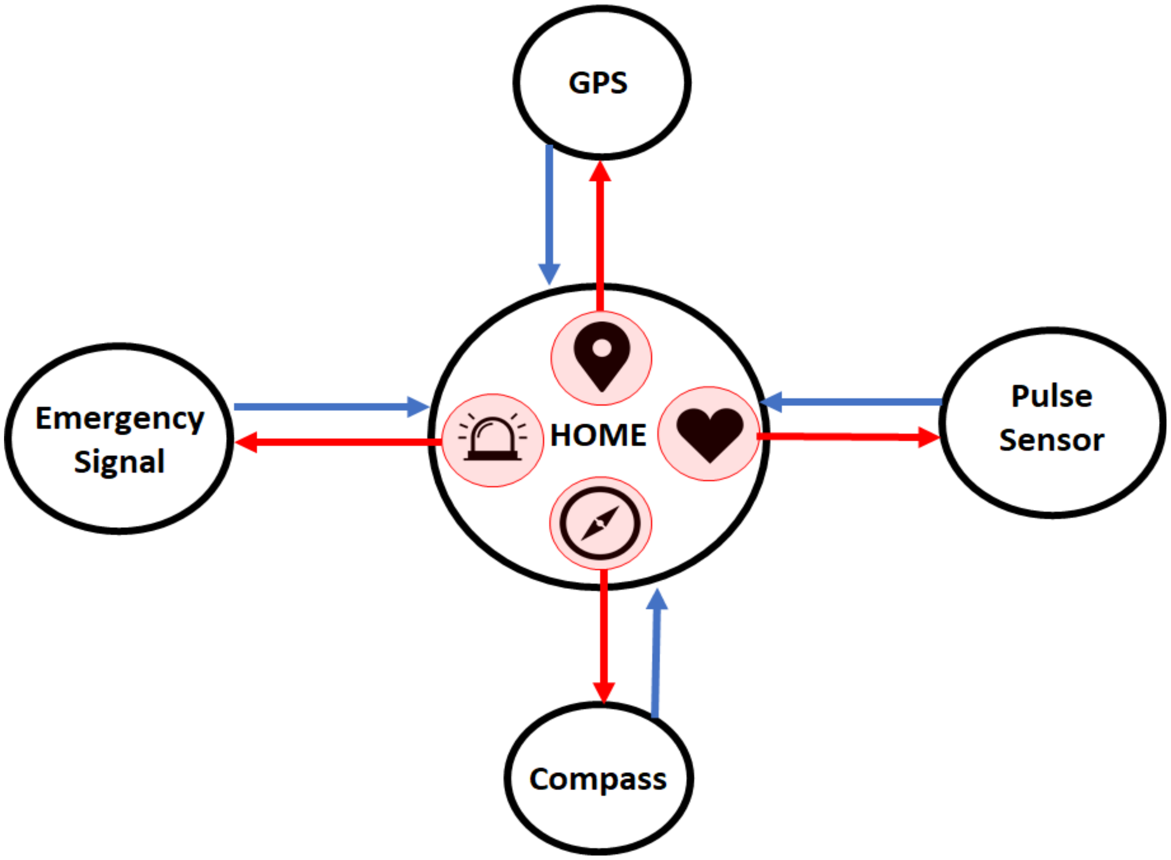

Figure 14. *A state diagram of how the hiking pole device software was set up. The red lines indicate the user touching the specific icon, and the blue indicates the user touching anywhere on*

the screen. The feature page stays on the main screen until the user touches the screen again to return home.

B. Issues Encountered

We encountered a couple issues while creating this product. The first major issue was encountered back when we were setting up our Serial Communication modules with the RS232 chip and running a loopback test. The voltage outputs on the RX/TX pins on the RS232 were not matching our expected values. After extensively checking the wiring we found it was that our ground pins were not fully soldered and were not grounding the wires properly. This was easily fixed by resoldering, and we were thankful we encountered this error early in the process.

Another issue we encountered was the Pulse Monitors were very finicky devices. The monitors easily burned after a couple uses and would output strange values. A major problem was found in converting Analog to Digital signal for the pulse sensor. In testing, the sampling range used in converting the sensor's output was not exactly the voltage range used by the sensor. It appears that internal resistance in the board reduced the power source voltage for the sensor from 5V to 4.5 or 5V. The REF pin on the Atmega328p was connected to the sensor's power source to ensure sampling is accurate. In addition the pulse sensor implementation was initially designed to utilize interrupts to process the sensor's signal periodically every 2ms. When integrated into the main design, interrupts seem to interfere with the display output. It is possible that the LCD display requires a consistent refresh while a periodic interrupt collided with the refresh mechanism.

