Outdoor Safety Bracelet

By

Sameeth Gosike

Seth Katz

Samuel Sitzmann

Final Report for ECE 445, Senior Design, Fall 2020

TA: AJ Schroeder

Professor: Arne Fliflet

9 December 2020

Project No. 37

Abstract

In this project, we designed an Outdoor Safety Bracelet that allows a guardian/supervising person to keep track of a dependent in their care who is wearing the bracelet. Our aim was to build a comfortable bracelet that can wirelessly communicate with a monitoring device from a distance. Not only can it track a dependent's location within a certain radius, but it can also detect whether the bracelet wearer has had a fall. Additionally, both devices can send out alerts if an emergency occurs. The caretaker is able to view the state of the bracelet wearer on an easy-to-use touchscreen display. The project involved designing two devices, a bracelet and a monitor.

During the implementation, we faced several issues dealing with power supply, choice of parts, and PCB design. As a result, the finished prototype was not fitted onto a bracelet and the fall detection feature was not implemented. However, in the end we were able to demonstrate a working proof of concept to showcase most of the functionality of this idea. This report will detail the design, verification, and challenges faced during implementation as well as further thoughts on improvements that can be made.

Contents

1. Introduction

1.1. Problem and Solution Overview

Caretakers have a constant responsibility to keep track of whoever they are watching. Whether it be children, the elderly, or anyone else that needs constant supervision, keeping an eye on them can be a full-time job. If the caretaker looks away for even a moment they might lose track of their dependent and, depending on the situation, this could be incredibly dangerous.

Our objective is to make this job easier by creating a device that will track and report location and situational information to the supervising person. Specifically, this product will track a dependent's location using GPS and utilize an accelerometer to collect information about motion and impact to detect falls. These components are fitted into a comfortable, tamper-proof wristband/bracelet for easy wearability and comfort. This information is sent over wireless RF communication to a monitoring device possessed by the caretaker on which the dependent's location will be displayed to allow constant tracking within a large radius. Additionally, an alerting system is integrated to notify the caretaker if a fall has occurred or if the dependent is wandering out of a safe range. The RF components also allow for 2-way communication between the bracelet and monitoring device to send alerts to each other in an emergency situation.

Current tracking devices on the market require a monthly subscription to a service because they utilize SIM cards for wireless communication almost anywhere. Other devices utilize Bluetooth, however this provides a shorter range communication. In addition, these devices lack in providing important situational information. Our device will utilize 915 MHz RF to communicate over long distances without the need for a monthly service. Additionally, our device is not just for tracking location, but also sends situational motion information. This information can be important to gather a more complete picture of a dependent's state of safety. Caregivers can feel less stressed about constantly watching their dependent. Whether at the park, in a city, or even on one's own property, this device will be incredibly helpful to keep track of dependents and keep them safe.

1.2 Functions & Features

The following functions and features make our device unique and provide a more complete picture of a dependent's safety.

- Two way wireless communication between monitoring device and bracelet

- AES-128 bit encrypted, 915 MHz radio frequency communication
- Relative distance and location determination
- Fall detection using a multidirectional accelerometer
- Touchscreen display that allows user to track location, state of safety and flashes emergency alerts

1.3 Block Diagram

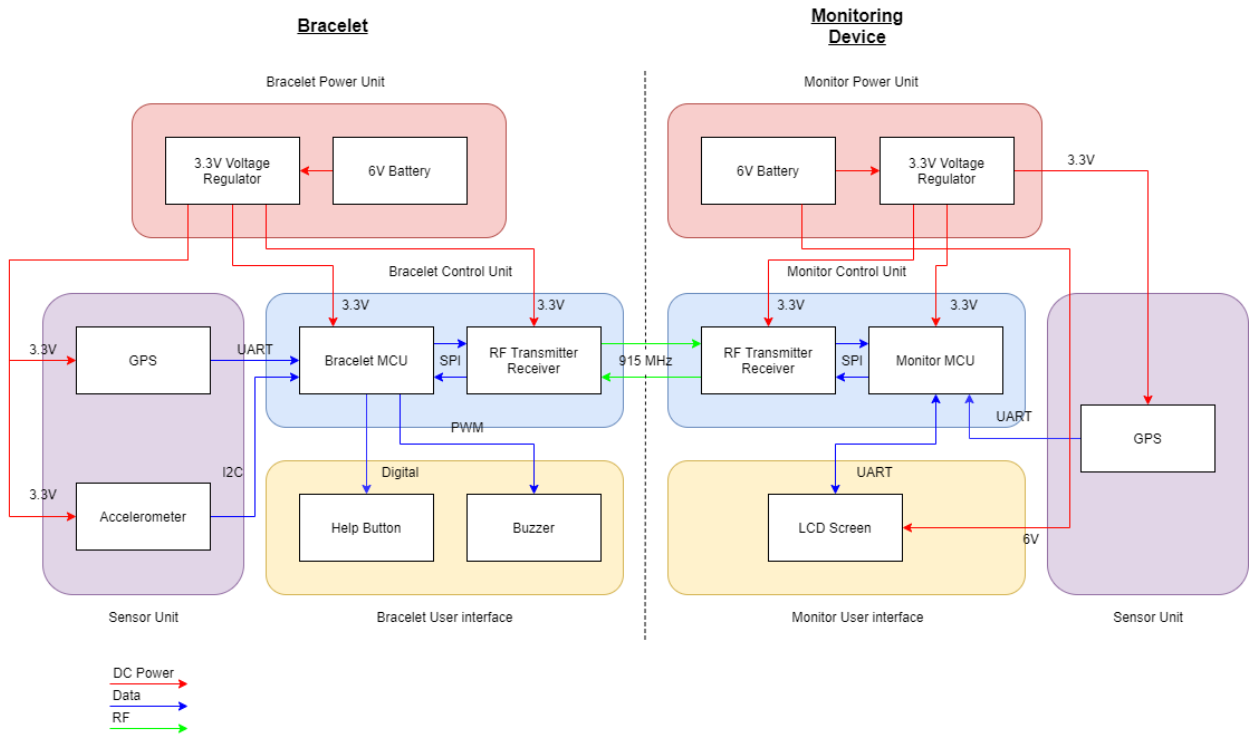

Figure 1: Outdoor Safety Bracelet Block Diagram

1.4 Block Descriptions

There are two parts to this design. The bracelet is the device that is carried by the dependent and the monitoring device is possessed by the caretaker or supervising person.

1.4.1 Power Unit

The power unit is responsible for supplying power to each module in both the bracelet and monitoring device. It consists of a 6V battery and a voltage regulator to step down the voltage for components that require less power. The power unit outputs both a 6V supply and a 3.3V supply.

1.4.2 Control Unit

The control unit on both bracelet and monitoring device consists of a microcontroller as well as a RF transceiver.

Bracelet Control Unit

The bracelet control unit is responsible for receiving data from the Sensor Unit, processing this data to extract the bracelet coordinates and to detect if the bracelet wearer has fallen, and subsequently sending this information wirelessly over RF communication for the monitoring device to receive. Additionally, it communicates an emergency alert to the monitoring device when the Help Button is pressed or activates the Buzzer when it is triggered from the monitoring device. The microcontroller collects GPS data in order to parse and extract the bracelet's coordinates. To detect if a fall occurred, it will use the accelerometer data collected in a fall detection algorithm described later in detail. The MCU packages any outgoing data, which includes bracelet coordinates, fall alerts, or help alerts, outputs it to the RF transceiver where it will be sent to the monitoring device's RF transceiver. When the buzzer is triggered on the monitoring device, the bracelet's RF transceiver will receive this alert and the buzzer will be activated by the MCU.

Monitoring Device Control Unit

The monitoring device control unit's primary functions are to calculate distance and relative location between itself and the bracelet as well as updating the screen display with location data and any triggered alerts. Once the bracelet's coordinates are received from the RF transceiver, the MCU will use this and its own coordinates to calculate the distance and relative location between the two. If it has received a Fall Alert or Help Alert, the MCU will update the screen to flash this alert to the user. Additionally, if the buzzer is triggered by the user, the MCU will package this information and send it to the bracelet over RF communication.

1.4.3 Sensor Unit

The sensor unit consists of the GPS module and accelerometer (the monitoring device's sensor unit consists only of the GPS module). The GPS module outputs NMEA-0183 data to the MCU which contains the device's coordinates. The accelerometer outputs motion data of the bracelet and will be used to detect falls.

1.4.4 User Interface

Bracelet UI

The bracelet's U.I. consists of a Help Button and a Piezo Buzzer. The Help Button is pressed when the bracelet wearer needs to alert the supervising person of an emergency. This alert will be sent to the monitoring device to be flashed on the screen. In the other direction, if the supervising person needs to alert the dependent, he/she can trigger the buzzer from the monitoring device. This alert is sent to the bracelet over RF where the buzzer is activated, outputting an audible sound to the dependent.

