Train Detection and Warning System Documentation

By: Karlie D'Apuzzo and Raphael Jara

Weber State University
April 23, 2024

Contents

1 Introduction

Train crossings are a hazard that many drivers must deal with. If there are train crossings in large cities or towns, they are usually accompanied with lights and crossing gates to warn and prevent drivers from colliding with trains. The railroads might even be completely separated from roads. On the other hand, rural areas often lack these same safety precautions. Rural areas often lack the resources to implement gates or even lights. Gates may be excessive for a rural area with little traffic, but adding lights that help warn drivers has been shown to improve the safety of crossing. The system this document describes is a system that detects the magnetic field of a train and wirelessly communicates and activates lights to warn drivers of an oncoming train.

2 Scope

This document explains the hardware design, software design, operation, and overall implementation of the detection and warning system design.

3 Design Overview

This section discusses the project requirements, theory of operation, and other design alternatives.

3.1 Requirements

1. The system shall be capable of being solar-powered
2. The system shall have lights that flash no less than 35 times a minute and no more than 65 times a minute, following the U.S. Department of Transportation Federal Railroad Administrations rules
3. The system shall be modular, and easily integrated into existing railroad infrastructure
4. The system should be able to reliably detect trains and warn drivers
5. If the system stops being able to detect trains, the warning lights should be activated indefinitely or until the modules can be serviced
6. If the system is not charging reliably the batteries should have enough capacity to run the lights continuosly for 1.5 days
7. The system should be designed to be withstand envioronmental conditions
8. The system should wirelessly communicate if a train is approaching
9. The system should give at least 20 seconds of warning per regulation

3.2 Dependencies

In the Dependencies section, we enumerate the essential components and resources upon which our design relies for functionality and operation.

1. **PIC24FJ128GL306-I/PT:** microcontroller responsible for interfacing with peripherals and performing calculations on recieved data
2. **K-MD7 Radar:** Used to track speed and direction of objects (trains)
3. **LIS3MDL Magnetometer Sensor:** Used to sense magnetic fields
4. **LoRa-E5-HF Wireless Transceiver:** Sends radio frequency signals for communication between modules of the system
5. **TPSM84203EAB 12V DC-3V DC Converter:** Converts 12V battery source power into a 3.3V rail voltage the PIC24 can run on.
6. **LEDs:** Indicator lights for which software mode the PIC24 is operating in
7. **Switches:** Allows for mode control of the PIC24
8. **SIB912DK-T1-GE3 Power Mosfet:** Let's the PIC24 turn lights on and off without pulling 1A of current through the microcontroller
9. **Batteries:** This device is meant to be battery powered, so will require some kind of DC power source, such as a battery.

3.3 Operation

This section discusses a brief overview of the operational modes of the Train Detection and Warning System.

1. **Main Mode:** When in this mode the module operates as the light controller.
2. **Detector Mode 1:** When in this mode the module operates as the detection module and sends a signal to the light controller module if a train was detected.
3. **Detector Mode 2:** This mode operates the same as Detector Mode 1, however it sends a different address when communicating to the light controller module.
4. **LoRa Receiver Test:** When in this mode the module sets the LoRa to act as a reciever and reads any messages it receives.
5. **Radar Test:** When in this mode the module reads the data from the radar.
6. **Magnetometer Test:** When in this mode the module reads the data from the magnetometer.
7. **LoRa Transmit Test:** When in this mode the module configures the LoRa to act as a transmitter and sends a test message at a set frequency.
8. **Light Test:** When in this mode the module runs the lights on a timer.
9. **Magnetometer Detection Test:** When in this mode the module reads the magnetometer data and activates an LED if the magnetic field is above a certain threshold.
10. **Other Modes:** The number of switches makes it possible to implement other modes besides the ones presented here.

3.3.1 Mode Selection

The onboard switches are used in the selection of the mode. To select a mode the switches must be changed and then the board must be reset, either by power on reset or by reset button. The routine that sets the mode only runs once per reset. This way, the mode cannot be switched mid operation.

There is also some initial peripheral setup that is done upon reset. This includes any system initialization such as clocks, pin setup, communication setup, etc. The only peripheral that gets initialized at reset is the magnetometer. It's operation is the same no matter which mode we are in. The LoRa peripheral is initialized as needed when the mode is set. Similarly, the rader is also initialized as needed.

3.3.2 Main Mode

The main mode is the mode that operates the lights. If this module is meant to be the light controller module, it should have only the magnetometer and LoRa installed as peripherals. This is because in this mode the PIC24 receives messages via LoRa and reads the magnetometer data. A more detailed discussion of this mode can be found in 6.

3.3.3 Detector Mode 1 and 2

The Detector modes are the modes that communicate to the light controller module if a train is coming. Detector mode 1 and 2 are the same except they send a different ID. This is used in case multiple detector modules are used. If this module is meant to be the detector module, it should only have the radar and LoRa installed as peripherals. This is because in this mode the module only reads from the radar and sends a message via LoRa. A more in depth discussion of this mode can be found in Section 6.

3.4 Design Alternatives

The safest way for trains and cars to cross each other, would be for them to not cross each other. This is called grade separation. However, in rural areas grade separation is almost always unnecessary and incredibly expensive. The next safest option would be to have a light with gates system, where the gates prevent drivers from crossing the tracks. Again, this may not be cost-effective for rural areas, as they have little traffic with which to actively block. A step lower and they system would just have active warning lights. The trend is that rural areas have little traffic at railroad crossings that anything more than a sign is not cost-effective.

Most rural areas simply go with the cheapest minimum required warning system, which is a passive sign telling drivers of the railroad crossing. For most cases a simple sign is adequate.

Some might argue that a passive sign is better since drivers will be more likely to stop and check. Whereas if it was an active warning sign, drivers might be more inclined to only rely on the active system. Relying purely on the active system is dangerous since the active system could possibly fail. Of course, the decision of system to choose relies on several factors including the population of drivers that will be utilizing the system.

Besides determining if an active warning system is necessary, there is also a standard way that active warning systems are implemented. This is usually done by using track circuits. Where the track itself acts as the wire and as the train comes into the circuit, the wheels and axles act as a shunt. Engineers exploit this behavior to know when a train is on a section of track.

4 Design Details

4.1 PCB Components

The main brain of this project is the PIC24FJ128GL306. This microcontroller communicates with the peripherals. This system is also designed so that not every PCB needs every component. It also only has one whole program that runs differently based off of the input switches. This allows for all the microcontrollers to be programmed with one program. The program will run differently based on the input switches.

4.1.1 PIC24FJ128GL306

The PIC24 is the main brains of the system. It is used to read and calculate data, and determine what to do based on that data. This project has a PCB designed around the PIC24. The PIC24FJ128GL306 has 64 pins. This project only uses a small number of them, however for debug purposes all pins (including the ones used for peripherals and power connections) were brought out to headers. This turned out to work well since the PCB design did have some issues where other pins needed to be used/debugged.

The PIC24 contains several useful features that were utilized for this project. It is a 16-bit microcontroller, with 128 kilobytes of program memory, 5 16-bit timers, 2 variable width SPI modules, and 4 UART modules. It has many other features but these were the biggest contributors to why it was chosen. Microchip, the makers of the PIC microcontrollers, also have plentiful documentation as well as an IDE for their chips. The IDE turned out to be very convenient for developing the software for this project.

4.1.2 LIS3MDL Magnetometer

This project utilizes a magnetometer. A magnetometer has the capability of detecting the magnetic field strength around itself. The magnetometer in this project is the LIS3MDL. It was picked because it has various sensitivity settings, the most important being the ± 4 Gauss setting. With 16 bits of resolution for the readings, we get 0.12207 mG/bit. This works really well for measure earth's magnetic field which can be anywhere from 250 mG to 650 mG depending on location.

Another useful feature of this device is that it communicates via 8-bit SPI. This works well with the PIC24 since it has SPI capability.

4.1.3 K-MD7 radar

A radar is utilized in this project to detect approaching trains. The K-MD7 was chosen for this task because it can detect moving objects up to 120 mph. The maximum speed at which trains can be traveling and on a track that intersects roads is 120mph. So this radar fits the purpose. However, the expected speed of trains is a lot slower than 120 mph, with the average speed in rural areas being 40-80 mph. This device communicates using UART.

The K-MD7 is a 24GHz radar that utilizes the doppler effect and implements the FFT to detect the speed and direction of objects in it's view. It can also detect multiple objects at once, however for this project that feature was not utilized.

4.1.4 WIO-E5 MINI DEV BRD STM32WLE5JC: LoRa Module

The LoRa module is used for long-range communications between the detection module and the warning module in the event a train is coming toward the railroad crossing. The LoRa module sends information

in the form of RF signals.

4.1.5 SIB912DK-T1-GE3: Power MOSFET Package

For the warning lights to operate while adhering to the standards set by the FRA, they need to have a current of 1 amp flow through them. The microcontroller of the warning module is what operates the flow of current in the light through the opening and closing of the gate of our power MOSFET.

The drain of the MOSFET is connected to the negative terminal of the light and the source to the ground of the system. Each light needs its own MOSFET because they're to blink alternatively, not simultaneously. However, the package contains 2 power MOSFETs so the system only needs one package to operate.

The MOSFETs take a max V_{DS} of 20V and a continuous drain current of 1.5A which works well with our 12V battery and 1A@12V lights. The MOSFETs have a turn-on voltage of less than 1V which means we can use a GPIO pin of the microcontroller to turn on the gate. The gates do have to have a resistor of $\approx$ 1kΩ connecting them to the ground or else the capacitance of the board can cause the lights to stay on.

4.1.6 TPSM84203EAB: 12VDC to 3.3VDC Converter

The whole system operates on a 12V sealed lead acid batter but for it to be usable for the microcontroller and other components on the board, it needs to step down to 3.3V. The DCtoDC converter takes a voltage in the range of 4.5V to 28V and converts it to a 3.3V output voltage that is used by many of the components on the board.

4.1.7 1729128: Screw Terminals

To connect to the input 12V battery of the system and to the lights, screw terminals were used. Consider using bigger ones, some wires took more energy to get into the terminals than others.