Another issue encountered was that our compass sensor was giving us highly inaccurate readings. Even observing the Adafruit Libraries for the LSM303 did not seem to help as their unit conversions did not make sense and gave us very noisy values. All in all, due to the inaccuracies with that sensor we were only able to display the information read from the LSM303 rather than display a line on a compass bearing.

C. Memory Usage

Type	Usage	Percentage
ROM (Program)	14030 Bytes	42.8% Full
RAM (Data)	1755 Bytes	85.7% Full

Table 4. *Firmware Memory Usage on Atmega328p*

D. What we would do differently

There were many aspects of the project we would have approached differently. One of the main things we agreed on was doing more research on the parts we bought and checking the type of communication that each part required. It was difficult for our devices to be spread across 4 different types of communication with Serial, SPI, I2C, and

ADC. In hindsight, it would have been smart to pick devices that were limited to a few communication types.

In addition, making sure a lot of the sensors were using interrupts instead of polling would have been more stable in the long run. A definite issue with allowing interrupts was the LCD getting finicky; however, if we had more time to investigate the issue we would have been able to find the solution.

VI. Work Distribution

A. Distribution

	Alex	Ebrahim	Juliana	Jeffrey
System Design	28.33	28.33	28.33	15
Hardware Design	28.33	28.33	28.33	15
Hardware Assembly	28.33	28.33	28.33	15
Hardware Debugging	30	30	30	10
Microcontroller Software Design	31.67	31.67	31.67	5
Microcontroller Debugging	33.33	33.33	33.33	0
System Integration	30	30	30	10
Project Report (Oral)	26.67	26.67	26.67	20
Project Report (Written)	33.33	33.33	33.33	0

B. Signatures

[illegible]

EBRAHIM ALHADDAD

ALEXANDER LEE [illegible]

JULIANA ECHTERNACH [illegible]

QIHONG WANG [illegible]

VII. Appendix

Tables:

Inputs	Outputs
User Touch	LCD Display
GPS Information	RF Transmit Signal
Pulse Sensor	Pulse Beat Rate
Motion / Direction (Accelerometer)	Compass Position
RF Receive Signal	

Table 1. *The inputs and outputs used in the prototype.*

Part	Cost
GPS	$39.95

TFT LCD Touchscreen	$39.95
Accelerometer / Compass	$14.95
RF Transceiver (x2)	$19.90 ($9.95 per)
Pulse Sensor	$24.95
ATmega328p	$2.14
Total:	**$141.84**

Table 2. *The cost analysis of the parts purchased for the Smart Hiking Pole device.*

Variable Name	**Size**	**Purpose**
BPM (Beats Per Minute)	2 Bytes	Stores the final BPM value calculated using the latest signal read by ADC pin
IBI (Interval of Beat Irregularity)	2 Bytes	A changing value representing the wait time necessary to consider a pulse in the signal to be heartbeat. It is used to detect the drop in voltage following a pulse. It makes sure the signal is shaped like a heartbeat pulse and not noise
Pulse	1 Byte	A flag set when a pulse in signal is detected. It is used to start detecting a drop in voltage following a pulse
SampleCounter	4 Bytes	A counter that tracks the time between individual pulses
lastBeatTime	4 Bytes	A timestamp of the last measured heartbeat. Used with sample counter to determine the time interval between pulses to calculate BPM
P	2 Bytes	Stores the local maximum value of a peak
T	2 Bytes	Store the local minimum value of a trough

thresh	2 Bytes	Threshold value to determine whether the signal is high or low. It is pivotal in detecting pulses in signal and must be updated according to the middle value the pulse sensor stabilize around. The variable is initially set to the middle value of ADC conversion. Overtime, the local amplitude of the signal is calculated using P and T. The middle value of the amplitude is then stored in thresh.
amp	2 Bytes	Stores the local amplitude of the signal. The wave signal may change its center and range. Updating amp is necessary to update the threshold value used to detect pulses.
firstBeat	1 Byte	A flag to track the first beat. This flag is used to prevent updating BPM when the first beat of the signal is detected given it will result in producing an inaccurate reading
secondBeat	1 Byte	Second beat is used to start calculating BPM after the first beat
rate[10]	20 Bytes	An array storing beat interval time of the past 10 pulses detected. The content of the array is used to calculate the average BPM.

