Author: Samuel Nicklaus
Team Member: Luke Farmer
Team 2
Professor Beichel
ECE:3360:0001 Embeded Systems
05/03/2023
Final Report

1. Introduction

The motivation behind this project is that we wanted to have some way to measure distance with the ATMega328P. At first, we considered doing this like a range finder device but decided to go with a GPS. Using a GPS to measure distance allows the user to set specific points where a rangefinder has to be a distance to a certain point. We liked the freedom that a GPS distance device provides.

This project involves using a GPS sensor to determine the distance between two given points. We will be using a GPS, LCD display, and push buttons to make it possible for a user to go to a point, and press a button that will populate their latitude and longitude on the LCD. The user can then go to a different point, press the same button again, and the LCD will read out what distance the user is from the first point. In addition to this, the user will be able to clear their previous push with a different push button. The final part of this project is that we will be storing past points in the memory. The user will be able to see what the past points and distances are by scrolling an RPG.

2. Implementation

a. Overview

The big picture of this project is that we needed a way to get a user's location, display that location to the user, and then get a location at a second point and calculate the distance between the two. To accomplish getting the location data, we used a GT-7U GPS module. We used a LCD Display to show the user relevant data. Finally, we used push buttons and an RPG to allow the user to operate the device.

Our design process started with discovering how to obtain data from the GPS. After accomplishing that, we worked on getting our LCD to display strings. After the LCD worked, we went back to the GPS and worked on getting the data into a form we could display to the LCD. Around the same time, we implemented functions for the push buttons to perform and tested that they were working correctly by printing out test data to the LCD.

b. Hardware Description

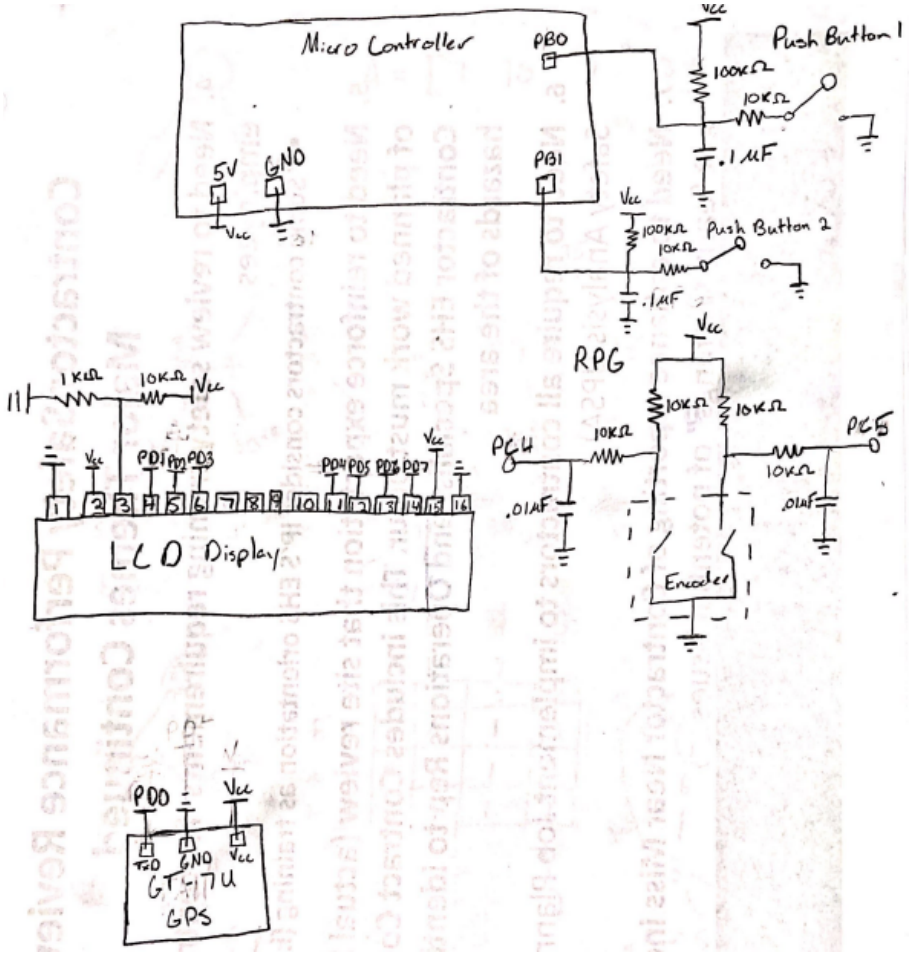

Figure 1: Circuit Schematic. The IC attached to the 7-segment display is an SNx4HC595 8-Bit Shift Register