4.1.8 EVERRAY 12" LED Crossing Signal: Crossing Lights

To warn commuters at the crossing, LED lights flashing alternatively will be used to signal a train coming towards the crossing. The lights need to operate at 85% or more of the lights rated max voltage to adhear to standards set by the FRA. The max for these lights is 14V which means we can drive the lights with a 12V battery and meet that requirement. The lights must also flash more than 35 but less than 65 times a minute which is done in software with a timer and a toggle of gate pins of the power MOSFET. The lights draw a current of 1A@12V.

4.1.9 A6FR-4101: Switches

Since we have two different modules, we wanted to program them with their own different code but it would be tedious to keep track of two different code files. Instead, we can program both with the both the detection and warning module code but use an external peripheral to tell it what to operate as. With the switches, we differentiate the two modules, as well as incorporate test modes for testing other components and sensors like the radar and magnetometer.

The lights also drive an LED to better see what mode it is in. The microcontroller looks at the voltages at the switches to see if it is off and floating the rail voltage or if the switch is open, it will read as ground.

4.1.10 PH1RB-08-UA: PICKit Pin Header

To program the microcontroller, we'll need to connect the debugger and programmer to the required pins. Though the programmer doesn't need to connect all of its pins, we use a pin header that'll connect to the entire thing so as to not cause unnecessary torque damage to the programmer.

4.2 PCB Design

4.2.1 PIC24FJ128GL306

The microcontroller was placed in the center of the PCB to be easily accessible to connect to other components. All of the microcontroller's pins were routed up to headers for debugging. The decoupling capacitors to the rail voltages and ground were placed in the back of the PCB because we wanted the capacitors to be as close to the pins but unfortunately, the front was too crowded from the header traces.

4.2.2 K-MD7-RFB-00H-02

The radar didn't have to be in a special position but because of the orientation of the microcontroller, the radar was placed on the side closest to RD4 and RD5 which were the UART communication lines to the radar. Most of the pins were left floating but power needs to get to the radar by connecting the input power voltage to the output voltage of the DC to DC converter.

4.2.3 WIO-E5 MINI DEV BRD STM32WLE5JC

The LoRa does not come with pins and must be soldered to pin headers to solder to the board which makes it easy to remove off the board if needed by cutting off the unessential pin headers. the only connections made were the UART comlines to PIC24 pins RD11 and RD10 and the power pins supplied by voltage from the DC to DC converter.

4.2.4 A6FR-4101

Switches were connected to LEDs with pull up 200Ω resistors to the 3.3V rail and the other side of the switches where connected to ground to allow current flow through the LED when closed which also asserts 0 on the GPIO pin of the microcontroller to tell it what mode we want the system to operate in.

4.2.5 SIB912DK-T1-GE3

The power MOSFET package was placed to be as close to the screw terminals of the lights as possible because we don't want high current traces going all over the board. The pads of the package are mirrored so there are multiple ways to connect the package for it to work.

The package is very small, so to prevent overheating, the drain of each power MOSFET should be a copper plane to act like a heat sink. The planes should not connect and only need to be $0.1875mm^2$. Soldering of this device should be treated with care so as to not lose it. The steps of soldering the package take the order of **1**. Apply very little solder paste to the pad as to not bridge. **2**. Heat up the solder with a heat gun to tin the pads. **3**. Cover the pads with a moderate amount of solder flux and place the power MOSFET on top. **4**. With very low airflow, heat up with the heat gun once more as vertically as possible. Once the power MOSFET self aligns, test whether the gate pins RE0 and RE1, ground, or either negative terminal of the lights are shorted together if not, test the board. If so, remove the package, clean the pads with solder flux and wick, and try again.

4.2.6 PH1RB-08-UA

The 8-pin horizontal header is used to program the microcontroller and must be connected to the $\overline{MCLR}$, VDD, GND, a data pin, and a clk pin. The PIC24 has a couple of pairs of data pins and clk pins but the bottom left pair were used to allow for the pin header to extend over the board for easy connection to the PICKit 5.

4.2.7 Issues with PCB

The board was designed probably slightly rushed which caused design issues with the board. One issue included the magnetometer which had one of its com traces right under it which caused the magnetometer to read poor values because the fast switching trace caused its own magnetic field that was visible to the magnetometer. The solution to this was to solder the magnetometer to a perf board and connect through the headers of the microcontroller.

The screw terminal of the input load voltage of the PCB operates as intended but the ones for the lights are not so. The terminals are essentially connected in parallel to the power MOSFET which means if the lights are connected to them as designated on the board, the lights will always be on. To fix this, we'll need to connect them in series again. To do this, we connected the anode of both lights to the 12V battery through the charge controller of the system and each cathode to the terminal marked with the "+" of each screw block. The lights this way are in series with the power MOSFETs and can have a controlled current through them. The entire schematic of the circuit can be seen in Figure 1

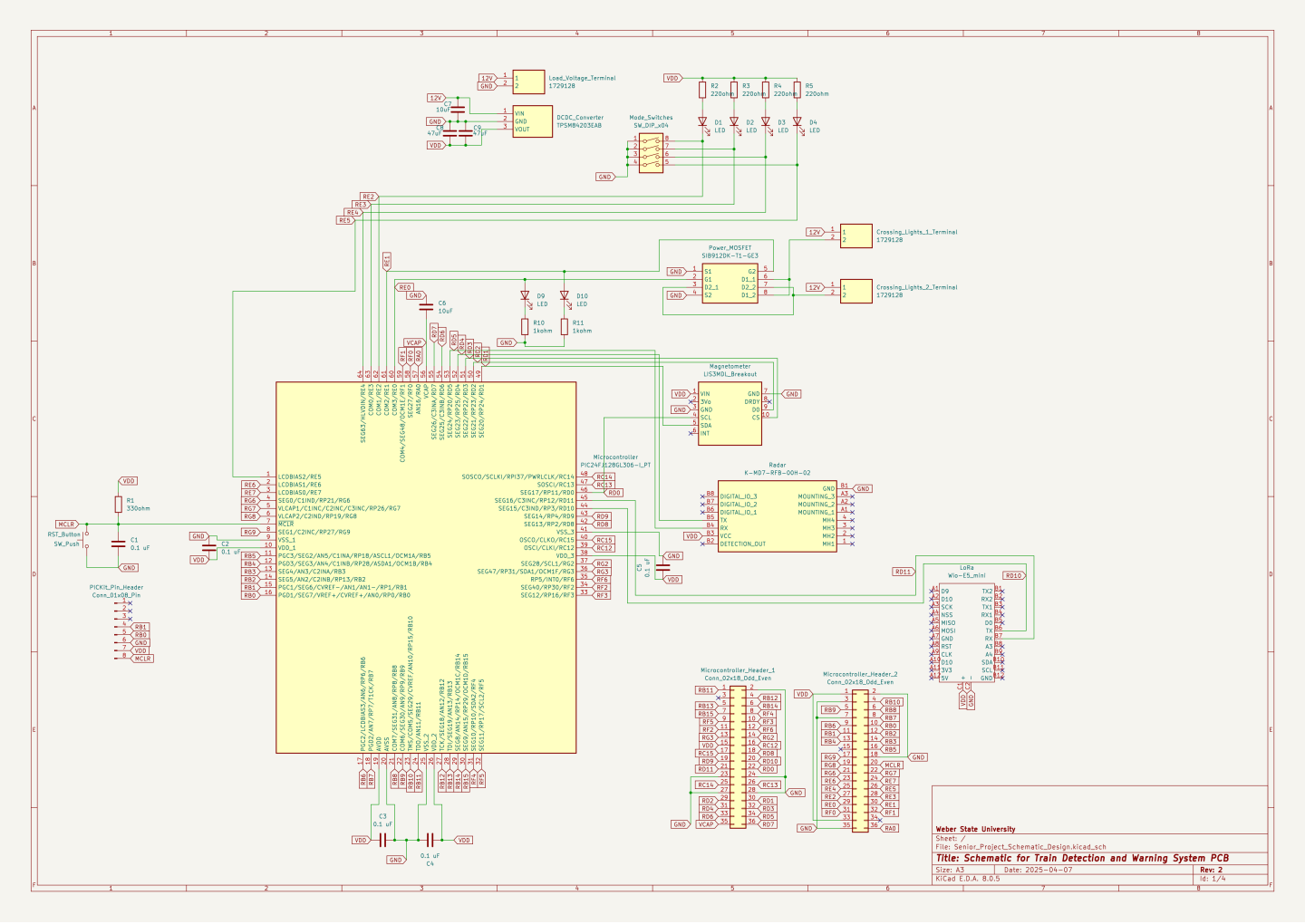

Figure 1: Schematic of entire PCB.

5 Physical Implementation

5.1 Enclosure

We initially chose a NEMA-rated enclosure for the project but eventually had to pivot to a cheaper alternative after making changes to the BOM for a different radar module. The enclosure used for the project was an ABS plastic electrical junction box that was advertised for indoor and outdoor mounting. The enclosure was ventilated, had water proof cabling grommets designed into the box, and most importantly was made of materials that were invisible to the radar.

The box had a plastic mesh that was used to hide the wires of the lights under and used to mount the PCB and the solar charge controller. The holes were too big for the M2 screw holes of the PCB to mount to so the PCB and charge controller were mounted to a piece of non conductive material that was then mounted to the mesh of the box. The dimensions of these holes are talked in the Mounting section.

5.2 Wiring

The wires of the lights were pretty short when they got to us so extensions were made by using solder crimp connectors to connect them to longer wires so that they could get into the enclosure. Connections were then covered over by heat shrink. The anode was extended using 16AWG wire and the cathode with 14AWG wire. The tips of the cathode were tinned to allow ease of connection to the board.

The only things that need to connect to the 12V battery were the anodes of the two lights and the power voltage of the PCB but because the PCB had issues, all of them had to get the voltage from the load of the charge controller which only had one port. An 18AWG wire that would be the wire that connected to the board's power voltage was put into one end of a solder crimp connector with the 2 anodes of the lights and the other end was a sole 16AWG wire that would connect to the charge controller. The connection with the connector was made then wrapped under some heat shrink. The wires were routed under the enclosure mesh to reduce messy wire and then routed back up when close to the screw block terminals of the board they needed to connect into.