Table 3. *A table of all the variables used in processing the pulse sensor signal.*

Type	Usage	Percentage
ROM (Program)	14030 Bytes	42.8% Full
RAM (Data)	1755 Bytes	85.7% Full

Table 4. *Firmware Memory Usage on Atmega328p*

Figures:

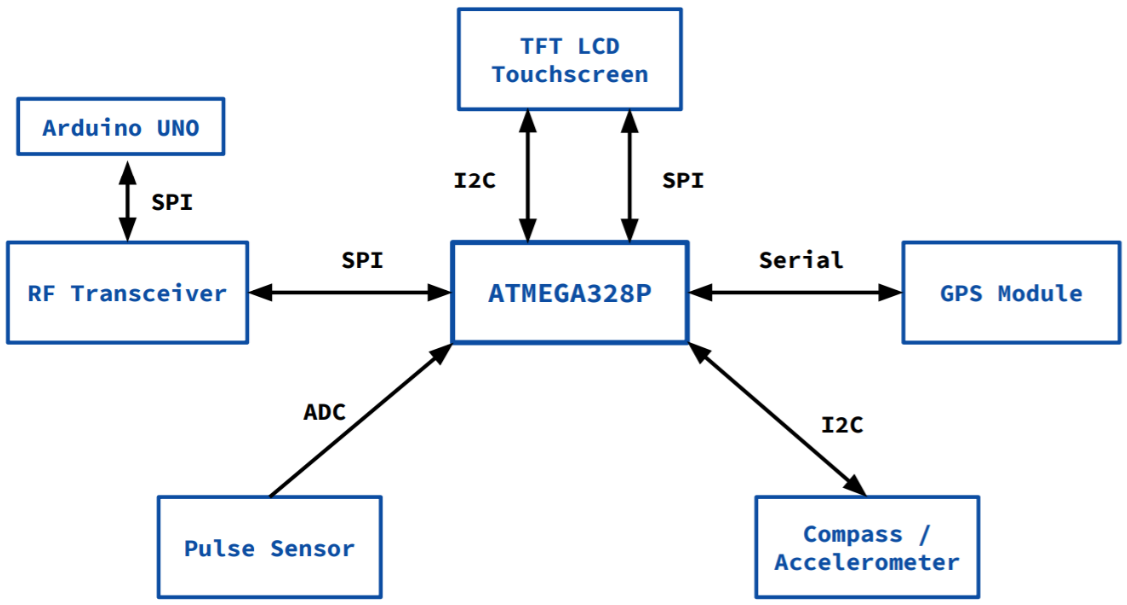

Figure 1. *Design Block Diagram. Every feature in our design is driven by the Atmega328P Microcontroller. The connecting arrows illustrate the communication method used to communicate between the chip and the devices.*