We chose the GT-7U GPS module because it was affordable and used USART we knew we could interface. The LCD, push buttons, and RPG components were all the same components we used in class, so we chose those because of familiarity. All datasheets that we used throughout the project are referenced in the Appendix. We implemented both a hardware debounce for our push buttons and both software and hardware debounce for the RPG. For the push buttons and RPG, we have capacitors for hardware debounce and activated the pull-up resistors. On the software side, when the RPG is turned, the code infinite loops at portions until the necessary sequence takes place to show that a full turn has happened. All of our resistor values were chosen based on diagrams given to us in slides or in diagrams.

c. Software Description

All the software was written in the C programming language and the functions can be split up into 5 separate categories: USART, EEPROM, Interrupts, Helper functions, and the main function. The main function is where we pull all of our GPS data. In an infinite loop, we constantly check for incoming data and save it if there is valid GPS data. The

USART functions are there to assist with receiving the GPS data through the RX port of the ATMEGA328P while the EEPROM functions assist with saving, deleting, and overwriting existing distance data saved in the non-volatile memory of the ATMEGA. We also have two interrupt functions, one for the RPG and one for the two push buttons. On either button press the button interrupt function is called where it checks to see what button was pressed and then acts accordingly. If the left button was pressed, the latitude and longitude at the given time are saved into memory. If the user decides to press that button again it will compare the points of when the user pressed the button and give the distance between the two while also saving that data in non-volatile memory. If the user stores more than ten data points in memory the 11th data point will overwrite the first one and so on. If the user presses the right button at any given point all stored distance data in the ATMEGA (both volatile and non-volatile) will be cleared. When the RPG is turned, the interrupt on PCINT13 is triggered. We only want to check RPG turns when PC5 goes low, so that's the first thing we check for. If PC5 is low, we then check to see if PC4 is high or low. If it's high, we know the RPG is turning counterclockwise. We then have while loops in place that block the code until the next sequence in the RPG turn occurs. The counterclockwise check is the mirror of the clockwise. During the sequence of turns, we do have a check if the RPG starts to turn the wrong way. If that occurs, we set a variable, full turn, equal to 0. At the end of the turn sequence, we check if the full turn is equal to 1. If it is, it's a valid turn, and behave as such. If the full turn variable is equal to 0, we don't perform the turn actions. Once we have determined which way the RPG is turning we increment/decrement our EEPROM memory counter and read the data at that given memory address. Once we have this data, we display it to the user along with the current memory location they are in. Finally, we have the miscellaneous helper functions that are mostly used to help us calculate the distance between coordinates. These math formulas were already created for us we just had to code them in C with the help of the built-in Math library.

3. Experimental Methods

As we briefly discussed in our overview, our experimental methods were based on breaking up our large projects into smaller projects. We started by confirming our GPS worked. Most of the documentation for the GPS module dealt with the Arduino IDE. Knowing that we tested the GPS using the Arduino IDE to make sure we could get valid values. Once we solved that problem, we moved on to getting our LCD set up and displaying strings. To do this, we found a C library, set up our connections as it laid them out, and were able to get our LCD to display strings. After accomplishing this, we went back to using the GPS and getting data from the GPS to the actual microcontroller which we accomplished with USART functions. To confirm this problem was solved, we printed values out to the LCD to see if we were reading them incorrectly. We encountered an error in our methods here because our readings weren't in the right decimal form. We learned that we had to change the strings that we were reading into the form we wanted to display to the LCD display. While doing this, we had to write the string into float form to use in our future calculations. At the same time, we were working on getting interrupts working for our RPG and push buttons. As we did this, we tested if the interrupts by working by printing values to the screen as a check. Once we had the ability to get points from the GPS, print to the LCD