5.3 Mounting

To mount the PCB onto the mesh of the enclosure, a 6 x 9in^2 panel of nonconductive material was used. A drill and measurement diagram is given in Figure 2. The $\frac{1}{8}$in drill hole Xs in the corners of the panel are what affixes the panel to the mesh of the enclosure. The $\frac{5}{64}$in drill holes for the PCB have the screws going into the back of the panels into $\frac{3}{8}$in tall female to female standoffs to elevate the PCB above the panel and the PCB is then screwed into the offsets. The charge controller is screwed into the panels with M3 screws and nuts on the same side as the PCB. After the panel has the PCB and the charge controller affixed onto it, it is affixed to the mesh of the enclosure with M3 screws and studs.

For demonstration purposes the enclosure that contained our system had to be put on a stand. For the enclosure to be put on the stand, a bracket had to be made. A dimension and drill diagram is given in Figure 3. The material is 2mm thick aluminum sheet. The top $\frac{7}{32}$in holes are for the M4 T-slot nuts of the 1x1in^2 T-slot rail. It should be noted that only the warning module needs these holes because the detection module does not have lights to hold. The $\frac{25}{64}$in hole in the middle is the hole for the stud of the tripod and held in place with a wingnut. The bottom two $\frac{13}{64}$in holes are used to screw M4 screws into the box's premade holes. After all holes are drilled, the entire sheet is bent at a 90° angle along the 3in mark of the 6in side.

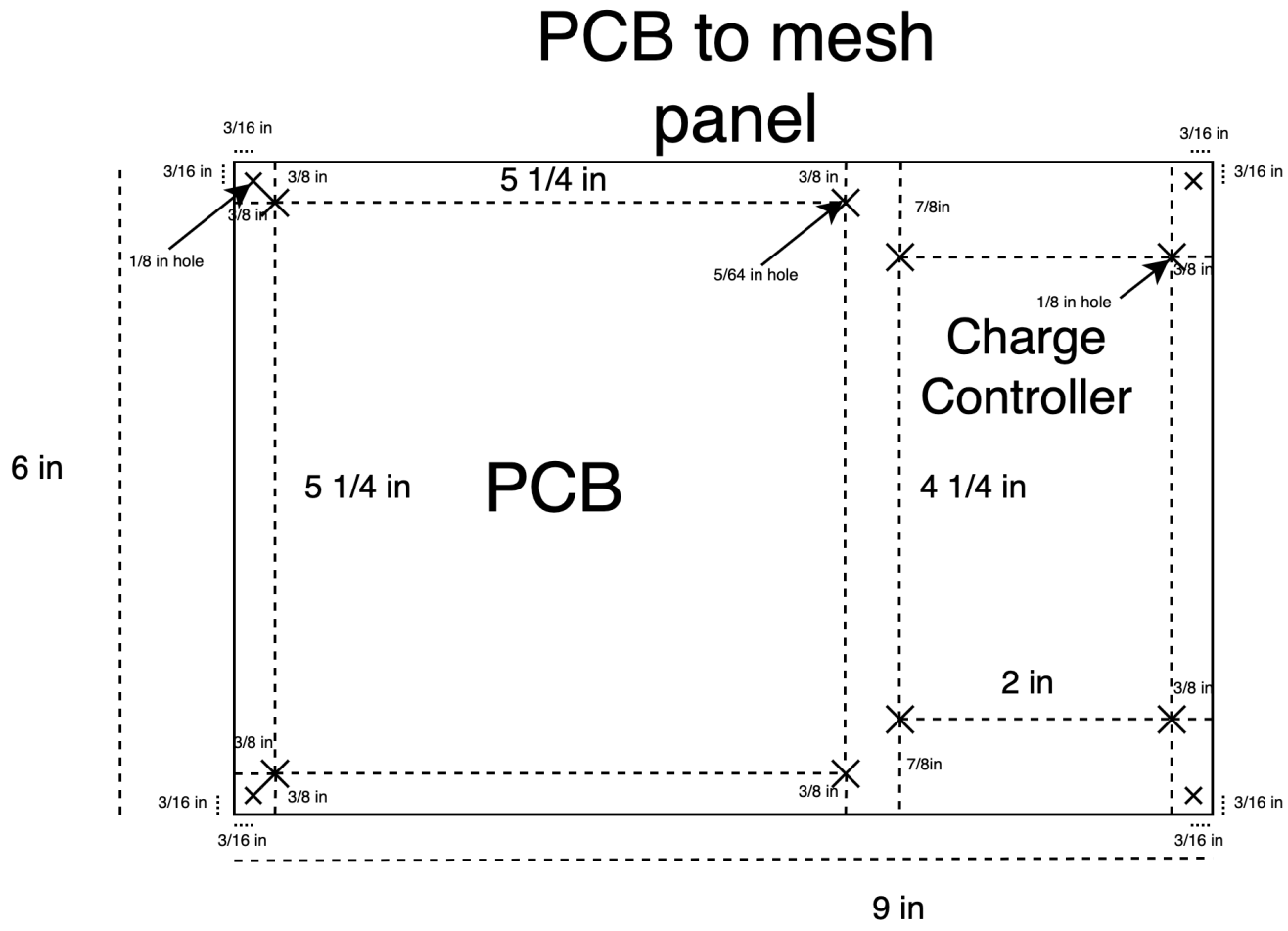

Figure 2: PCB to mesh panel measurement and drill diagram.

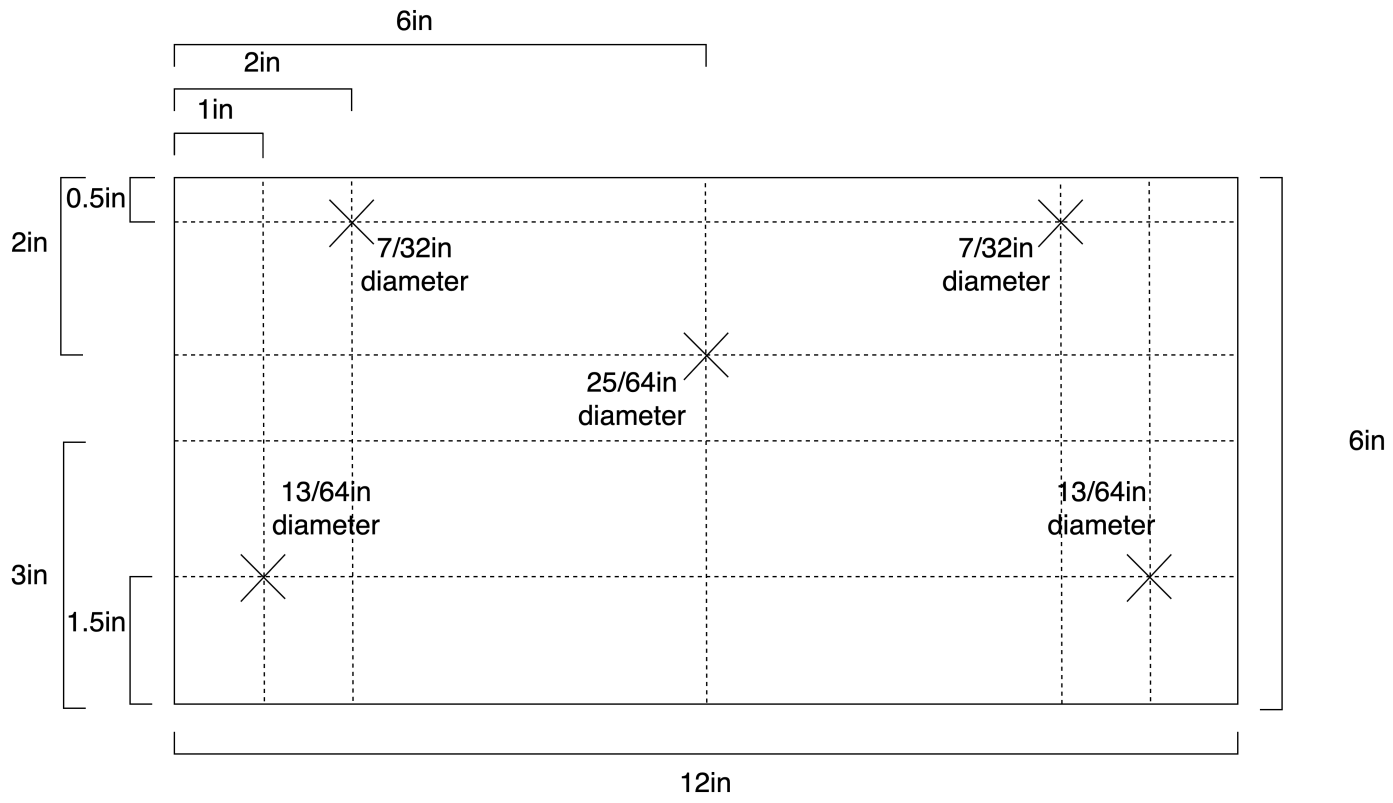

Figure 3: Bracket dimension and screw hole diagram to attach the enclosure to stud of the tripod and T-slot rail for lights.

A 3ft 1x1in^2 T-slot bar was used to hold the lights on each side of the warning system. The lights were attached to the bar by way of T-slot nuts and screws being attached to the back part of the LEDs with holes $\frac{7}{32}$in in diameter. The lights were disassembled and the back shell of the LEDs were drilled with holes to fit the M4 T-slot screw and nut. The holes were 4.5in apart and each hole was 2in from the edge of the diameter of the flat surface of the back of the LED's. A dimension and drill diagram of the lights are given in Figure 4. After the drill holes were made and the nuts and screws were fed through the back shell, the T-slot bolts were slid through the bar until the center of the back LED shells were center to the bar and the bolts were tightened to keep them in place. Once centered, the lights are ready to be reassembled with the bar attaching them. After reassembly, the T-bolts of the bracket could be fed through the rail so that the LEDs face the same direction as the opening of the enclosure and the center of the rail is centered to the wingnut of the tripod. The T-bolts of the bracket were tightened to keep them in place.