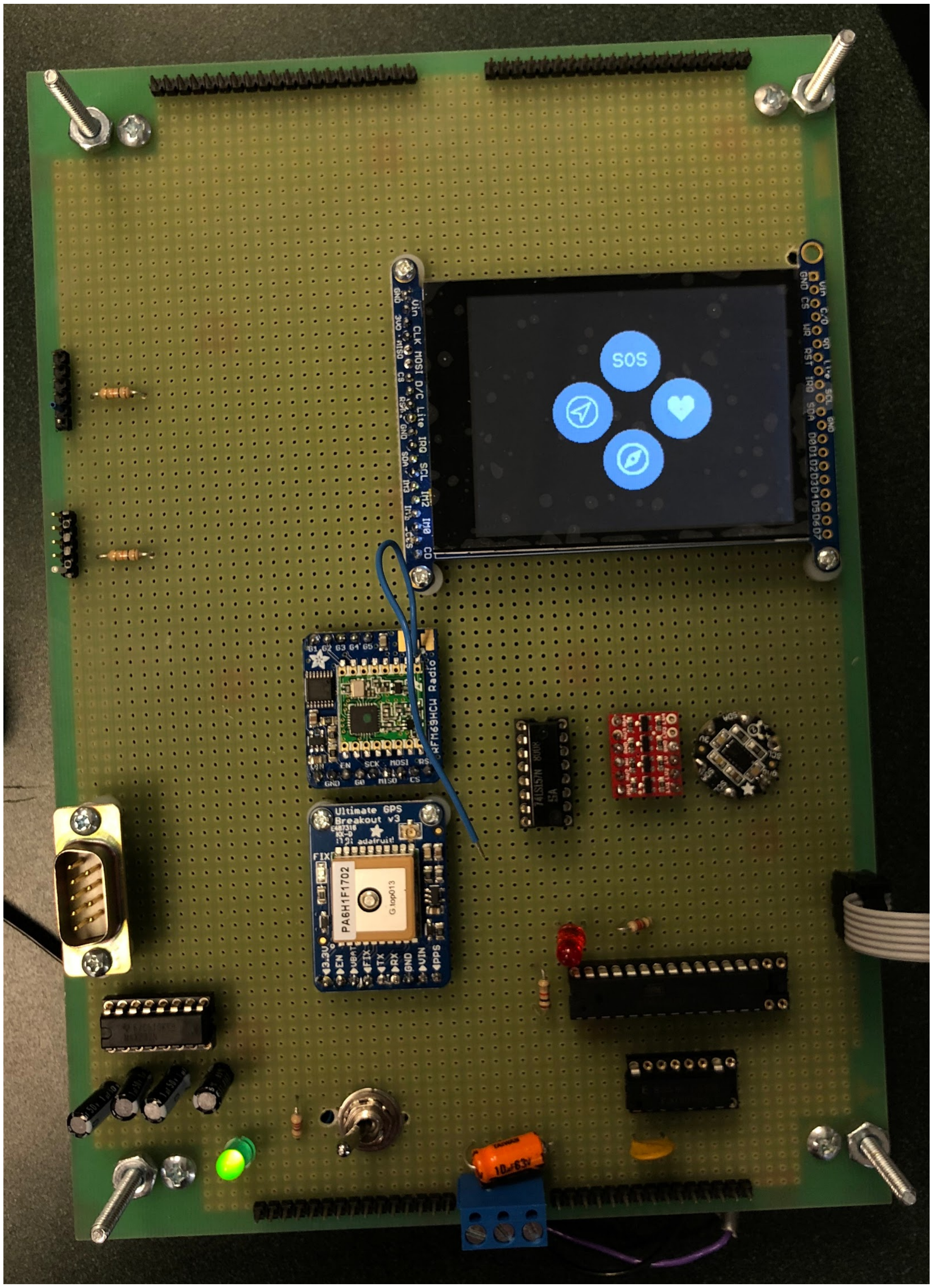

Figure 2. *Finalized Engineering Prototype (Top)*

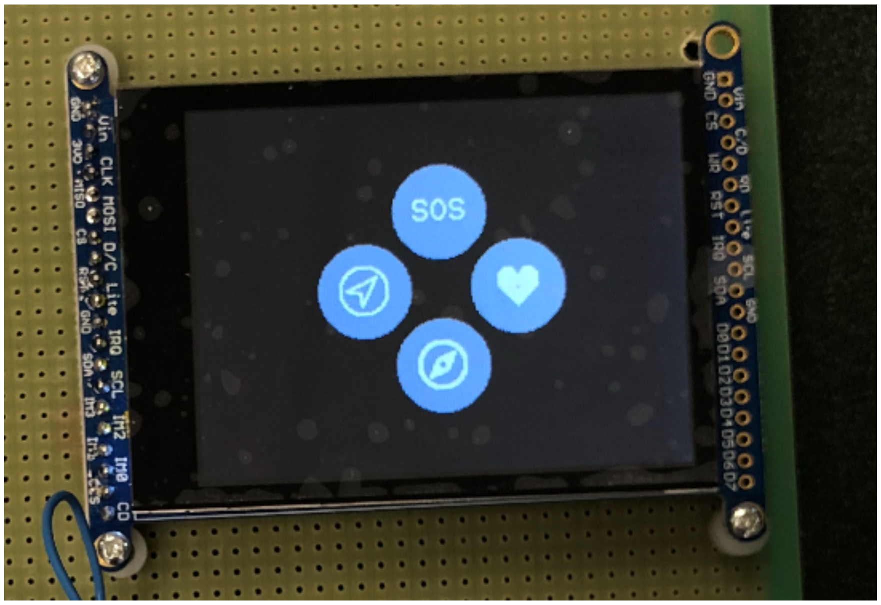

Figure 3. *Home Screen GUI showcasing the TFT LCD*

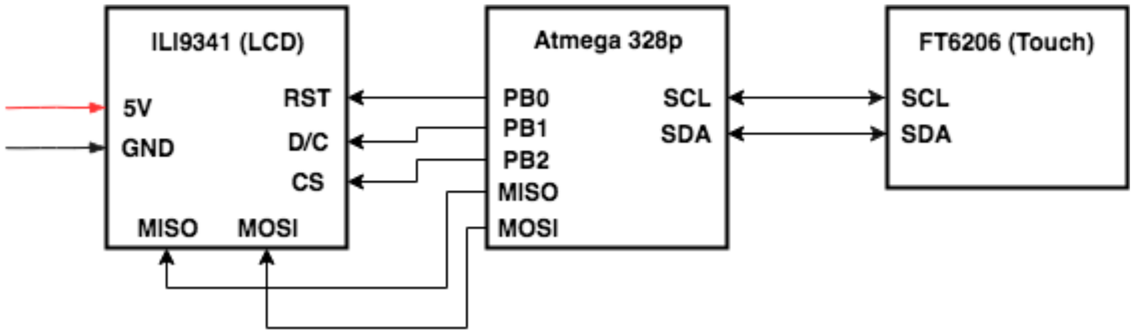

Figure 4. *TFT LCD Block Diagram*

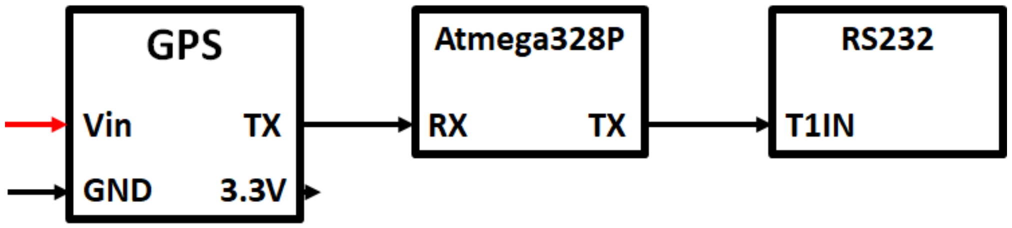

Figure 5. *Diagram of how the GPS was wired on the project board. The GPS Receives satellite