display, use the push buttons, and turn the RPG, we moved on to tie all of the functionality together. We then set it up to get two points on back-to-back button pushes. We then had two latitude, and longitude points and need to calculate the distance between them. To do this, we had two helper functions: the haversine and deg2rad functions. The haversine function implements the haversine formula to calculate the distance in km between two points that have been captured from the GPS module. At the end of this function, we convert the value to feet and return it. As for the other helper function, It takes in a float of a degree value, multiplies it by pi / 180, and returns it for the equivalent radian value. This is used in the haversine function. After calculating the distance, we store it in EEPROM using the EEPROM library we used to store the values in the EEPROM. The last part of our process was using the RPG to move through the different EEPROM memory locations and using the other push button clear all the EEPROM memory locations.

4. Results

While we didn't have enough time in our presentation to show off all of our functionality, you can find the whole uncut video of our demo **HERE** which shows the core functionality of saving points, calculating distance, and looking through values in non-volatile memory

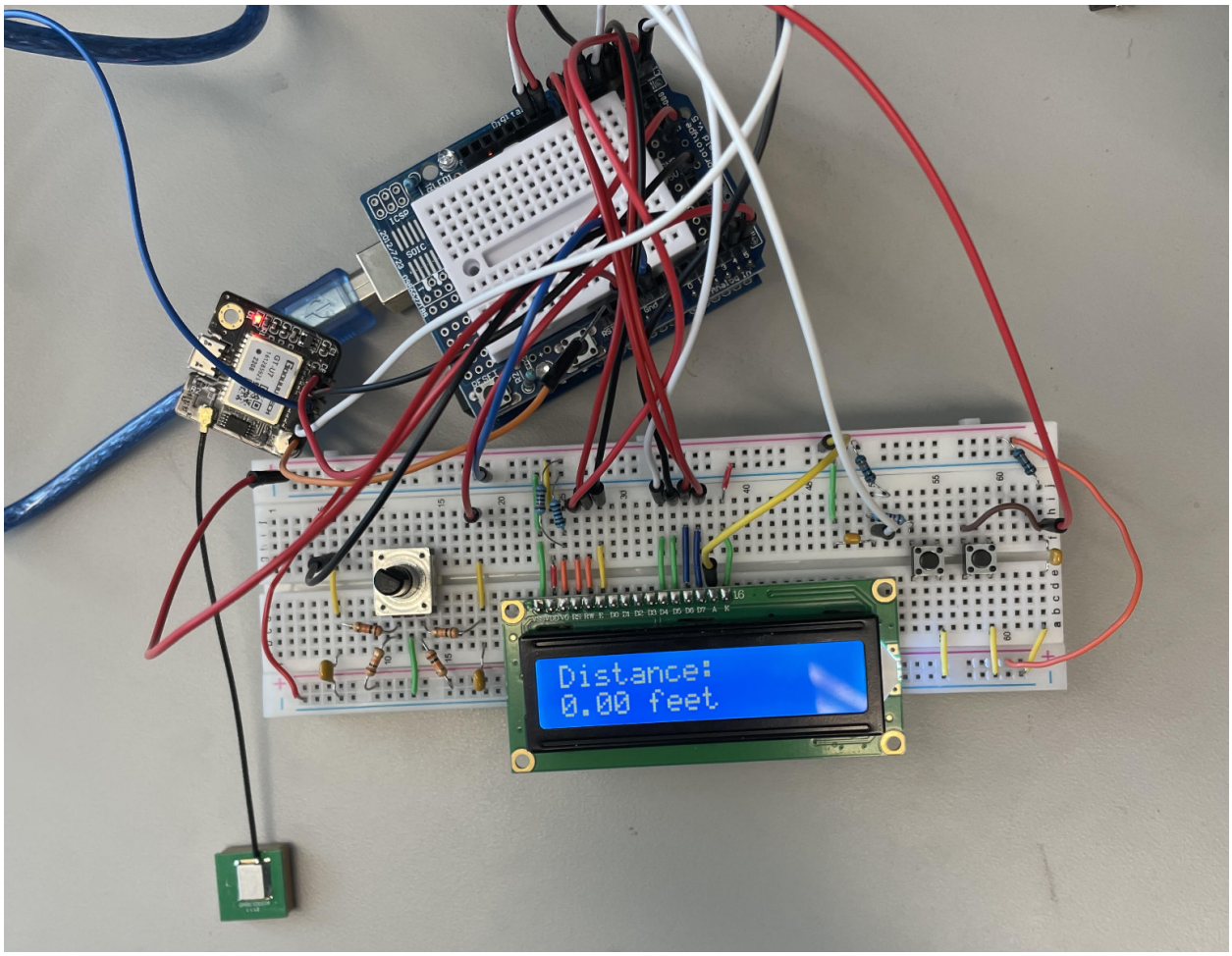

Figure 2: Our finished circuit on the breadboard

The circuit was programmed using the source code and functionality was tested by using USART to output GPS data to a laptop before displaying to our LCD display. The following items were verified:

When powered on, the LCD and GPS are initialized properly, with the LCD displaying “Searching…” after the GPS is initialized.	X
Both the RPG and Pushbutton are properly debounced	X
When the left button is pushed for the first time, the current latitude and longitude are saved and the user is notified.	X
When the left button is pushed for the second time, the distance between the user's current latitude and longitude and the previous button press is calculated and displayed to the user for five seconds	X
When the distance is calculated, the value is stored in the first available memory in EEPROM	X
When the right button is pushed, all previous user points are cleared as well as current points in volatile memory.	X
When the RPG is rotated CW or CCW stored values (if there are any) are displayed to the user to see the distance of previous points	X