Monitoring Device UI

The monitoring device U.I. is fully implemented in a 2.4 inch touch screen LCD display. The purpose of the monitoring device U.I. is to communicate to the user all the important information relating to the whereabouts of the bracelet wearer including relative distance, relative location, and alerts. The U.I. also allows the user to contact the bracelet wearer to achieve two way communication between the monitoring device user and the bracelet wearer. The U.I. is completely implemented into the screen. Three alerts are displayed as lights and when an alert is triggered, the respective light flashes. The relative distance is printed below the lights in meters. The relative location is displayed on gauge. The angle between the two devices is calculated and returned to the gauge so that it points in the direction of the bracelet. Finally, a button at the bottom of the screen allows the monitoring device user to trigger a buzzer on the bracelet notifying the bracelet wearer to stay put or return to a predetermined location. The screen communicates with the MCU through serial UART protocol and requires the abilities to both transmit and receive data.

2. Design

2.1 Power Unit

2.1.1 6V Battery

To supply 6V, we combined two 3V coin batteries in series (3V+3V=6V), as shown in Figure 2. We decided to use two 3V coin batteries due to their availability, capacity, and price. Each battery has a capacity of 580mAh which is more than enough to support our systems requiring 170mA for over three hours.

$$\frac{2*580mAh}{170mA} = 6.8hours \quad \text{Eq. 1}$$

6V was necessary to provide a stable 3.3V output from the voltage regulator as well as to power the screen on the monitoring device. The screen operates at a voltage between 4.75V and 7V, so 6V is fine for operation.

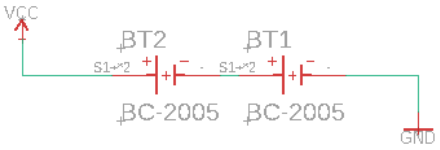

Figure 2: Battery Schematic

2.2.2 Voltage Regulator

The TPS7A03 low-dropout voltage regulator takes in the 6V from the batteries and regulates to a stable 3.3V to power the circuit. Decoupling capacitors stabilize the voltage on both the input and output, shown in Figure 3. This LDO is rated to allow 200mA, as specified in the datasheet [1]. This LDO was chosen because the required current between the modules is 170mA and our desired input voltage is 6V.

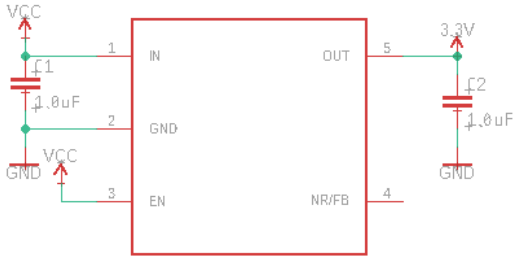

Figure 3: Voltage Regulator with Decoupling Capacitors

2.2 Control Unit

2.2.1 MCU

One of the microcontroller's main functions is communicating with the GPS, accelerometer, and RF transceiver which each utilizes a different I/O protocol. Namely, it inputs location data from the GPS over UART, inputs accelerometer data over I2C, and uses SPI to interface with the RF transceiver. On the monitoring device side, the MCU interfaces with the GPS over UART, the RF transceiver over SPI and the screen display over UART. In this use-case, speed and performance were not huge requirements for the design, therefore, the ATMega328P-PU 8-bit microcontroller was chosen, the same used on the Arduino Uno. In particular this part included all the interfaces needed for the peripheral devices being used plus additional digital pins that are needed for the user interface I/O. Schematics of the peripheral connections to the MCU can be seen in Figure 4.

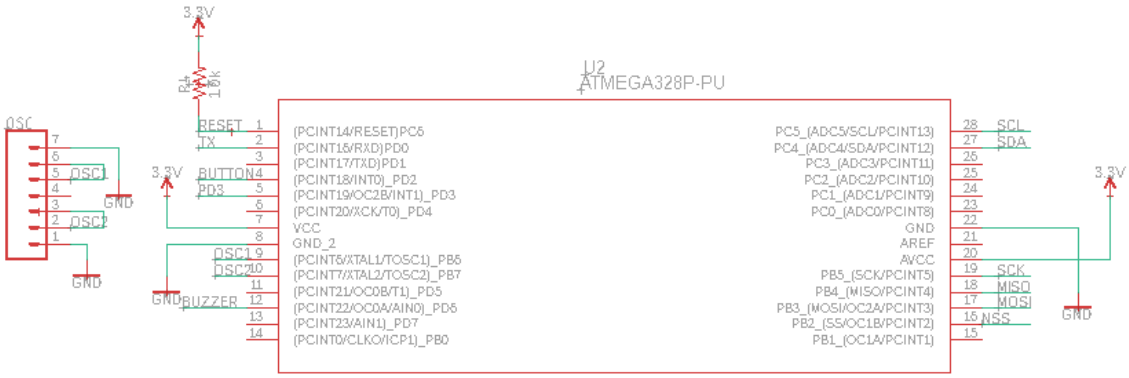

Figure 4: MCU Peripheral Connections

Software

On the bracelet side, the MCU will continuously read and parse the NMEA sentences outputting from the GPS to extract the bracelet coordinates as well as process the accelerometer data to detect if a fall has occurred. If a fall has not been detected, the MCU will package the latitude/longitude coordinates as well as the timestamp that the coordinates were sent by the GPS satellite. The details of the parsing will be detailed in Section 2.3.1. The package will then be sent over to the RF transceiver to send wirelessly to the monitoring device. However, if a fall has been detected, the MCU will package an alert message and send this over RF along with the location data. The details of the fall detection algorithm will be detailed in Section 2.3.2. The software flow diagram illustrating this loop can be seen in Figure 5. It is important to note that the diagram does not include the Help Alert interrupt function used to alert the supervising person of an emergency. Since the Help Button is connected to one of the MCU interrupt pins once pressed it will cause the

MCU to seize its current operation and immediately send out an alert message over RF to the monitoring device where it will be flashed on the display. This function is also included in the reverse direction - if the Buzz Alert button is pressed on the display on the monitoring device side, an alert message will be sent over RF to the bracelet. Once the bracelet MCU receives the alert message from the RF transceiver, it will sound the Piezo buzzer included on the bracelet at an audible frequency to alert the bracelet wearer.

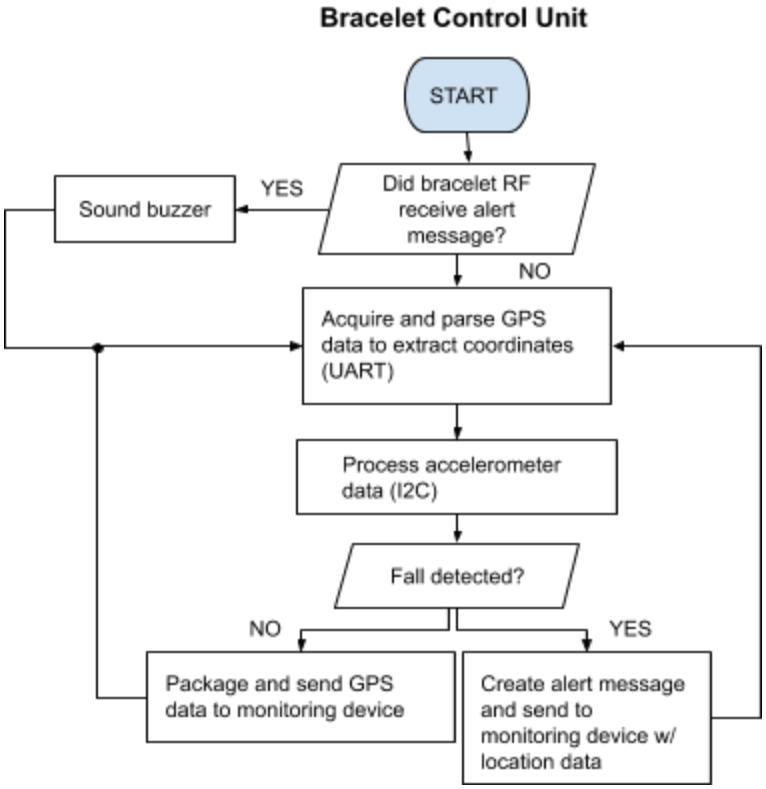

Figure 5: Software Flow Diagram for Bracelet MCU

On the monitoring device side, the MCU calculates the distance and relative location between the bracelet and monitoring device using the received coordinates and the coordinates extracted from its own GPS. The details of these calculations are detailed in the next section. The microcontroller will continuously update the screen these two location values to the display via UART. If a Fall Alert or Help Alert is received from the bracelet, it will update the display to flash an alert to the user. Additionally if the bracelet is starting to wander out of range it will display a corresponding alert. The software flow diagram illustrating this loop can be seen in Figure 6. As described earlier, if the user triggers the Buzzer Alert from the button on the touchscreen display, the MCU will package an alert message and send it to the bracelet RF. Once the bracelet receives this message, it will sound the buzzer on the bracelet.