information and outputs this through the (TX) pin which goes to the microcontroller RX pin. This is connected to the RS232 T1IN pin.*

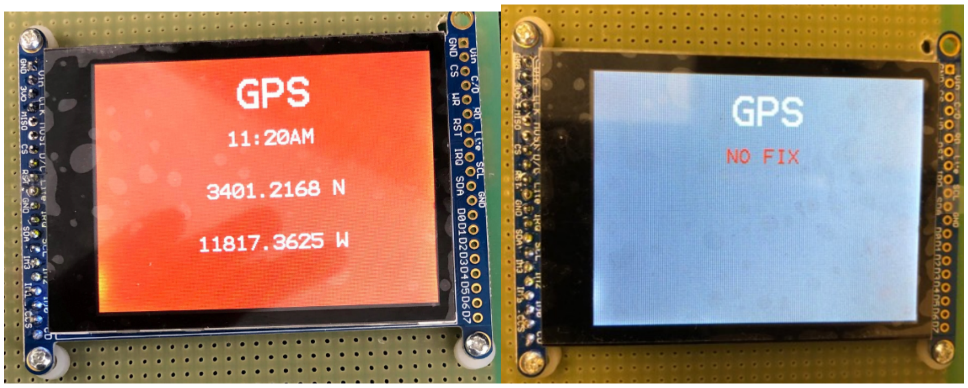

Figure 6. *Image of the GPS output on the LCD Display. The left image shows the GPS working with time, latitude, and longitude displayed on the screen. The right image shows the output if the GPS was unable to get a fix.*