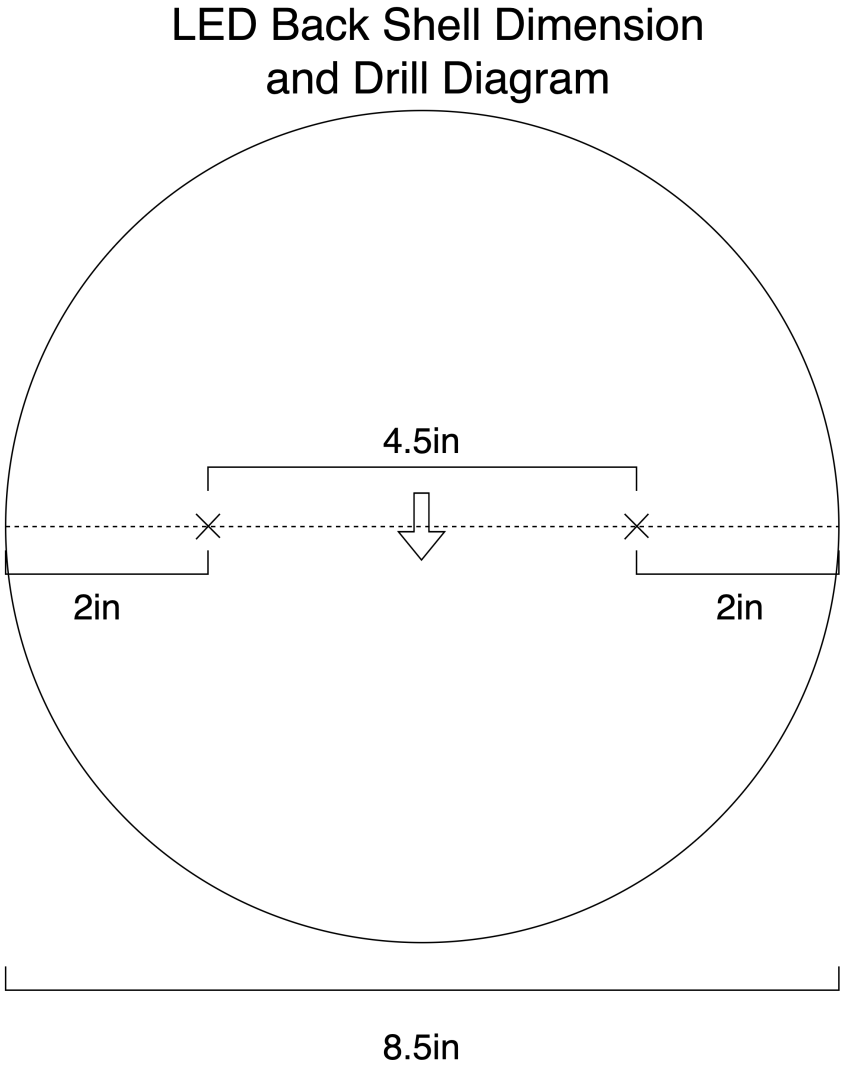

Figure 4: Dimension and drill diagram for the back shell surface of the crossing lights.

6 Software

This section describes in detail how the software is setup to interface with each component, process data, and how each main function runs.

6.1 Programming with MPLAB X IDE

6.1.1 MPLAB X IDE Introduction

To program the PIC24, the microchip IDE MPLAB X IDE was used. The IDE allows for setup, programming and debugging of the PIC24. The IDE contains many useful features that were utilized for this project. One feature that was heavily relied upon was the MPLAB Code Configurator, this tool generates code that controls the registers of the PIC24.

The PIC24 is a complicated microcontroller containing thousands of registers. MPLABs Code Configurator allows for efficient control of these registers. It generates C code that sets up registers based on user selection. This streamlines the process of programming. This useful feature allows for the developer to setup needed peripherals without needing to read through hundreds of pages of documentation. Of course, reading the documentation is incredibly important, however due to the complexity of the PIC24, the documentation is hundreds of pages long and may even reference other documents.

The MPLAB Code Configurator (MCC) can be setup for use with any of the Microchip devices. For this project the specific microchip is the PIC24FJ128GL306. With the right device selected the Code Configurator will show all the PIC24 peripherals that are available. For the PIC24 this includes several timers, serial communication modules, system modules, and more. For this project the modules that are utilized are the UART modules, the timer modules, a spi module, the system module, the interrupt module, and the pin module.

6.1.2 MPLAB X IDE Project File Structure

When starting a new MPLAB project, a base file structure is setup for the project. For this project the file structure is as shown in Figure 5

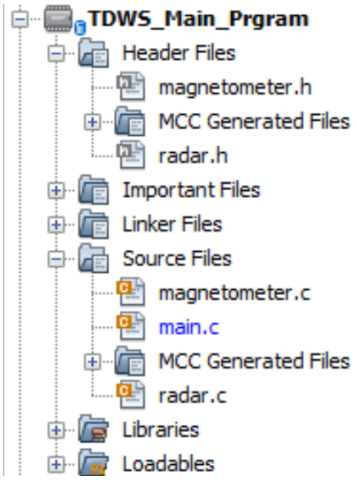

Figure 5: TDWS File Structure

In Figure 5 MPLAB generates the header files folder, important files folder, linker files folder, source files folder, libraries folder, and loadables folder. It also generates an empty main.c and respective header file.

This basic file structure is then later expanded on when using the MCC and also when writing the program for the project. The MCC Generated Files folders in the Header Files, and Source Files folders are files generated by MCC. These folders contain system initialization files as well as peripheral initialization files. An example of the files that may be in those folder is shown in Figure 6

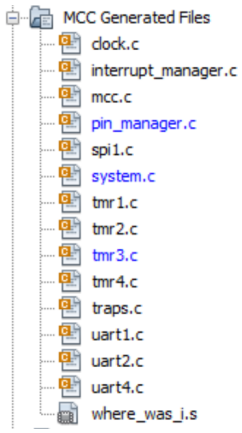

Figure 6: TDWS File Structure

The MCC Generated Files come from the MCC. To get these files a user would select the module to add, adjust the settings as necessary and generate them. The MCC then generates a C file and respective header file. This process is demonstrated in Section 6.2.

6.1.3 PIC24 MCC Modules

The PIC24 has several modules, all of which are manually programmable or programmable through the MCC. MCC streamlines the process of programming these modules by abstracting changing register values into an easy to use GUI. The modules used in this project are all programmed using the MCC.

The system module was used to adjust the system clock. The system clock is and 8 MHz clock, but a PLL can be enabled to get the oscillator frequency to be 32 MHz and then system clock is half of that at 16 MHz. This was necessary for running a higher baud rate for the radar communication UART and for the PC communication UART. The system module contains other things that were not utilized in this project.

The pin module contains all the pin definitions. MPLAB also has a very useful visual indicator of what pins are set and what they are set for. This indicator is the the Pin Manager: Grid View. When a module, like UART1 for example, it's respective inputs and outputs would show up in the Pin Manager. Another useful feature of the PIC24 is that almost any pin can be used for any peripheral. So, using the Pin Manager the pins for UART1 can be selected to not interfere with any other peripheral. The Pin Manager is shown in Figure 7. Figure 7 also shows an example of modules with output/input pins configured.

There is also a pin manager gui that allows for setting more specific settings like whether to start high, have a weak pullup or pulldown resistor, or to interrupt on change, among other things. Something that can also be done is changing the name of the pin, however all this does is change the reference name in the generated C code. This could be utilized for making code more readable, but it was not utilized for this project.

On another note about the Code Configurator, for any module selected there is also a viewer for all the individual registers that that module can effect. So if more fine tuning with the registers is needed, it can

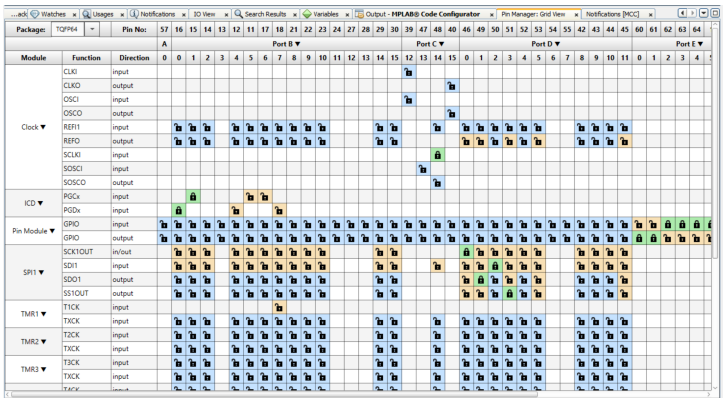

Figure 7: MPLAB X IDE Pin Manager: Grid View, showing peripherals and their respective selected pins.

be achieved through the Code Configurator. Of course the Code Configurator is not perfect and it may be nessecary to read the documentation to understand what the Code Configurator is doing.

6.2 Programming and Interfacing Peripherals

6.2.1 LISMDL Interfacing

The LIS3MDL Magnetometer communicates via SPI. To communicate with it the PIC24 SPI1 module was selected in the MCC and setup according the LIS3MDL requirements. The LIS3MDL datasheet states that it can communicate via SPI at a clock frequency of up to 10 MHz, however, for this project 800 kHz was selected for the SPI clock frequency. The MCC allows for easily configuring the PIC24 SPI module to operate for the LIS3MDL. Figure 8 shows how you would set up the SPI1 module to communicate with the LIS3MDL magnetometer. It also shows the pins that were selected for interfacing with the module. For this project, all the SPI1 pins are on Port D of the PIC24. The specific port number for each SPI line is shown in the SPI1xPort D section of the Pin Manager in Figure 8.

After setting up the SPI1 Module using the MCC, the MCC module can then generate a C file with a respective header. This C file contains the abstracted functions that can be used to interact with the SPI module. The C file is named spi1.c with a respective header. These files are located in the folder named MCC Generated Files.