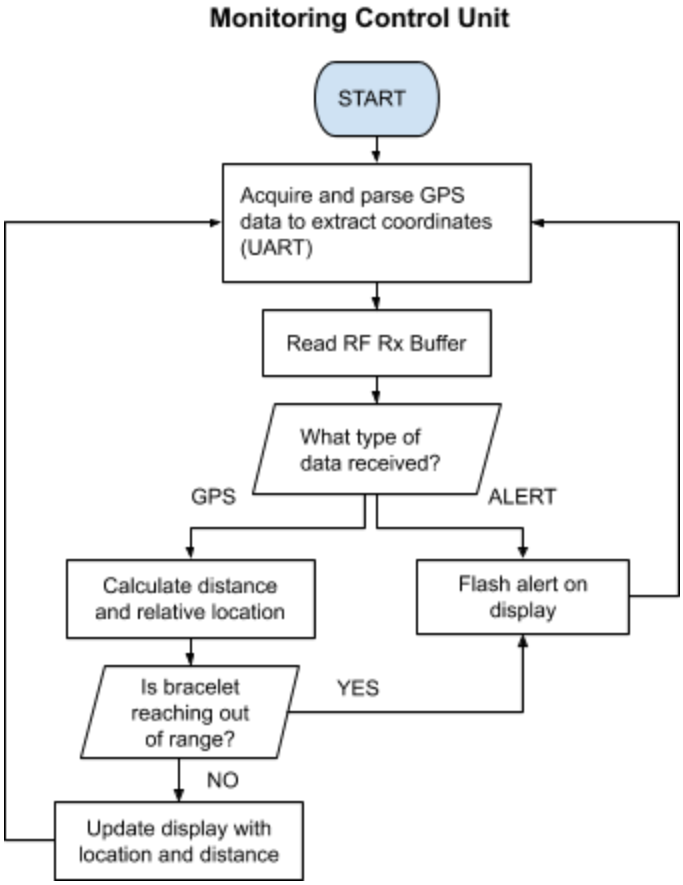

Figure 6: Software Flow Diagram for Monitoring Device MCU

Calculating Distance & Relative Location

Once the coordinates of both bracelet and monitoring device are extracted for a particular point in time, the Haversine Formula is used to calculate the distance between the two points. This equation determines the great-circle distance between two points, given the longitude and latitude coordinates of each. In other words it calculates the shortest distance over the earth's surface [2]. The Haversine formula is shown below

$$d = 2r * arcsin(\sqrt{sin^2(\frac{\varphi_2-\varphi_1}{2}) + cos(\varphi_1)cos(\varphi_2)sin^2(\frac{\lambda_2-\lambda_1}{2})} \quad \text{Eq. 2}$$

where d is distance in meters, (Φ_1, λ_1) and (Φ_2, λ_2) are the latitude/longitude coordinate pairs in degrees of the monitoring device and bracelet, respectively, and r is the radius of the Earth (6,371,000 meters). There are several methods to calculate distance between two GPS coordinates, including Vincenty's formulae, Law of Cosines, or even an equirectangular approximation (Pythagoras' Theorem). However Vincenty's proves to be more computationally intensive than Haversine's, the Law of Cosines is less accurate, and since an equirectangular approximation assumes the Earth has a flat surface it is only accurate

for small distances [2][3]. The Haversine Formula is much more reasonable in terms of computational performance and has shown an average error percentage of 0.07% for distances less than 500m.
The relative location is given using compass direction with cardinal direction and a bearing in degrees (i.e. N is 0°, E is 90°, S is 180°, and W is 270°). Once the distance between the bracelet and monitoring device is computed, the horizontal distance is calculated using Haversine's Formula. Using both distance values, simple trigonometry is used to calculate the angle between the horizontal and hypotenuse created by the two devices, $\Theta = cos(\frac{d_x}{d})$. The theta value is adjusted to match that of a compass dial. For example, if the bracelet is NE of the monitoring device, the angle is subtracted from 90° to give the actual bearing.

2.2.2 RF Transceiver

The RFM69HCW transceiver module was chosen because it is relatively cheap, can transmit up to 500 meters, has low power requirements, and is known by a member on our team. The module operates at 3.3V drawing a maximum of 16mA when receiving and 130mA when transmitting [4]. The module has AES-128 bit encryption capability to automatically encrypt each message sent. The module communicates in the ISM band at 915 MHz which is an open frequency band for the industrial, scientific, and medical space. This module communicates with the MCU using the SPI protocol and is interrupt driven. Our group utilized a ¼ wavelength antenna for each module. This can be calculated using the equation $\lambda = c \div f$ where λ is the wavelength, c is the speed of light in air, and f is the frequency of the carrier waves. Therefore, our antennas were about eight centimeters in length.

2.3 Sensor Unit

2.3.1 GPS

The GP-20U7 GPS Receiver was chosen for this design to determine the location of both bracelet and monitoring device. This module is capable of providing position and time feedback at a rate of 1Hz with location accuracy of about 2.5m, well within eyesight. According to the datasheet [5], given its low power consumption (40mA at 3.3V) and its relatively small size (18.4 x 18.4 x 4mm - smaller than the dimensions of an Apple Watch [6]) the part was deemed fit for our design. Power consumption is important since these are portable devices with limited battery supply. It was known that the GPS would be the largest component of the circuit and so to adhere to a watch-like design for the bracelet, this part fit our size constraints.

Parsing GPS Output

The GPS continuously outputs NMEA-1083 in different record formats data through a UART interface to the MCU at a rate of 1x/second. To extract the coordinates of the device, the RMC sentence was parsed accordingly. Table 1 shows the RMC data fields, where each sequential field is comma separated. Figure 7 depicts how the MCU software parses and saves the latitude, longitude and timestamp the coordinates were read.

Name	Description
Message ID	RMC protocol header ($GPRMC)
UTC Position	hhmmss.sss
Status	A=data valid or V=data not valid
Latitude	ddmm.mmmm
N/S Indicator	N=north or S=south
Longitude	ddmm.mmmm
E/W Indicator	E=east or W=west
Speed Over Ground	
Course Over Ground	
Date	ddmmyy
Magnetic Variation	E=east or W=west
Checksum	
<CR><LF>	Message termination

Table 1: RMC sentence data fields

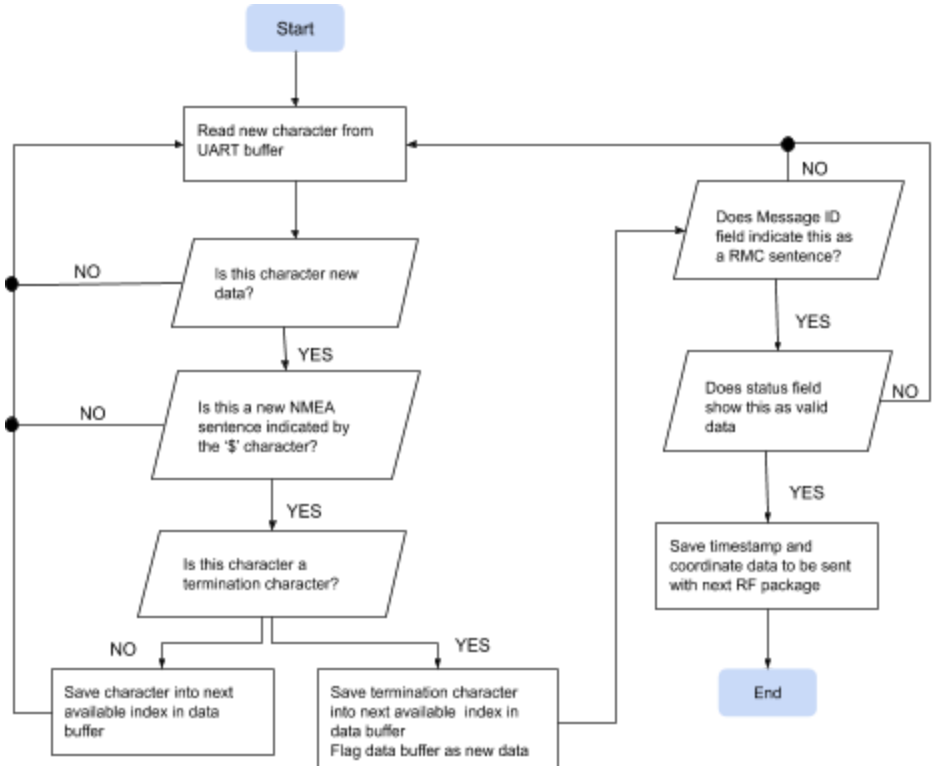

Figure 7: Software flow diagram for parsing coordinate data from GPS

2.3.2 Accelerometer

The Kionix KX124-1051 Tri-axis Digital Accelerometer was chosen for this design to detect if a fall has occurred. It is capable of outputting the acceleration rate of an object in the x, y, and z directions to a range up to ±8*g*, where *g* is the rate of acceleration due to gravity (9.81 m/s^2). The MCU communicates with the chip through I2C protocol. This particular accelerometer was chosen according to the features given in its datasheet [7]. The accelerometer is fitted with an ASIC to detect free-falls, jerk events (sudden peaks in acceleration), and change in orientation, which are all required to detect a person falling. Given the extremely small size (3 x 3 x 0.9mm) and low power consumption (145µA at 3.3V with further power optimization built into the SoC), this part was well suited for the initial bracelet design.