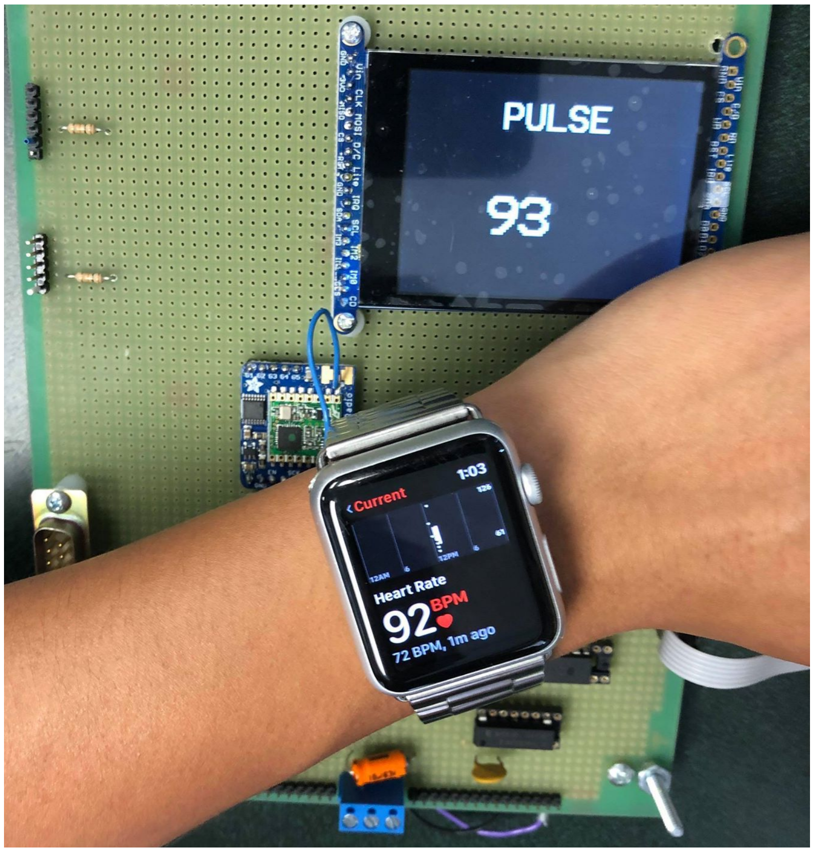

Figure 7. *Pulse Sensor compared to Apple Watch Pulse Reading.*

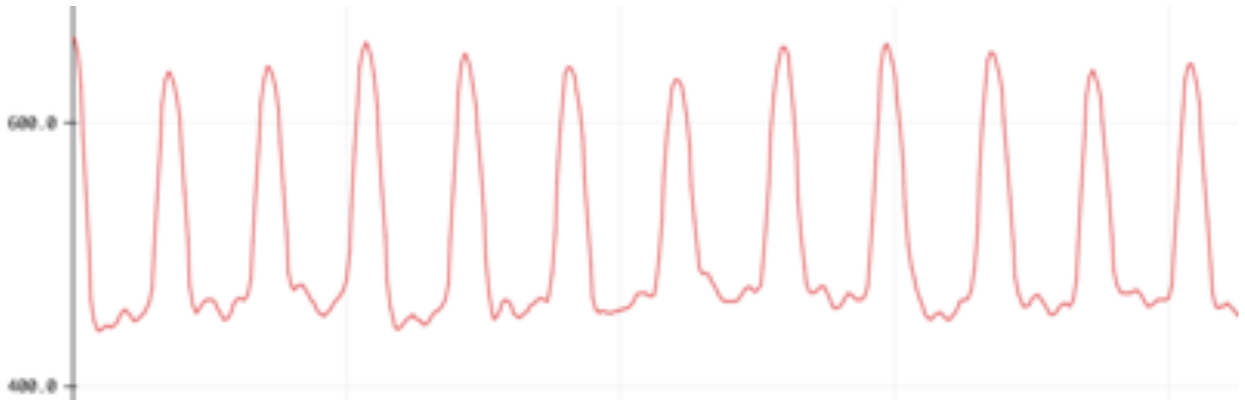

Figure 8. *A graph replicating the pulse sensor's output measured using the oscilloscope.*

The wave signal ranges between 0V to 5V and is centered at 2.5V.