EEPROM: We wrote distance data to our EEPROM. To do this, we used the write and read functions found in the AVR EEPROM library to read and write a single byte from the EEPROM. We then made another function that called the given functions multiple times to have the functionality of placing or getting a float in the EEPROM.

EEPROM Location: We keep a counter reading and writing to the EEPROM. These are separate counter-variables. The write counter is incremented as more floats are added to the EEPROM with the max being 10 stored floats. When 10 floats have been stored and the user attempts to store another, the designated slots are cleared for new entries. For reading, the read EEPROM counter is manipulated by the RPG to allow the user to rotate through all of the stored readings. the

Interrupts: We used two interrupts: PCINT13 and PCINT0. We used PCINT0 for both the push buttons, so when that interrupt is triggered, we check if push button 1 or 2 is low and then operate based on that knowledge. If push button 1 is pushed, we clear the distances stored in the EEPROM. If push button 2 is pushed, we grab the data from the GPS and store that as point 1 if it’s the first time pressed. If it’s the second push, we calculate the distance from the first point to the second using the Haversine formula and print that to the LCD display. The last interrupt is for the RPG. When that interrupt is activated, we check if PC5 is low, and if it is, we detect if it’s a counter-clockwise or clockwise rotation and perform the corresponding actions.

USART: We use USART to read data from the GPS module. First, we initialize the USART setting the Baud to our calculated rate, enabled receiver and receiver interrupt, and set the data frame format. As for using USART, we have a base function that receives one unsigned char. Then, to open up the ability to read entire strings, we made a function that repeatedly calls USART_Receive until a new line or null character is found. We then parse this data into relevant readings and convert those to floats for our latitude and longitude values that are printed on the LCD.

Baud Rate: We set our Baud rate to 9600 by using the equation in Table 20-1 in accordance with the Asynchronous Normal Mode. We used the same table for calculating our UBBR value. Those equations are BAUD = (fosc)/[16*(UBRRn+1)] and UBRRn = (fosc)/[16*BAUD] - 1 respectively.