Fall Detection

A person falling, whether from a height or from a standing position, typically consists of 4 phases. Before a fall occurs, the acceleration of a person towards the Earth's surface, a_z, is equal to *g* due to the force of gravity. When a fall starts to occur, a_z approaches 0 m/s^2 and a falling person enters a short period of free-fall. The second phase is impact which can be characterized by a sudden spike in acceleration. This spike event illustrates a sudden and

great change in acceleration, or jerk. The impact is followed by a change in orientation and a prolonged rest state where $a_z = g$. If this sequence of events occurs in a short time frame, about 2-3s, it can be considered a fall.The graph in Figure 8 illustrates the accelerometer data of a phone being dropped from about 5ft which clearly depicts this chain of events.

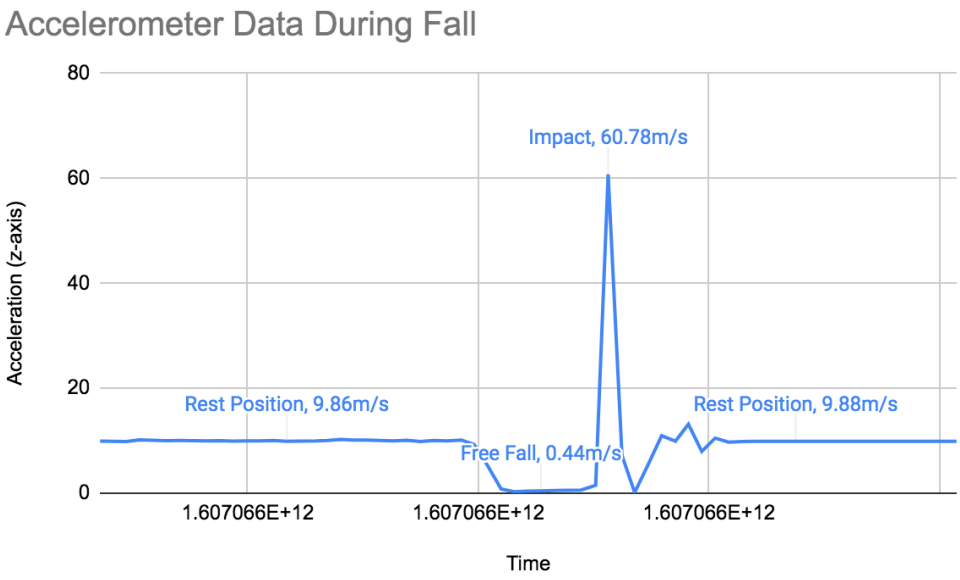

Figure 8: Acceleration (z-axis) of a phone falling from 5ft height

The KX124-1051 is a nifty device since it is capable of sending an interrupt signal to the MCU if either a free-fall, jerk event, or change of orientation occurred. Unfortunately due to issues discussed in Section 3.3.2, the part was unusable. However given that the issues could be surpassed, the bracelet MCU would have been programmed to wait for an free-fall interrupt signal to go high. Once this occurs, it would subsequently check for a jerk, change in orientation and a prolonged rest state. Using a counter, it would check if this chain of events occurred within a 2-3s. If so, a fall would have been detected, and an alert message would be sent over RF to the monitoring device to notify the supervising person.

2.4 User Interface

2.4.1 Monitoring Device TouchScreen

The Nextion Basic Display NX3224T024 touch screen encompasses the entire user interface for the monitoring device. We chose this device because of its existing library and touch screen capabilities. Along the top are 3 “LEDs”, that flash for the 3 alerts: help button pressed, distance almost out of range, and fall detected. Beneath that is the relative distance in meters, and a gauge showing the direction to the bracelet. On the bottom is the buzz

button, which the guardian can press to cause the buzzer on the bracelet to go off. The layout of this interface can be seen in Figure 9.

Figure 9: Monitoring Screen Layout

2.4.2 Bracelet Help Button

The Help Button on the bracelet allows the wearer to communicate with their guardian, alerting them that help is needed. It consists of a simple button, and a pull down resistor shown in Figure 10, so that the MCU receives a high signal on an interrupt pin whenever the button is pressed.

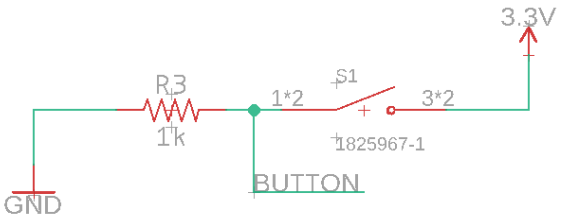

Figure 10: Button Circuit Layout

2.4.3 Bracelet Buzzer

The buzzer on the bracelet is a PS1240 Piezo Buzzer that allows the guardian to alert the wearer with a predetermined signal. The buzzer will buzz on and off 5 times. It is connected to an NPN bipolar junction transistor as shown in Figure 11, which amplifies the PWM signal from the MCU.

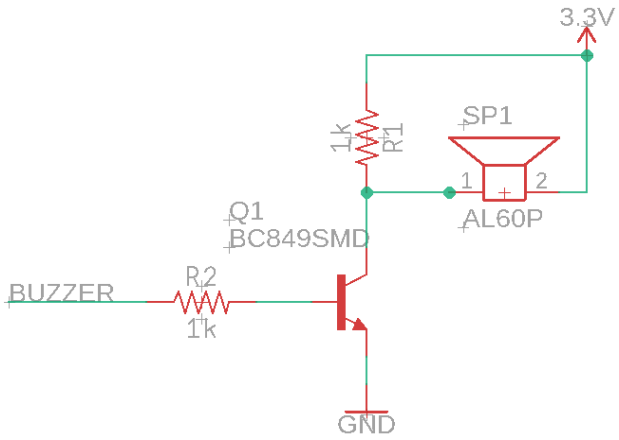

Figure 11: Buzzer Circuit Layout

3. Design Verification

3.1 Power Units

3.1.1 6V Battery

To verify our battery, we connected the positive and negative leads of the batteries to a multimeter, which confirmed that each battery output was at 3V, and confirmed that when combined in series, the voltage output totaled 6V. However, after inserting the batteries on the PCB we saw a huge drop in voltage as to where the voltage across both batteries was essentially zero. We believe this may have been due to a fault in the PCB since we tested a variety of potential causes including high resistance in the battery holders, reverse polarity, and loose connections. None of these tests showed any resolve. As a result we attempted to attach the batteries together off the PCB. While the voltage issue was resolved, when applied to a load the battery voltage was unstable and was unusable. Instead we opted to use a 5V supply from an arduino board.

3.2.2 Voltage Regulator

We confirmed the voltage regulator worked correctly by using a multimeter on both the input and output voltages, which showed the input at 6V and the output at 3.3V, meeting the requirement listed in Appendix A. However, when the bracelet ran with all the modules integrated, the regulator was forced to supply more than 200mA, the maximum output of the LDO according to the datasheet [1]. Although we calculated the max current draw of our system to be 170mA we believe the RF module drew over its max current rating when transmitting. This caused several of the voltage regulators to burn out. Switching to the Arduino board for a power supply also resolved this issue as the 3.3V voltage regulator on the board can supply up to 600mA.

3.2 Control Units

3.2.1 MCU

The MCU requirements listed in Appendix A are having a SPI, I2C, UART, and at least 4 additional digital IO pins in order to interface with each peripheral device included in the design. This was quickly verified using the ATMega's datasheet [8]. To further ensure that all devices were able to communicate with the MCU, each part was connected to the MCU which was programmed to input and/or output data from the peripheral. The GPS was connected to the UART serial pins on the MCU which was programmed to read and print the GPS location data. The accelerometer was connected across the I2C interface on the MCU which was programmed to read the accelerometer's Device ID from an internal register. This ensured that communication was stable both ways since the MCU had to send the slave and register address to the accelerometer and then read the Device ID register's data. A RF transceiver was connected to the SPI pins of the MCU which was programmed to read the Device ID and temperature of the RF chip. This also ensured 2-way communication between the two devices. Lastly, the MCU was programmed to light an LED upon pressing the connected Help Button and subsequently sound the bracelet Buzzer. All the requirements were met and communication with each peripheral was successful.