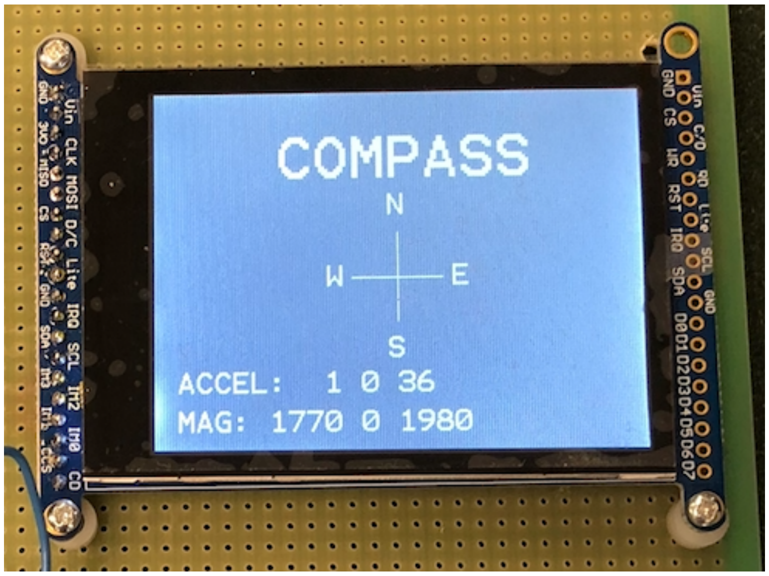

Figure 9. *Compass Interface.*

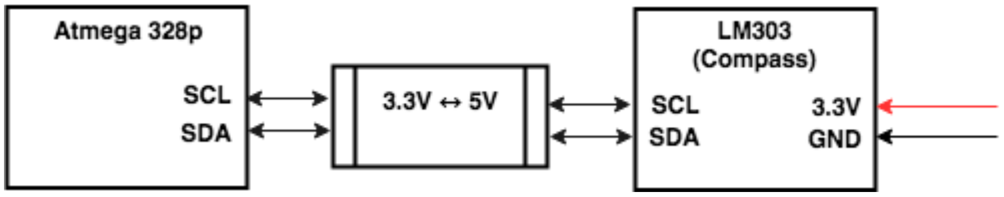

Figure 10. *Compass Block Diagram (LM303 should be LSM303)*

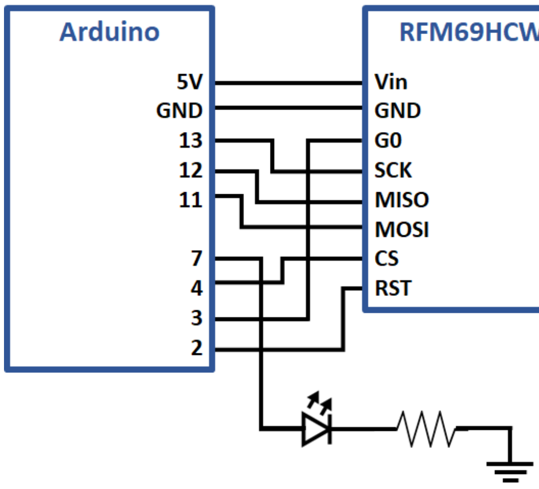

Figure 11. *Wiring diagram of the seconds RFM69HCW Radio connected to the Arduino UNO. We included an LED circuit to blink to indicate when information was sent and received.*

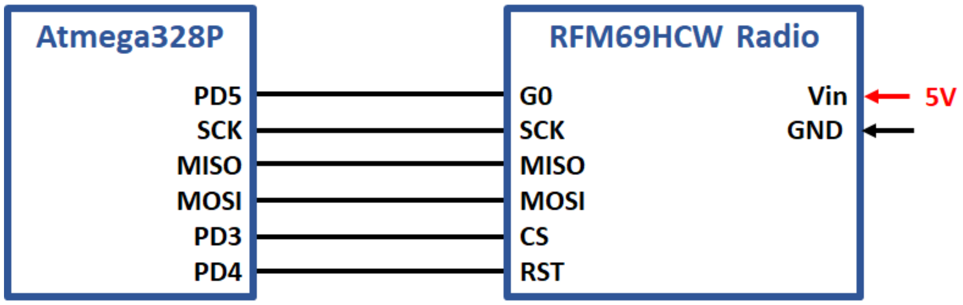

Figure 12. *Wiring diagram of the transmitting radio on the main board connected to the Microcontroller.*