Included in the generated code are the functions SPI1_Exchange16bitBuffer and SPI1_Exchange16bit. The Exchange16bitBuffer allows for the PIC24 to write to the buffer and read anything that comes back. This is used to request data from the LIS3MDL. The Exchange16bit allows for us to only write to the LIS3MDL without reading anything back.

Using the generated functions we can communicate with the LIS3MDL. The LIS3MDL requires some initial setup to be used for this project. The way the LIS3MDL is set up is in registers. The read/write protocal for communicating is shown in Figure 9. The transaction is described in the documentation shown in Figure 10.

From this code was developed which is in the magnetometer.c file and its respective header. The magnetometer.c file contains all the code to read and write to the LIS3MDL.

The LIS3MDL has registers which are addressed one at a time. Each register is 8 bits wide. The important registers for the function of this project are the status register, the control 3 register, and the high and low byte registers of the x, y, and z axis. The registers and their respective addresses are shown in Figure 11.

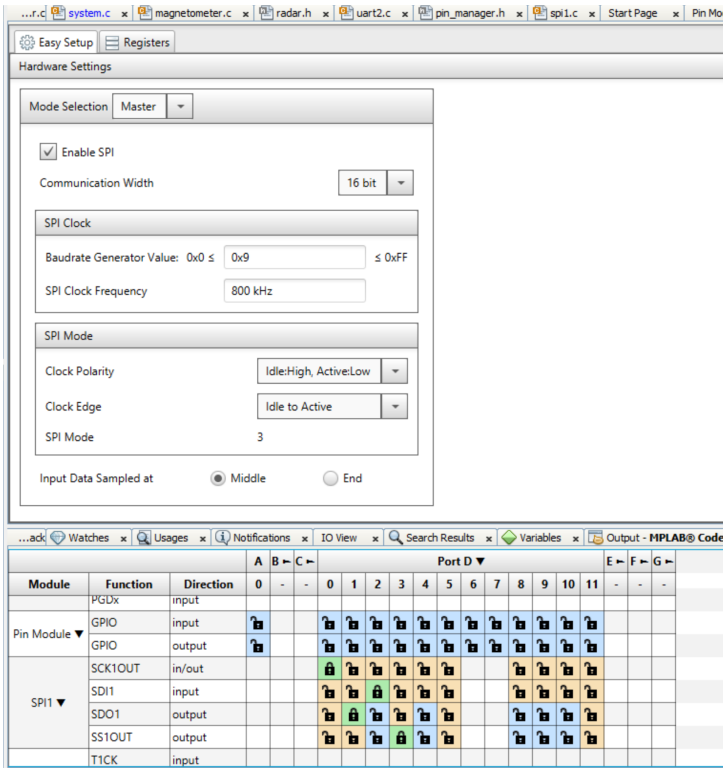

Figure 8: MPLAB X IDE MCC setup for SPI1 module to interface with LIS3MDL magnetometer module.

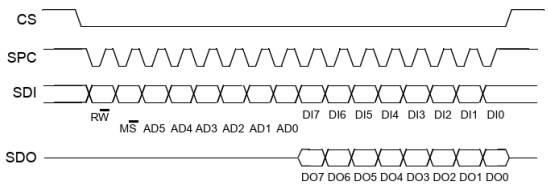

Figure 9: LIS3MDL read/write protocol from its documentation

CS enables the serial port and it is controlled by the SPI master. It goes low at the start of the transmission and goes back high at the end. **SPC** is the serial port clock and it is controlled by the SPI master. It is stopped high when **CS** is high (no transmission). **SDI** and **SDO** are respectively the serial port data input and output. Those lines are driven at the falling edge of **SPC** and should be captured at the rising edge of **SPC**.

Both the read register and write register commands are completed in 16 clock pulses or in multiples of 8 in the case of multiple byte read/write. Bit duration is the time between two falling edges of **SPC**. The first bit (bit 0) starts at the first falling edge of **SPC** after the falling edge of **CS** while the last bit (bit 15, bit 23, ...) starts at the last falling edge of SPC just before the rising edge of **CS**.

bit 0: R$\overline{W}$ bit. When 0, the data DI(7:0) is written into the device. When 1, the data DO(7:0) from the device is read. In the latter case, the chip drives **SDO** at the start of bit 8.

bit 1: M$\overline{S}$ bit. When 0, the address remains unchanged in multiple read/write commands. When 1, the address is autoincremented in multiple read/write commands.

bit 2-7: address AD(5:0). This is the address field of the indexed register.

bit 8-15: data DI(7:0) (write mode). This is the data that is written into the device (MSb first).

bit 8-15: data DO(7:0) (read mode). This is the data that is read from the device (MSb first).

Figure 10: LIS3MDL read/write transaction description from its documentation

```
#define mag_x_regl 0x2800 //address x_data low byte in magnetometer registers
#define mag_x_regh 0x2900 //address x_data high byte in magnetometer registers
#define mag_y_regl 0x2a00 //address y_data low byte in magnetometer registers
#define mag_y_regh 0x2b00 //address y_data high byte in magnetometer registers
#define mag_z_regl 0x2c00 //address z_data low byte in magnetometer registers
#define mag_z_regh 0x2d00 //address z_data high byte in magnetometer registers

#define status_reg 0x2700 //address of status register in magnetometer
#define ctrl_3_reg 0x2200 //address of control register 3 in magnetometer

#define dummy_reg 0x0fff; //address of dummy register in magnetometer
#define read_bit 0x8000; //sets instruction to read
#define write_bit 0x0000; //sets instruction to write
```

Figure 11: LIS3MDL register addresses used in the TDWS program.

Function Name	Purpose
init_magnetometer	Initialize LIS3MDL to run in continuous conversion mode
binary_to_bcd	Converts 16 bit binary value to a 16 bit binary coded decimal value
transmit_magnitude_to_pc	Transmits the calculated magnitude of three axis over UART4 (see section **??**)
transmit_mag_to_pc	Transmits individual axis over UART4 (see section **??**)
read_xaxis	Reads the high and low byte registers of the x-axis of the LIS3MDL
read_yaxis	Reads the high and low byte registers of the y-axis of the LIS3MDL
read_zaxis	Reads the high and low byte registers of the z-axis of the LIS3MDL

Table 1: Functions in magnetometer.c of the TDWS project

In the magnetometer.c file are several functions. These functions are described in Table 1. These functions are used in the main program to read data to be analyzed.

6.2.2 Radar Interfacing

The radar data sheet describes the communication as based on a client-server model. There are two types of packets transmitted. Commands are sent from the client (microcontroller) to the server (radar) and messages are sent from the server to the client. This relationship can be seen in Figure 12. The packets are formatted like the description below in Figure 13 and sent through UART.

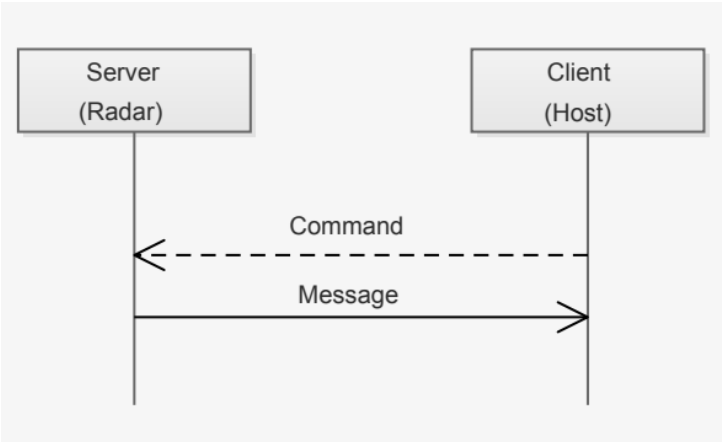

Figure 12: Diagram of radar communication to its client (the PIC24).

After each command is sent, the radar will send an acknowledgment RESP package with an error code. If the error code is anything but 0, an error happened. These error codes are described below in Figure 14.

Description	Datatype	Length
Header The header describes the command or message type (e.g. INIT, RADC, ...)	ASCII c haracter	4 Bytes
Payload Length Defines the size of the added payload. The payload length is always sent even if the payload is zero. It is sent as little endian (LSB first).	UINT32	4 Bytes
Payload The payload is message and command dependent. If the payload includes datatypes with multiple bytes (e.g. UINT16, INT32, ...) then they are sent as little endian (LSB first).	Binary data	x

Figure 13: Description of how the radar packets are formatted.

Error codes:
0=OK, no error
1=Unknown command,
2=Invalid parameter value
3=Invalid RPST version
4=Uart error (parity, framing, noise)
5=No calibration values
6=Timeout
7= Application corrupt or not programmed

Figure 14: Description of the error codes the radar can give in response to a command.

Each command is usually the 4 ASCII character header, then the payload length which is defined for each command, and the actual data you want to give to the radar. We formatted functions that would send the header and payload length part of the packet from a C array and then would send a user input value so that the person coding does not need to make an array of header characters and payload length and can just call the function with a value and will return an error code that the programmer can use to see if the command worked.

The radar is initially set to a baud rate of 115200, 8 bits, even parity, 1 stop bit, and no flow control. To talk to the radar we had to set up UART1 of the same configurations which means using MPLAB X MCC UART configurator with the RX pin as RD4 and TX as RD5. Once we have the UART configured, we can now set up the radar. The first command is the RADAR_facreset to restore factory settings. Then we send our initialization command, RADAR_init, and a baud rate setting we'd like to communicate in. In our case, we kept all communication to 115200 bits/sec. We can now set up the radar using the functions in Table 2.

We have different settings depending on what mode the switches were set on. We implemented radar human, car, and train detection modes and the radar settings for those are in Table 3.