3.2.2 RF Transceiver

The transmission of our RF modules were verified in two ways. We first attached the probe of an oscilloscope to the antenna of the module and triggered a transmission. With a single transmission we could see a small jump in the waveform displayed on the oscilloscope, but with continuous transmissions a 915 MHz waveform was displayed on the scope. Additionally, an external software defined radio (SDR) module was used to ensure transmission at the correct frequency. Using the SDR module along with SDR software, we

were able to see spikes appear on a frequency plotter centered at 915MHz for both modules. Both methods were helpful in verifying transmission and debugging the devices.

Once transmission was verified, we utilized an Arduino library made for the RFM69 module to easily program the modules and test communication between them. We were able to run test code on the MCU to send desired messages between the two modules and verify the modules were receiving the others messages.

3.3 Sensor Unit

3.3.1 GPS

According to Appendix A, the GPS was acquired to output latitude/longitude coordinates at least 1x/second with an accuracy of about 0.00001°. This was verified by powering the modules, having the MCU read the GPS data over UART, and printing the data to a Serial monitor. We were then able to verify that the GPS indeed output latitude/longitude coordinates and the 1Hz update by seeing the satellite timestamps were each 1 second apart. Additionally, we allowed the modules to run overnight to monitor any fluctuations in returned data and compare the received coordinates to the actual coordinates using Google Maps. The data showed some fluctuation, but up to the tolerance specified in the data sheet, around 2.5 meters [5]. Although this did not meet our accuracy requirement, a 2.5 meter tolerance was deemed fit for tracking the bracelet over larger distances where a caretaker might not have a direct eyeline to the dependent.

3.3.2 Accelerometer

To verify the accelerometer, it was directly connected to the bracelet PCB that was designed instead of an Arduino board. This was done due to the fact that the accelerometer IO pins use 3.3V and the Arduino board pins operate at 5V. Since a level converter was not at hand to downstep the Arduino IO voltage level, the accelerometer had to be soldered on the PCB and connected directly to the ATMega chip, which uses 3.3V levels on its pins. This led to a number of problems with connectivity and accidental shortage of the accelerometer due to its extremely small size, depicted in Figure 13. Unfortunately due to this, the accelerometer was neither verifiable or usable as this required quite a skilled hand for soldering. As a result, one of the main features of this project, Fall Detection, had to be scrapped.

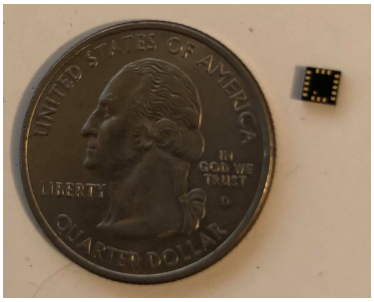

Figure 13: Accelerometer Size Comparison

3.4 User Interface

3.4.1 Monitoring Device TouchScreen

We used the screen to verify many other components, such as the button and buzzer on the bracelet, as well as the communication of the RF modules. Simply by observing and interacting with the screen, we knew that both the screen and the other modules were working as expected.

3.4.2 Bracelet Help Button

To verify that the button was hooked up correctly and working with the MCU, we pressed the button, which was connected to an interrupt pin on the MCU. Whenever we pressed the button, the Serial Monitor would show an alert saying it recognized the button being pressed.

3.4.3 Bracelet Buzzer

We verified the buzzer simply by connecting it to an MCU digital pin and driving it with a PWM. This successfully verified that the buzzer was operational. Through some trial and error, we settled on a frequency of about 261 Hz that was both loud and well within the audible range of human hearing.

4. Costs

Our design work cost $40/hour, 10 hours/week, over 16 weeks for three people. The cost of all the supplies can be seen in Table 2.

$$3 * \frac{\$40}{hr} * \frac{10hr}{wk} * 16wks * 2.5 = \$48,000 \quad \text{Eq. 3}$$

Parts	Prototype	Bulk
Buzzer (PS1240)	$1.50	$1.13
Bracelet MCU (ATmega328P)	$2.01	$1.67
Monitoring MCU (ATmega328P)	$2.01	$1.67
Accelerometer (KX124-1051)	$1.88	$0.88
GPS x2 (GP-20U7)	$17.95	$17.95
Voltage Regulator x2 (LD1117-3.3)	$1.25	$1.00
RF Transceiver x2 (RFM69HCW)digikey	$5.95	$4.99 on Amazon
Battery x4 (CR2450)	$0.79	$0.55
LCD Screen (ESP8266)	$21.49	$21.49
Button (ALCOSWITCH)	$0.19	$0.16
PCB (PCBway)	$5.00	$0.37
Total	$63.64	$53.51

Table 2: Materials Cost Summary

5. Conclusion

5.1 Executive Summary

In the end of the semester, our final product did not showcase all the goals that were intended, including fitting onto a comfortable bracelet and the capability to detect falls. However during the implementation process, our team was able to learn a great deal about the design, build, and verification process. We learned that much more planning is needed in the PCB design to make sure that all modules can be tested and programmed accordingly by exposing the correct connections. Careful thought must also go into choosing parts that are workable and have higher tolerances as seen in our failures with the accelerometer and LDO. Additionally, with the special challenge of COVID-19 this year, we were able to realize that a stable test environment for remote programming and collaboration was needed which was successfully built using a Raspberry Pi and VNC software. Although the final design was implemented on a breadboard, we were able to display most of the core functionality of our idea and are happy that we were able to push through almost all of our challenges to create a working design.

5.2 Going Forward

To further work on this project, we intend to advance our existing goals and add new high level goals to our system. In terms of advancing our goals, we will continue to work on stable RF communication at increasing distances between the bracelet and the monitoring device, using a longer antenna, and implement fall detection on a more suited accelerometer package. Detection of a dependent drowning can be added by using a water/pressure sensor in combination with the accelerometer. Possibilities of added features can also include detecting high sun exposure and monitoring vitals.

5.3 Ethics and Safety

As a product designed to keep susceptible people safe, there are many safety precautions that we had to keep in mind throughout the project. First and foremost, we have a responsibility to ensure that the physical devices themselves are safe and will not harm a user. This means parts must be large enough to not be swallowed by children, electronics are safely housed and do not pose any concern, there are no sharp or pointed edges… Although we were unable to make a compact design/housing for the devices, this idea is something that is extremely important as we move forward with the project and future designs. Especially considering this product would most likely be utilized by parents and their children. According to the New York Department of Health, choking is the fourth

leading cause of unintentional death in children under five years old [9]. There are other types of bracelets on the market for children and we would ensure that there would be no small pieces that could fall off and/or be swallowed.

Another concern is over data transmission through radio frequency communication. With radio communication you must keep data safety in mind. Especially when the data contains any type of personal information. In our case the data being sent is only related to the location of a person and can only be accessed over a range of 500 m. Although it is unlikely that someone would be close enough to extract this information, it is still important that this concern is accounted for. Similarly, someone may try to send incorrect data to throw off a caretaker and lead them away from the location of the bracelet wearer. To prevent any type of data sniffing/tampering, we utilized AES-128 bit encryption on all RF messages. This way only two devices with the same key can communicate with each other. Any other message will be ignored. Another concern with RF is whether or not the energy transmitted is enough to cause damage to users and/or people in the vicinity. The FCC has very clear regulations on the amount of RF energy that can be emitted by a device to limit exposure to human tissue. At high power levels/kg and high frequencies, there is potential to do damage [10]. Because of this potential the FCC decided to regulate the specific absorption rate (SAR) for humans in different contexts. The monitoring device and bracelet fall under the limits for general population/uncontrolled exposure which has a SAR limit of 0.08 W/kg as averaged over the whole body and a peak spatial average SAR for extremities (e.g. hands, wrists, feet, ankles, and pinnae) of 4 W/kg averaged over any 10 grams of tissue [11]. The maximum transmit power of our transceiver is 100mW [12] which averaged over an extremity and body is much less than the FCC limits.

Finally, one of the most important concepts while working on this project and will continue to be important through future work is reliability. “We … in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession… agree: to accept the responsibility in making decisions consistent with the safety, health, and welfare of the public…” [13]. We have a responsibility to design reliable products especially when caretakers will be trusting our product to “watch over” their subject. Of course there are potentially legal ways to get out of liability, but this is often overlooked by consumers. On top of that the idea of the product is to make the job of caretakers easier and therefore we must ensure that the devices are reliable and have fail safes in place if something goes wrong.