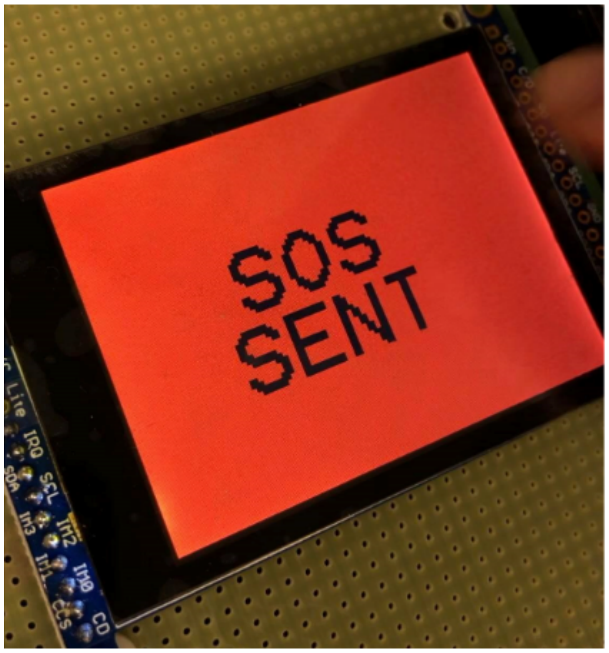

Figure 13. *Outputted message when the user touches the SOS icon on the LCD.*

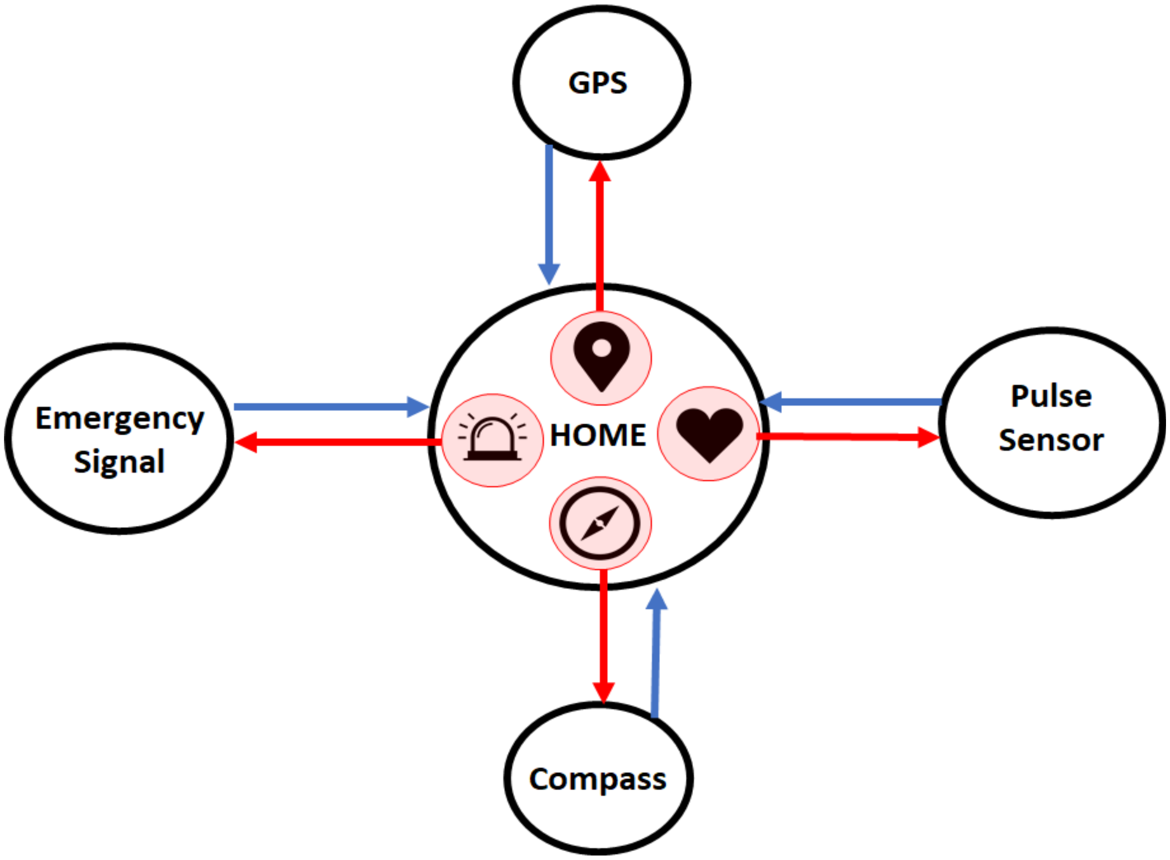

Figure 14. *A state diagram of how the hiking pole device software was set up. The red lines indicate the user touching the specific icon, and the blue indicates the user touching anywhere on the screen. The feature page stays on the main screen until the user touches the screen again to return home.*