After we set up the settings, we can start asking the radar for tracked data with our function RADAR_nexttdat where the radar will respond with a set of data parameters of up to 8 tracked targets. Our function takes a struct input of type Radar_data that stores the speed, distance, angle to the radar, magnitude, and number of tracked targets into that struct into 8 wide arrays in the form of their high and low bytes. Converting those bytes to integers with our UINT8to16() function that takes 2 unsigned 8-bit numbers and converts them to unsigned 16-bit or our UINT8toINT16() that does the same but to signed 16-bit instead, we can determine whether something is coming towards our crossing by checking if the speed is negative or receding. If nothing is tracked, the number of targets in the struct will be zero and the system

Function Name	Purpose	Input Parameter
RADAR_facreset	Restore factory settings.	-
RADAR_disconnect	Disconnect from sensor.	-
RADAR_init	Command to start a connection with a defined baud rate.	0 = 115200 1 = 460800 2 = 921600 3 = 2000000 4 = 3000000
RADAR_rangeset	Set range setting.	0 = 100m 1 = 200m 2 = 300m
RADAR_mindetzone	Change minimum detection zone distance.	0%-100% of range setting
RADAR_maxdetzone	Change maximum detection zone distance.	0%-100% of range setting
RADAR_speedset	Set speed setting.	0 = 50km/h 1 = 100km/h 2 = 200km/h
RADAR_minspeed	Set minimum detection speed.	0%-100% of speed setting
RADAR_maxspeed	Set maximum detection speed.	0%-100% of speed setting
RADAR_threshoffset	Change threshold offset.	0-60dB
RADAR_filttype	Set tracking filter type.	0 = Standard 1 = Fast detection 2 = Long visibility
RADAR_minangle	Change minimum detection zone angle.	-30 - 30°
RADAR_maxangle	Change maximum detection zone angle.	-30 - 30°
RADAR_directset	Change detection direction filter.	0 = Receding 1 = Approaching 2 = Both
RADAR_nexttdat	Get next tracked frame data.	struct Radar_data
RARAR_readparam	Read complete radar parameter structure.	struct Radar_param

Table 2: Functions in radar.c of the TDWS project

Setting	Humans	Cars	Trains
Speed Setting	100km/h	100km/h	200km/h
Range Setting	100m	100m	100m
Min Range	2%	10%	10%
Max Range	15%	20%	25%
Min Angle	-15°	-15°	-15°
Max Angle	15°	15°	15°
Min Speed	14%	16%	32%
Max Speed	30%	40%	73%

Table 3: Radar setting for each mode.

will continue to ask the radar till it does.

Were the programmer interested in the output of radar's data, we have made functions that will writes a header through UART4 in RADAR_printhead and another function that prints the tracked data separated by commas in RADAR_printdata where you pass the data struct.

6.2.3 LoRa Interfacing

Another crucial peripheral is the WIO-E5 MINI DEV BRD STM32WLE5JC (LoRa Module). The LoRa module communicates via UART. The PIC24 contains several UART modules. UART2 was selected for this task. UART2 was selected because UART1 is used for the radar communications. The LoRa module can communicate at several different baud rates, however for this application the 115200 baud rate was chosen. The communication width is also only 8 bits as it sends and receives ascii characters. Due to how the AT command structure works, this UART operates on interrupts. Specifically the receiver interrupt. Shown in Figure 15 is the UART2 setup and pin selection. The receiver and transmitter are on Port D10 and D11 respectively.

A major note, the first time the LoRa module is used it must be manually set to use a baud rate of 115200. This can be done using a PuTTY terminal configured according to the datasheet of the LoRa module.

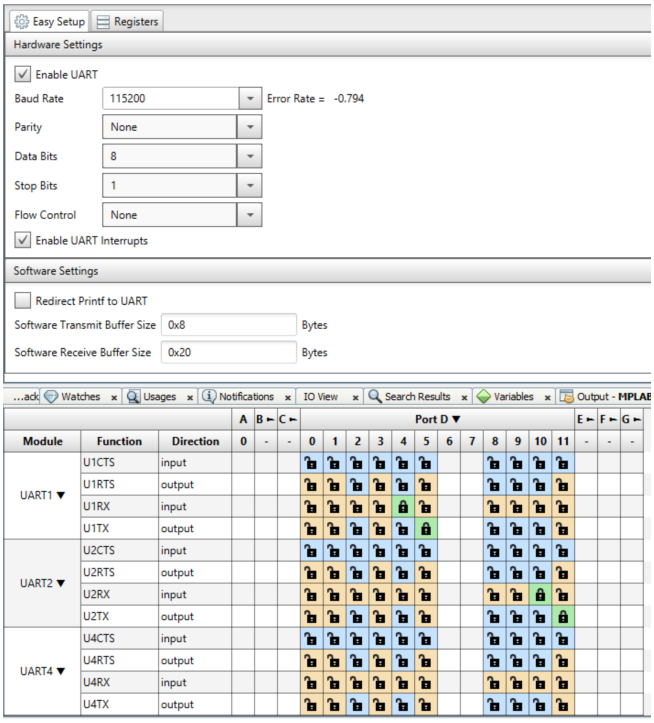

Figure 15: UART2 setup for communication with LoRa Module.

The reason that the UART2 interrupt is enabled is due to the structure of the AT commands. The AT commands are different lengths depending on the command. The only thing they have in common is that every command ends in a newline and carriage return character. So, using the newline character as when to end the UART2 can effectively read any message that it receives. The UART2 transmit interrupt is not utilized since sending the command over UART is simpler. Another reason for utilizing the receiver interrupt is because when the LoRa is set to its receiver mode, it can unpredictably send a message over UART. Because this is an asynchronous communication the interrupt is utilized to know when the LoRa module received data.

The UART2 code from the MCC contains a UART2_Read function. This function reads one byte from the PIC24s internal receiver buffer. Using this an interrupt service routine (ISR) is written that reads one byte upon interrupt and stores it in a global array. The ISR counts how many bytes it has read until it sees that last byte is a newline character, at which point it resets its character counter and waits to receive another message.The global array that the ISR writes to can then be read by other functions.

The LoRa module has several important commands that are utilized. The commands that are utilized are shown in Table 4. These commands are sent over UART2 in ASCII format.

Function	**AT Command**	**Response**	**Purpose**
check_lora	AT	+AT: OK	Check if LoRa is active and okay
lora_set_mode	AT+MODE=TEST	+MODE: TEST	Put LoRa module into test mode for point-to-point communication
lora_rfconfig	AT+TEST=RFCFG, 915,SF10,125,12,15,9, ON,OFF,OFF	+TEST: RFCFG F:915000000, SF10, BW125K, TXPR:12, RXPR:15, POW:9dBm, CRC:ON, IQ:OFF, NET:OFF	Sets the radio frequency configuration of the LoRa module (must be in test mode)
lora_set_rx	AT+TEST= RXLRPKT	+TEST: RXLRSTR	Sets LoRa module to receiver mode (must be in test mode)
LoRa_transmit_msg	AT+TEST= TXLRSTR,"string to transmit"	+TEST: TXLRSTR "string to transmit" +TEST: DONE	Transmits a message from the LoRa module (must be in test mode)

Table 4: LoRa AT Command Table

Each function in Table 4 utilizes the UART2 module. The function LoRa_transmit_msg sends any message based on a string input.

6.2.4 Transmitting Data To a PC

An important part of the TDWS project is being able to gather data and debug it. To gather information from the PIC24, the UART4 module is set to be able to communicate with a computer. **An additional external TTL device (such as the CP2102) is needed to facilitate this communication**.

Various functions within the code utilize UART4 to send data to a PC. The magnetometer.c file sends the axial data over UART4. In addition the Radar uses UART4 to transmit data.

UART4 is not essential for the operation of the project, however it is in the program so it can be utilized for debugging and data collection. The setup for UART4 is shown in Figure 16.

6.2.5 Other PIC24 Modules Used

The PIC24 has 5 timers. Of the timers, 3 were utilized for this project. This leaves 2 timers that can be used in the future for furthering the functionality of the project. Table 5 shows the 5 timers and their respective functions. Each timer is set up differently depending on the function.

Timer 2 is setup to run at 200us. This is for adequate sampling of the magnetometer data. Running faster would be optimal, but that would require a faster clock than the PIC24 is equipped with. This timer also

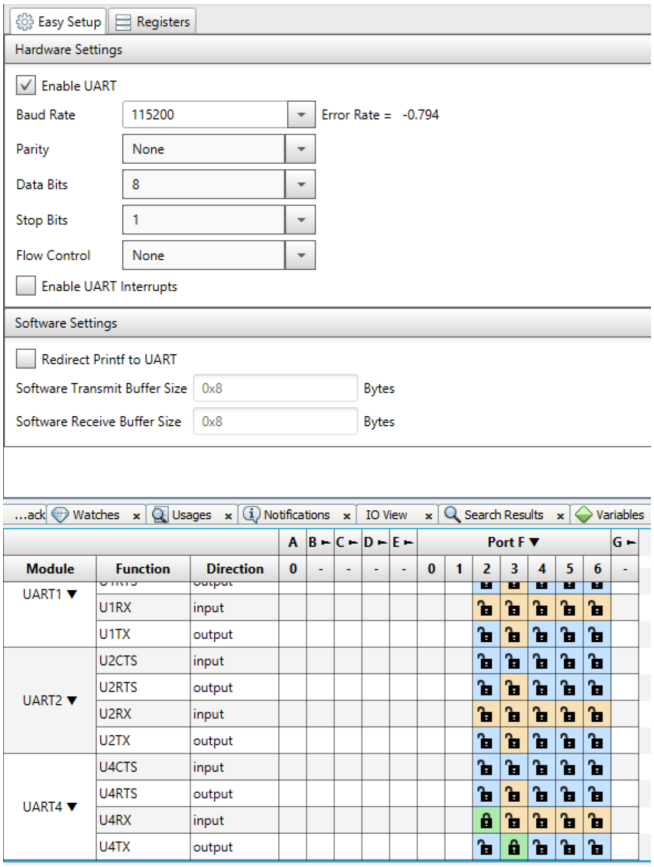

Figure 16: UART4 setup for PC communication

Timer	Function
Timer 1	Unimplemented
Timer 2	LIS3MDL data sampling
Timer 3	Timer to turn off the lights
Timer 4	Flash lights alternately
Timer 5	Unimplemented

Table 5: Software Modes Based on Switch Inputs

interrupts and is managed by an interrupt service routine (ISR). When in the ISR the magnetometer is polled by the PIC24 for each of its axis of data.