6. References

[1] Texas Instruments “Low-Dropout Voltage Regulator” TPS7A03 datasheet, July 2019 [Revised April 2020]. Available: https://www.ti.com/lit/ds/symlink/tps7a03.pdf?ts=1604280847082&ref_url=https%25A%252F%252Fwww.ti.com%252Fproduct%252FTPS7A03

[2] Movable Type Scripts “Calculate distance, bearing and more between Latitude/Longitude points”, [Revised February 2019]. Available: https://www.movable-type.co.uk/scripts/latlong.html

[3] Neova Solutions “Haversine Vs Vincenty: Which is the best?”, October 2019. Available: https://www.neovasolutions.com/2019/10/04/haversine-vs-vincenty-which-is-the-best/

[4] RFM69HCW RF Transceiver Datasheet (Sparkfun). [Online] Available: https://cdn.sparkfun.com/datasheets/Wireless/General/RFM69HCW-V1.1.pdf [Accessed: 9-Dec-2020]

[5] GP-20U7 GPS Receiver Datasheet (Sparkfun). [Online] Available: https://cdn.sparkfun.com/datasheets/GPS/GP-20U7.pdf [Accessed: 9-Dec-2020].

[6] Apple Watch. [Online] Available: https://www.apple.com/watch/ [Accessed: 9-Dec-2020].

[7] KX124-1051 Accelerometer Datasheet (Kionix). [Online]. Available: https://www.mouser.com/datasheet/2/348/KX124-1051%20Specifications%20Rev%201.0-1019154.pdf [Accessed: 9-Dec-2020].

[8] ATMega328P MCU Datasheet (Microchip). [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf [Accessed: 9-Dec-2020].

[9] New York Department of Health “Choking Prevention for Children”, [Revised April 2017]. Available: https://www.health.ny.gov/prevention/injury_prevention/choking_prevention_for_children.htm

[10] RF Safety FAQ (Federal Communications Commission). [Online] Available: https://www.fcc.gov/engineering-technology/electromagnetic-compatibility-division/radio-frequency-safety/faq/rf-safety#Q6. [Accessed: 1- Oct- 2020].

[11] 47 CFR § 1.1310 - Radiofrequency radiation exposure limits. (Legal Information Institute). [Online]. Available: https://www.law.cornell.edu/cfr/text/47/1.1310 [Accessed: 1- Oct- 2020].

[12] RFM69HCW Datasheet (Hoperf Electronic). [Online]. Available: https://cdn.sparkfun.com/datasheets/Wireless/General/RFM69HCW-V1.1.pdf. [Accessed: 1- Oct- 2020].

[13] IEEE Code of Ethics (IEEE). [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 16- Sep- 2020].

Appendix A: R&V Tables

Module Name	High-Level Requirements	Points
Control Module (MCU)	• Module should successfully acquire and parse data from GPS (UART) and accelerometer (I2C) to calculate location and detect fall • Module should be able to receive alert interrupts and activate buzzer/flash alert on screen • (Bracelet) Module should be able to package GPS/Alert data and send to RF module • (Monitoring) Module should be able to update screen with Location/Status data	10
RF Transceiver Module	• Module should be able to successfully send and receive data to/from bracelet/monitoring device over wireless RF communication at 915MHz	5
GPS Module	• Module should be able to obtain bracelet/monitoring device's coordinates and send to MCU over UART • Should update coordinates every 1 second (1Hz) • Should be able to be accurate to 0.00001°	5
Fall Detection Module (Accelerometer)	• Module should be able to raise interrupt during free-fall event, single-tap (acceleration spike) event, and orientation change event • Module should be able calculate bracelet's acceleration within +-2g range on 3 axis	5
Monitoring Display Module	• Module should display monitoring device User Interface with no glitches with bracelet status and location • User should be able to interact with touch-screen button to send out alert to bracelet	5

Bracelet HMI Module (Help Button+Buzzer)	• Button should be debounced and raise signal to send out ‘Help Alert’ to monitoring device • Buzzer should sound at hearable frequency when ‘Buzz Alert’ is sent to bracelet	5
Power Units	• Module should supply enough steady power to the circuit and its components	5
	Total	50

Appendix B: Arduino Code

Bracelet MCU Main Program

```
#include <SoftwareSerial.h>
#include "Coordinates.h"
#include "RF.h"

//#define RAWGPS true
//#define SHOWGPSPARSE true
#define RFDEMO true

SoftwareSerial gpsSerial(9,8);
RFM69 rf; //(RF69_SPI_CS, IRQPIN, false, IRQNUM);

const byte HelpButtonPin = 2;
unsigned long button_time = 0;
unsigned long last_button_time = 0;

const byte BuzzerPin = 6;

void setup() {
 Serial.begin(9600);
 while(!Serial);
 gpsSerial.begin(9600);
 while(!gpsSerial);

//  pinMode(HelpButtonPin, INPUT);
//  attachInterrupt(digitalPinToInterrupt(HelpButtonPin), sendHelpAlert, RISING);
//  pinMode(BuzzerPin, OUTPUT);

 rf_init(rf);

#ifdef RAWGPS
 Serial.println("Printing raw GPS Data...enter 'e' to continue to next test");
 char c = 0;
 while(c != 'e') {
  char gpsByte;
  if(gpsSerial.available()) {
   gpsByte = gpsSerial.read();
   Serial.print(gpsByte);
  }
```

```
   if(Serial.available())
     c = Serial.read();
  }
#endif

#ifdef SHOWGPSPARSE
  char c = 0;
  Serial.println("Printing parsed GPS Data...enter 'e' to continue to next test");
  Coordinates coord(gpsSerial, Serial);
  while(c != 'e') {
    coord.get_coordinates();
    Serial.println("Parsed GPS Data");
    Serial.print("Time: "); Serial.println(coord.timestamp, 5);
    Serial.print("Latitude: "); Serial.println(coord.latitude, 5);
    Serial.print("Longitude: "); Serial.println(coord.longitude, 5);

    if(Serial.available())
     c = Serial.read();
  }
#endif

#ifdef RFDEMO
  char c = 0;
  Serial.println("RF Communication Demo...enter '~' to exit");
  Serial.print("Node "); Serial.print(MYNODEID, DEC); Serial.println(" ready!");
  while(c != 126) {
    //Sending
    static char sendbuffer[62];
    static int sendlength = 0;
    if(Serial.available()) {
       c = Serial.read();

     if(c != '\r') {
       sendbuffer[sendlength] = input;
       sendlength++;
     }
     if((input == '\r') || (sendlength == 61)) {
       Serial.print("sending to node ");
       Serial.print(TONODEID, DEC);
       Serial.print(", message [");
       for (byte i = 0; i < sendlength; i++)
         Serial.print(sendbuffer[i]);
```

```
      Serial.println("]");

      if(rf.sendWithRetry(TONODEID, sendbuffer, sendlength))
        Serial.println("Message sent sucessfully!");
      else Serial.println("Message not sent");

      sendlength = 0;
     }
    }

    //Receiving
    if(rf.receiveDone()) {
     Serial.print("received from node ");
     Serial.print(rf.SENDERID, DEC);
     Serial.print(", message [");
     for (byte i = 0; i < rf.DATALEN; i++)
      Serial.print((char)rf.DATA[i]);
     if(rf.ACKRequested()) {
      rf.sendACK();
      Serial.println("ACK sent");
     }
    }
  }
#endif
}

void loop() {
///Add ifndef to init gps and RF////
#ifndef DEMOMODE
#endif
  Serial.println("Starting normal system operation...");

  Coordinates coord(gpsSerial, Serial);
  if(coord.get_coordinates()) {
    Serial.println("Sending GPS Data to Monitoring Device: "); Serial.print(coord.timestamp, 5);
Serial.print(",");
    Serial.print(coord.latitude, 5); Serial.print(",");
    Serial.print(coord.longitude, 5); Serial.println();
    float gps_data[] = {coord.timestamp, coord.latitude, coord.longitude};
    if(rf_send(rf, gps_data, 0, 0))
     Serial.println("GPS Data sent successfully!");
    else Serial.println("ERROR: Data not sent");
  }
```

```
}

//Help Button Alert
void sendHelpAlert() {
  float gps_data[3];
  button_time = millis();
  if(button_time - last_button_time > 50) {
    Serial.println("HELP ALERT TRIGGERED!");
    last_button_time = button_time;
  }
}

//Buzzer Alert
void buzzer() {
  for(int i = 0; i < 5; i++) {
    tone(BuzzerPin, 2000);
    delay(1000);
    noTone(BuzzerPin);
    delay(1000);
  }
}
```