Helper Functions: We had two helper functions: the haversine and deg2rad functions. The haversine function implements the haversine formula to calculate the distance in km between two points that have been captured from the GPS module. At the end of this function, we convert the value to feet and return it. As for the other helper function, It takes in a float of a degree value, multiplies it by pi / 180, and returns it for the equivalent radian value. This is used in the haversine function.

5. Discussion of Results

Performance: Overall, I think we are fairly satisfied with our performance, any error that we would have in our distance would be the fault of our cheaper GPS which in the spec sheet has a distance error of +- 5 meters (~16 feet). In practice though, we found that the GPS was only off by a little under a foot when measuring smaller distances (under 30 feet) and we assume that the margin of error would be similar for points at a larger distance. As always, the accuracy of our points completely depends on how many satellites our module can lock onto.

Design Goals: We achieved everything we set out to do in our original proposal, we created a device that tells the distance between two points as well as saves those results in memory for the user to view later. I think we had a good overall scope for the project as it was challenging enough without us needing to cut anything out.

Limitations/Ideas for improvement: A limitation we currently have is that the device has to be plugged in to operate. For a future iteration, I think it would be cool to design a circuit that takes up less space, as well as the ability for it to be powered via battery for long periods of time. The GPS has a really low current draw (~30mA) so creating a device to run on battery power would be feasible. Another cool improvement we could make is to upgrade the display to something that can hold more text. Currently, we have to abbreviate longer text such as memory and latitude. Having a bigger display gives us additional usability with a better laid-out user interface for the user, and potentially gives us more options such as adding a dedicated settings menu for the user and a better laid-out history view of all the points.`

6. Conclusion

In this final project, we gained further experience in using the C programming language in embedded devices and how to import and modify libraries to meet our specific needs. We also learned about how to interact with third-party components, such as our GPS module, which already has its individual software set up. Overall, this project was a culmination of everything we have learned this past semester. Implications of this product are fairly simple as currently it is a one-dimensional device where it can only take the distance between two points. The idea of the technology is cool enough that you could expand it into other things such as also track velocity data while moving, attaching an ultrasonic sensor to the device to also measure more fine distances, etc. This final project acts as a cool example of everything you can do with distance sensors.

References: Datasheets and Software Libraries

Beichel, Reinhard, "ES23_04_Assembler_I.pdf," Embedded Systems, ECE:3360. The University of Iowa, 2023

Microchip, "megaAVR® Data Sheet", 653, 2020

Microchip, "AVR Assembler", 45, 2017

Atmel, " AVR Instruction Set Manual", 191, 2016

Beichel, Reinhard, "ES23_05_RPG_Lab03.pdf" Embedded Systems, ECE:3360. The University of Iowa, 2023

GT-U7 GPS, "GT-U7 Schematic.pdf", Hobby Components, 2017

Avr-libc, "Standard C library for AVR-GCC", Link Here, 2023

Lcd.c and Lcd.h library components, "LCD 16x2 8-bit data interface", Ovidiu Sabau, 2022

Glenn Baddeley," GPS - NMEA sentence information", Link Here, 2001

Beichel, Reinhard, "ES23_C_Programming.pdf" Embedded Systems, ECE:3360. The University of Iowa, 2023

Beichel, Reinhard, "ES23_EEPROM.pdf" Embedded Systems, ECE:3360. The University of Iowa, 2023

Beichel, Reinhard, "ES23_Projects.pdf" Embedded Systems, ECE:3360. The University of Iowa, 2023

Beichel, Reinhard, "ES23_SPI_I2C.pdf" Embedded Systems, ECE:3360. The University of Iowa, 2023

<avr/io.h>	AVR device-specific IO definitions
<util/delay.h>	Convenience functions for busy-wait delay loops
<avr/interrupt.h>	Used for global manipulation of interrupt flags
<avr/eeprom.h>	Used to write to and read from EEPROM
<stdio.h>	Defines three variable types, several macros, and various functions for performing input and output.
<stdlib.h>	Defines four variable types, several macros, and] functions for performing general functions.
<math.h>	Needed for haversine formula with cosine

<string.h>	Allowed us to use string-type variables
"LCD.h"	Used for initializing and writing to LCD