Timer 3 is set to disable timer 4 after 3 seconds. The interrupt for this timer is also enabled, where it's ISR disables timer 4.

Timer 4 is set to alternate which light is on at a rate of 600ms. The interrupt for this timer is enabled such that when in the ISR, the pins connected to the lights are toggled.

Timer 1 and timer 5 are currently unimplemented. However, in the future they could be used as heartbeat timers to ensure proper functioning of the detection modules.

6.3 Top Level Program View

After setting up each peripheral, a top-level view of the program can be discussed. Figure 17 shows the very beginning of the program. Firstly, the system is initialized, this is done through MCC generated code

which initializes the clock, UART, timer, and SPI modules. What is manually initialized in this phase is the timer interrupts. MCC generates callback functions which can then be pointed to a desired ISR. Also initialized is the magnetometer settings. After completing initialization, the mode switches are read. Depending on the value read from the switches the function goes to the associated mode. Table 6 shows all the possible modes with their associated switch values and hex values.

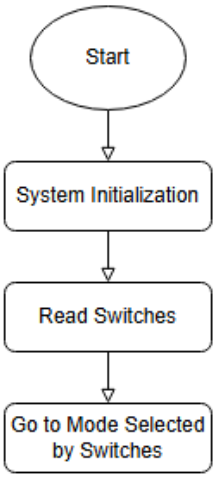

Figure 17: Program startup diagram.

Hex Value	SW3	SW2	SW1	SW0	Function
0x0	0	0	0	0	Main Module
0x1	0	0	0	1	Detection Module 1: People Tracking
0x2	0	0	1	0	Detection Module 2: People Tracking
0x3	0	0	1	1	Detection Module 1: Car Tracking
0x4	0	1	0	0	Detection Module 2: Car Tracking
0x5	0	1	0	1	Detection Module 1: Train Tracking
0x6	0	1	1	0	Detection Module 2: Train Tracking
0x7	0	1	1	1	LoRa Reciever Testing
0x8	1	0	0	0	Radar Testing
0x9	1	0	0	1	Magnetometer Testing
0xA	1	0	1	0	LoRa Transmit Testing
0xB	1	0	1	1	Light Testing
0xC	1	1	0	0	Magnetometer Detection Testing
0xD	1	1	0	1	Unimplemented: Defaults to Main Module
0xE	1	1	1	0	Unimplemented: Defaults to Main Module
0xF	1	1	1	1	Main Module

Table 6: Software modes from switch values.

After startup, the software goes to the selected mode. If no mode has been defined for a value of the switches, it will default to main module. The main module is the software mode that listens to LoRa messages and activates the lights. This mode uses the magnetometer as its sensor to detect changes in the magnetic field around the module. The main module flow is shown in Figure 18. Firstly, the LoRa module is initialized as a receiver, this is process is described in Section 4.2.3. Not shown in the diagram is the magnetometer data calibration. A function called calibrate_mag_data is called in which the ambient

magnetic field axis are averaged over a period of time. The average values are then saved and in subsequent magnetometer readings to remove the bias fo earth's magnetic field. Then it waits until the LoRa module gives the PIC24 a message. The message will have the format shown in Figure 7. If the error character is '0' the program continues on to read the the detection status character. If the character is a '1' it activates the lights (by turning on timer 4) and begins reading the magnetometer data.

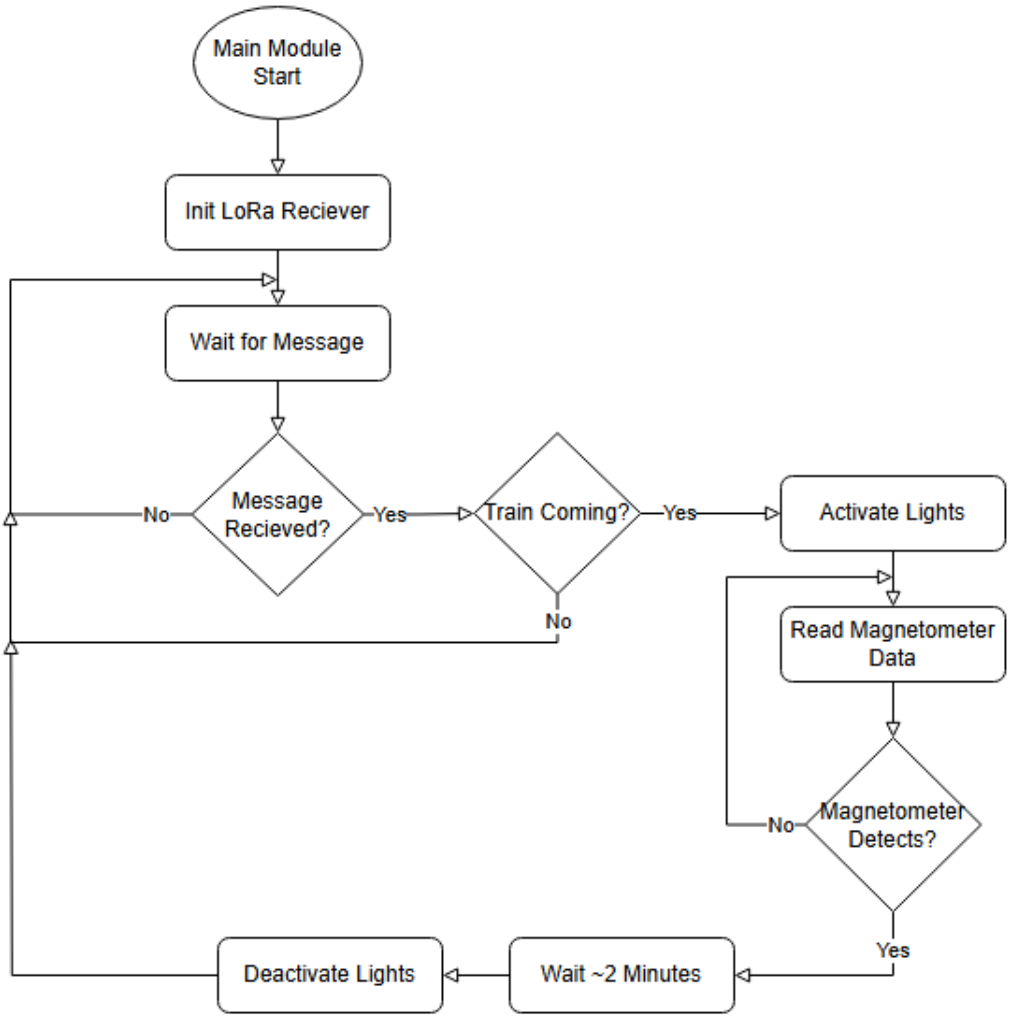

Figure 18: Main module software flow diagram.

In the "Read Magnetometer Block" the program executes a function called detect_train. This block takes a given sample of magnetometer data (from each axis), removes the bias using the previously calibrated value, calculates the magnitude of the three axis, puts the magnitude through and finite impulse response (FIR) low pass filter (LPF) then compares the value against a threshold value. If above the threshold it returns a 1, otherwise it returns a 0. If the detect_train function returns a 1 the software turns on timer 3 which turns timer 4 off after the set amount of time.

The next major function of the software is the detection modes. Figure 19 shows the software flow diagram of the detection module mode. In this mode the radar is the sensor that the PIC24 uses to detect objects. The first thing in this mode is initializing the radar to the settings based on the modes in Table 6, and the LoRa module is initialized as a transmitter as described in Section 4.2.3. Once those are initialized it goes straight to reading the Radar data. Upon seeing a target the software looks at the speed parameter, and based on the positive (receding) or negative (approaching) speed value determines whether to transmit

C0, C1, C2, C3	
Character	Function
C0	Module Status Unimplemented
C1-C2	Module ID '00' - Detection Module 1 '01' - Detection Module 2
C3	Detection Status '0' - Did not detect and object via radar '1' - Detected object via radar

Table 7: LoRa Message

of an oncoming target. If the speed is negative the object is approaching the radar and therefore going towards the crossing. Upon a negative speed the module transmits the message according to Table 7. The radar can "see" the approaching object multiple times, the module sends a transmission every time it sees an approaching target. This makes it so that even if one transmission is missed there will be multiple more coming after the missed transmission. Once it stops seeing a target approaching the module stops transmitting a message.

The other modes are testing modes that were convenient to have when implementing all the devices into one project. The process for developing the software was to create a separate project to interface with a device and once that was working take what was gleaned from the separate project and implement it into the main project. Out of this process came some test modes.

The lora_receive_test, which is selected according to Table 6. This mode sets the LoRa module to receive and then just reads the message.

The lora_transmit_test, selected according to Table 6. This mode initializes the LoRa module to transmit and repeatedly transmits a test message.

The light_test mode (selected according to Table 6) activates timer 3 and timer 4 to demonstrate the toggling of the lights and then the subsequent shutoff after a period of time.

The mag_detection_test (selected according to Table 6) is a simpler version of the main module. It calls the detect_train function but does not turn on the lights. It simply drives a pin high or low, where the pin can be connected to an LED to indicate when the magnetic field was above the threshold.

Similarly to the mag_detection_test, the radar_test (selected according to Table 6) simply reads the radar data and activates a pin when the radar detected something.

The mag_test (selected according to Table 6), is implemented to test the filtering of the magnetometer data. It does the calibration, then subsequently reads, removes the bias, calculates the magnitude, and filters the data. After filtering, the data is converted to binary coded decimal and then communicated over UART4. This allows for one to see save and graph the data on a computer.