Monitoring Device MCU Main Program (No Screen attached)

```
#include <SoftwareSerial.h>
#include <Nextion.h>
#include "Coordinates.h"
#include <RFM69.h>
#include <SPI.h>

#define IRQPIN 2
#define IRQNUM 0
#define NETWORKID 0
#define MYNODEID  2
#define TONODEID  1
#define FREQ      RF69_915MHZ
#define KEY       "SAMSETHSAMEEUIUC"

//#define RAWGPS true
//#define SHOWGPSPARSE true
//#define RFDEMO true
```

```
typedef struct {
  char message_type = 'G'; //G = GPS data, H = Help Alert, B = Buzzer Alert
  float gps_timestamp;
  float gps_latitude;
  float gps_longitude;
} RFPayload;

SoftwareSerial gpsSerial(9,8);
RFM69 rf; //(RF69_SPI_CS, IRQPIN, false, IRQNUM);

void setup() {
  Serial.begin(9600);
  while(!Serial);
  gpsSerial.begin(9600);
  while(!gpsSerial);

  //rf_init(rf);
  rf.setCS(10);
  rf.initialize(FREQ, MYNODEID, NETWORKID);
  rf.setHighPower();
  rf.encrypt(KEY);

#ifdef RAWGPS
  Serial.println("Printing raw GPS Data...enter 'e' to continue to next test");
  char c = 0;
  while(c != 'e') {
    char gpsByte;
    if(gpsSerial.available()) {
      gpsByte = gpsSerial.read();
      Serial.print(gpsByte);
    }

    if(Serial.available())
      c = Serial.read();
  }
#endif
#ifdef SHOWGPSPARSE
  Serial.println("Printing parsed GPS Data...enter 'e' to continue to next test");
  char c = 0;
```

```
  Coordinates coord(gpsSerial, Serial);
  while(c != 'e') {
    coord.get_coordinates();
    Serial.println("Parsed GPS Data");
    Serial.print("Time: "); Serial.println(coord.timestamp, 5);
    Serial.print("Latitude: "); Serial.println(coord.latitude, 5);
    Serial.print("Longitude: "); Serial.println(coord.longitude, 5);

    if(Serial.available())
      c = Serial.read();
  }
#endif
#ifdef RFDEMO

  Serial.println("RF Communication Demo...press '~' to exit");
  Serial.print("Node "); Serial.print(MYNODEID, DEC); Serial.println(" ready!");
  //char c = 0;
  while(1) {
    //Sending
    static char sendbuffer[62];
    static int sendlength = 0;
    if(Serial.available()) {
       char c = Serial.read();

      if(c != '\r') {
        sendbuffer[sendlength] = c;
        sendlength++;
      }
      if((c == '\r') || (sendlength == 61)) {
        Serial.print("sending to node ");
        Serial.print(TONODEID, DEC);
        Serial.print(", message [");
        for (byte i = 0; i < sendlength; i++)
          Serial.print(sendbuffer[i]);
        Serial.println("]");

        if(rf.sendWithRetry(TONODEID, sendbuffer, sendlength))
          Serial.println("Message sent sucessfully!");
        else Serial.println("Message not sent");
```

```
      sendlength = 0;
    }
   }

   //Receiving
   if(rf.receiveDone()) {
    Serial.print("received from node 1");
    Serial.print(", message [");
    for (byte i = 0; i < rf.DATALEN; i++)
     Serial.print((char)rf.DATA[i]);
    Serial.println("]");

    if(rf.ACKRequested()) {
     rf.sendACK();
     Serial.println("ACK sent");
    }
   }
  }
#endif

  Serial.println("Starting normal system operation...");
}

void loop() {
#ifndef DEMOMODE
#endif
  static RFPayload data;
  char c;
  if(Serial.available()) {
   c = Serial.read();
   if(c == 'B') {
    Serial.println("BUZZER ALERT TRIGGERED!");
    data.message_type = 'B';
    if(rf.sendWithRetry(TONODEID, (const void*)(&data), sizeof(data), 10, 500))
     Serial.println("Alert sent successfully!");
    else Serial.println("ERROR: Alert not sent");
   }
  }

  Coordinates coord(gpsSerial, Serial);
```

```
  if(coord.get_coordinates()) {
    Serial.println("----------");
    Serial.println("*Monitoring Device GPS Data*");
    Serial.print("Timestamp: "); Serial.println(coord.timestamp, 3);
    Serial.print("Latitude: "); Serial.println(coord.latitude, 5);
    Serial.print("Longitude: "); Serial.println(coord.longitude, 5);
  }
  else Serial.println("No satellite fix! GPS Data invalid");

  if(rf.receiveDone()) {
    Serial.println();
    data = *(RFPayload*)rf.DATA;

    if(data.message_type == 'G') {
      Serial.println("*Received Bracelet GPS Data*");
      Serial.print("Timestamp: "); Serial.println(data.gps_timestamp, 3);
      Serial.print("Latitude: "); Serial.println(data.gps_latitude, 5);
      Serial.print("Longitude: "); Serial.println(data.gps_longitude, 5);

      float d = calc_distance(coord.latitude, coord.longitude, data.gps_latitude,
data.gps_longitude);
      Serial.print("Distance: "); Serial.println(d, 3);
      int rel_loc = calc_rel_loc(coord.latitude, coord.longitude, data.gps_latitude,
data.gps_longitude, d);
    }
    if(data.message_type == 'H') {
      for(byte i = 0; i < 5; i++)
        Serial.println("HELP ALERT!");
      //Send alert to screen
    }
    if(rf.ACKRequested()) {
      rf.sendACK();
      Serial.println("ACK SENT");
    }
   }
    Serial.println("----------"); Serial.println();

}

//Distance Function
```

```
float calc_distance(float lat1, float long1, float lat2, float long2)
{
  float R = 6371000;
  //Convert to radians
  lat1 = lat1 * (M_PI/180);
  long1 = long1 * (M_PI/180);
  lat2 = lat2 * (M_PI/180);
  long2 = long2 * (M_PI/180);

  //Find deltas
  float dlat = lat2 - lat1;
  float dlong = long2 - long1;

  float d = pow(sin(dlat/2),2) + cos(lat1) * cos(lat2) * pow(sin(dlong/2),2);
  d = 2 * asin(sqrt(d));
  d = d * R;

  return d;
}

//Relative Location Function
int calc_rel_loc(float lat1, float long1, float lat2, float long2, float d) {
  float dir;
  float dlat = lat2 - lat1;
  float dlong = long2 - long1;

  if(dlat > 0 && dlong == 0)
    return 90;
  else if(dlat < 0 && dlong == 0)
    return 270;
  else if(dlat == 0 && dlong > 0)
    return 180;
  else if(dlat == 0 && dlong < 0)
    return 0;

  float dx;
  int theta;
  dx = calc_distance(lat1, long1, lat1, long2);
  theta = acos(dx / d);
  theta = theta * (180/M_PI);
```

```
  if(dlat > 0 && dlong > 0) { //NE
    int x = 90 - theta;
    Serial.print("Location: "); Serial.print(x, DEC); Serial.println("deg NE");
    theta = 180 - theta;
    return theta;
  }
  else if(dlat > 0 && dlong < 0) { //NW
    int x = 270 + theta;
    Serial.print("Location: "); Serial.print(x, DEC); Serial.println("deg NW");
    return theta;
  }
  else if(dlat < 0 && dlong > 0) { //SE
    int x = 90 + theta;
    Serial.print("Location: "); Serial.print(x, DEC); Serial.println("deg SE");
    theta = 180 + theta;

    return theta;
  }
  else if(dlat < 0 && dlong < 0) { //SW
    int x = 279 - theta;
    Serial.print("Location: "); Serial.print(x, DEC); Serial.println("deg SW");
    theta = 360-theta;
    return theta;
  }
}
```

Monitoring Device MCU Main Program (with Screen)

```
#include <SoftwareSerial.h>
#include <Nextion.h>
#include "Coordinates.h"
#include <RFM69.h>
#include <SPI.h>

#define IRQPIN 2
#define IRQNUM 0
#define NETWORKID 0
#define MYNODEID  2
```

```
#define TONODEID  1
#define FREQ      RF69_915MHZ
#define KEY       "SAMSETHSAMEEUIUC"

//#define RAWGPS true
//#define SHOWGPSPARSE true
//#define RFDEMO true

typedef struct {
  char message_type = 'G'; //G = GPS data, H = Help Alert, B = Buzzer Alert
  float gps_timestamp;
  float gps_latitude;
  float gps_longitude;
} RFPayload;

SoftwareSerial gpsSerial(9,8);
RFM69 rf; //(RF69_SPI_CS, IRQPIN, false, IRQNUM);

NexButton b0 = NexButton(0,1,"b0");
NexGauge z0 = NexGauge(0,2,"z0");
NexRadio r0 = NexRadio(0, 8, "r0");   //Help
NexRadio r1 = NexRadio(0, 9, "r1");   //Out of range
NexRadio r2 = NexRadio(0, 10, "r2");  //Fall
NexNumber n0 = NexNumber(0, 11, "n0");
NexTouch *nex_listen_list[]={&b0, NULL};

bool pressed = false;
void b0PushCallback(void *ptr) {
  pressed = !pressed;
  RFPayload data;
  if(pressed) {
    b0.setText("Buzzing");
    data.message_type = 'B';
    rf.sendWithRetry(TONODEID, (const void*)(&data), sizeof(data), 4, 500)
  }
  else b0.setText("Buzz");
}

void setup() {
  b0.attachPush(b0PushCallback);
```

```
  screenSerial.begin(9600);
  Serial.begin(9600);
  while(!Serial);
  gpsSerial.begin(9600);
  while(!gpsSerial);

  //rf_init(rf);
  rf.setCS(10);
  rf.initialize(FREQ, MYNODEID, NETWORKID);
  rf.setHighPower();
  rf.encrypt(KEY);
}

void loop() {
  nexLoop(nex_listen_list);

  static RFPayload data;
  Coordinates coord(gpsSerial, Serial);
  coord.get_coordinates()

  if(rf.receiveDone()) {
    data = *(RFPayload*)rf.DATA;

    if(data.message_type == 'G') {
      float d = calc_distance(coord.latitude, coord.longitude, data.gps_latitude,
data.gps_longitude);
      n0.setValue(dist);
      float rel_loc = calc_rel_loc(coord.latitude, coord.longitude, data.gps_latitude,
data.gps_longitude, d);
      z0.setValue(rel_loc);
    }
    if(data.message_type == 'H') {
     flash_alert(r0);
    }
    if(rf.ACKRequested())
      rf.sendACK();
  }

}
```

```
//Flash Alert Radio Function
void flash_alert(NexRadio r) {
  for(int i = 0; i < 10; i++) {
    r.setValue(1);
    delay(500);
    r.setValue(0);
    delay(500);
  }
}

//Distance Function
float calc_distance(float lat1, float long1, float lat2, float long2)
{
  float R = 6371000;
  //Convert to radians
  lat1 = lat1 * (M_PI/180);
  long1 = long1 * (M_PI/180);
  lat2 = lat2 * (M_PI/180);
  long2 = long2 * (M_PI/180);

  //Find deltas
  float dlat = lat2 - lat1;
  float dlong = long2 - long1;

  float d = pow(sin(dlat/2),2) + cos(lat1) * cos(lat2) * pow(sin(dlong/2),2);
  d = 2 * asin(sqrt(d));
  d = d * R;

  return d;
}

//Relative Location Function
int calc_rel_loc(float lat1, float long1, float lat2, float long2, float d) {
  float dir;
  float dlat = lat2 - lat1;
  float dlong = long2 - long1;

  if(dlat > 0 && dlong == 0)
    return 90;
  else if(dlat < 0 && dlong == 0)
```

```
  return 270;
 else if(dlat == 0 && dlong > 0)
  return 180;
 else if(dlat == 0 && dlong < 0)
  return 0;

 float dx;
 int theta;
 dx = calc_distance(lat1, long1, lat1, long2);
 theta = acos(dx / d);
 theta = theta * (180/M_PI);

 if(dlat > 0 && dlong > 0) { //NE
  theta = 180 - theta;
  return theta;
 }
 else if(dlat > 0 && dlong < 0) //NW
  return theta;
 else if(dlat < 0 && dlong > 0) { //SE
  theta = 180 + theta;
  return theta;
 }
 else if(dlat < 0 && dlong < 0) { //SW
  theta = 360-theta;
  return theta;
 }
}
```

RF.h Header File

```
#ifndef RF_H
#define RF_H

#include <RFM69.h>
#include <SPI.h>

#define IRQPIN 2
#define IRQNUM 0
#define NETWORKID 0
#define MYNODEID  1
```

```
#define TONODEID  2
#define FREQ     RF69_915MHZ
#define KEY      "SAMSETHSAMEEUIUC"

typedef struct {
  char message_type[2] = {0, 0};
  float gps_timestamp;
  float gps_latitude;
  float gps_longitude;
} RFPayload;

void rf_init(RFM69 rf) {
  rf.setCS(10);
  rf.initialize(FREQ, MYNODEID, NETWORKID);
  rf.setHighPower();
  rf.encrypt(KEY);
}

RFPayload rf_receive(RFM69 rf) {
  RFPayload data;
  if(rf.receiveDone() && rf.DATALEN == sizeof(RFPayload)) {
    data = *(RFPayload*)rf.DATA;
  }
  return data;
}

boolean rf_send(RFM69 rf, float gps_data[], boolean alert = 0, int alert_type = 0) { //HELP:
alert_type = 1, FALL; alert_type = 2
  RFPayload data;

  if(alert) {
    data.message_type[0] = 'A';
    if(alert_type == 1)
      data.message_type[1] = 'H';
    else if(alert_type == 2)
      data.message_type[1] = 'F';
  }

  else {
    data.message_type[0] = 'G';
    data.gps_timestamp = gps_data[0];
    data.gps_latitude = gps_data[1];
    data.gps_longitude = gps_data[2];
```

```
 }

 if(rf.sendWithRetry(TONODEID, (const void*)(&data), sizeof(data)))
  return 1;
 else return 0;
}
```

```
#endif
```

Coordinates.h Header File

```
#ifndef COORDINATES_H
#define COORDINATES_H

#include <Arduino.h>
#include <SoftwareSerial.h>
#include <math.h>

class Coordinates {

  public:
    Coordinates(SoftwareSerial &gps_ser, HardwareSerial &hw_ser);
    float latitude;
    float longitude;
    float timestamp;
    boolean get_coordinates();


  private:
    HardwareSerial *hw_stream;
    SoftwareSerial *gps_stream;
    boolean read_gps_uart(char gps_data[]);
    boolean findRMC(char gps_data[]);
    boolean parse_gps_data(char gps_data[]);
};
```

```
#endif
```

Coordinates.cpp CPP File

```
#include "Coordinates.h"

#define SHOWGPSPARSE true

Coordinates::Coordinates(SoftwareSerial &gps_ser, HardwareSerial &hw_ser) {
  gps_stream = &gps_ser;
  hw_stream = &hw_ser;
}

boolean Coordinates::get_coordinates() {
  char gps_data[128];
  boolean gps_data_valid = false;

    while(!gps_data_valid) {
      if(read_gps_uart(gps_data)) {
#ifdef SHOWGPSPARSE
  int i = 0;
  hw_stream->println("RMC GPS message: ");
  while(gps_data[i] != '\0') {
    hw_stream->print(gps_data[i]);
    i++;
  }
  hw_stream->println();
#endif
        gps_data_valid = parse_gps_data(gps_data);
        if(gps_data_valid)
          return 1;
    }
  }
}

boolean Coordinates::read_gps_uart(char gps_data[]) {
  boolean newData = false;
  static boolean recvInProgress = false;
```

```
  static byte idx = 0;
  char c;
  int i, j;
  char temp[128];

  while(gps_stream->available() > 0 && newData == false) {
    c = gps_stream->read();

    if(recvInProgress == true) {

      if(c != '\n') {
        temp[idx] = c;
        idx++;
      }

      else {
        temp[idx] = '\0';
        recvInProgress = false;
        idx = 0;
        if(findRMC(temp)) {
          newData = true;
          i = 0;
          while(temp[i] != '\0') {
            gps_data[i] = temp[i];
            i++;
          }
          for(j = i; j < 128; j++)
             gps_data[j] = '\0';
          return 1;
        }
      }
    }

    else if(c == '$')
      recvInProgress = true;
  }

  return 0;
}
```

```
boolean Coordinates::findRMC(char gps_data[]) {
  if(gps_data[2] == 'R' && gps_data[3] == 'M' && gps_data[4] == 'C')
    return true;
  else return false;
}

boolean Coordinates::parse_gps_data(char gps_data[]) {
  int i = 6;
  int j = 0;

  //Parse time
  char gps_time[11];
  while(gps_data[i] != ',') {
    gps_time[j] = gps_data[i];
    i++; j++;
  }
  while(j < 11){
    gps_time[j] = 0;
    j++;
  }
  timestamp = atof(gps_time);

  //Parse status
  char gps_status;
  i++;
  gps_status = gps_data[i];
  if(gps_status == 'A') {

    //Parse lat. coordinates
    char lat_deg[3]; lat_deg[2] = 0;
    char lat_min[9]; lat_min[8] = 0;
    i += 2; j = 0;
    while(j < 2) {
      lat_deg[j] = gps_data[i];
      i++; j++;
    }
    j = 0;
    while(j < 8) {
      lat_min[j] = gps_data[i];
      i++; j++;
```

```
    }
    latitude = atof(lat_deg) + (atof(lat_min)/60.00000);

    //Parse lat. direction
    i++;
    if(gps_data[i] == 'S')
      latitude = latitude * -1;

    //Parse long. coordinates
    char long_deg[4]; long_deg[3] = 0;
    char long_min[9]; long_min[8] = 0;
    i += 2; j = 0;
    while(j < 3) {
      long_deg[j] = gps_data[i];
      i++; j++;
    }
    j = 0;
    while(j < 8) {
      long_min[j] = gps_data[i];
      i++; j++;
    }
    longitude = atof(long_deg) + (atof(long_min)/60.00000);

    //Parse long. direction
    i++;
    if(gps_data[i] == 'W')
      longitude = longitude * -1;

    return 1;
  }

  else return 0;
}
```