Depending on which module you are testing the behaviors will be slightly different. Unfortunately, each module must be manually reset upon powering up. For the main module this requires holding the main reset button for 10 seconds, while holding the main reset button the reset button on the LoRa module

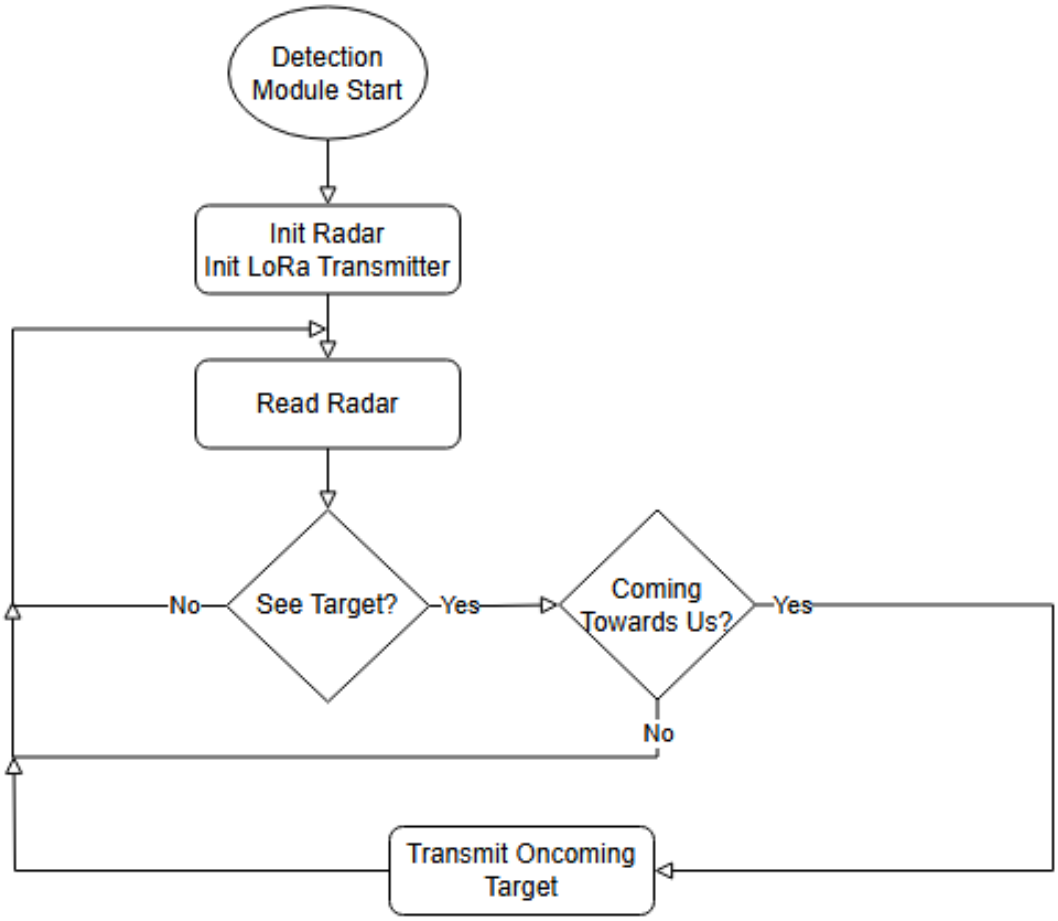

Figure 19: Detection module software flow diagram.

must be held for approximately 5 seconds. The main reset button can be let go about 2-5 seconds after the LoRa reset button is let go. This resets the software of the PIC24 and allows for proper operation.

The reset process for the detection module is simpler as only the main reset button needs to be held. It should be held for approximately 5-10 seconds. This allows the PIC24 to reset and begin running properly.

7 Data

Some final data from the tests was graphed. Figure 20 shows the detection rate as the distance between the magnetometer and the car increased. Figure 21 shows the final processed magnetometer data.

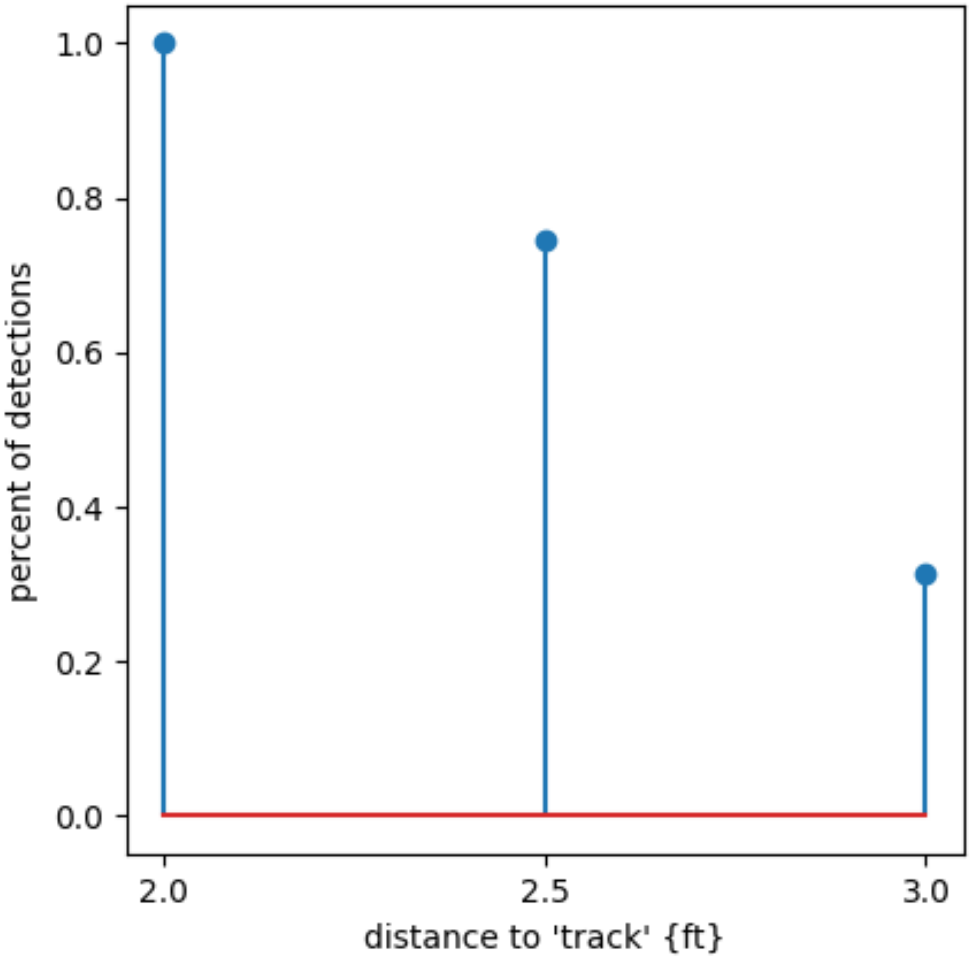

Figure 20: Magnetometer distance testing with a car.

Some information about the power consumption was also collected. Table 7 shows the measured power consumption of the detection module operating normally. Table 7 shows the measured power consumption of the main module operating normally. Table 7 shows the power consumption in the case where the lights on the main module would be continuously operating. Table 7 shows potential power generation from the solar panel, the charge time of the battery from empty to full, and the time it would run in the error operation if charging was not happening.

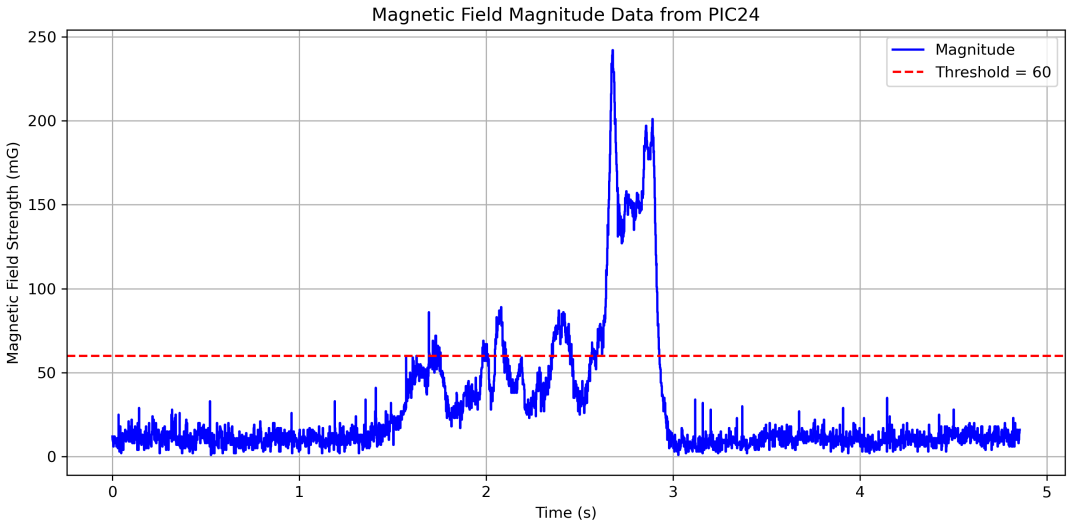

Figure 21: Frontrunnter magnetic field data after removing bias, calculating magnitude and filtering.

Normal Operation - Detection Module

Measured	Voltage (V)	Current (A)	Power (W)	Time (h)	Wh	Ah
Detection Module (Idle)	12.6	0.054	0.680	22.8	15.513	1.231
Detection Module (Transmitting)	12.6	0.06	0.756	1.2	0.907	0.072
Total					**16.420**	**1.303**

Normal Operation - Main Module

Measured	Voltage (V)	Current (A)	Power (W)	Time (h)	Wh	Ah
Main Module (Idle)	12.6	0.0267	0.336	20.4	6.863	0.545
Main Module (Active Lights)	12.6	0.94	11.844	3.6	42.638	3.384
Total					**49.501**	**3.929**

Error Operation - Main Module

Measured	Voltage (V)	Current (A)	Power (W)	Time (h)	Wh	Ah
Main Module (Idle)	12.6	0.0267	0.336	0	0	0
Main Module (Active Lights)	12.6	0.94	11.844	24	284.26	22.56
Total					**284.26**	**22.56**

Power Summary

Peak Sun Hours	5
Solar Power (100W Panel)	500 Wh/day
Battery Capacity	50 Ah
Days Without Charging (Worst Case)	2.216
Days to Charge (10.5V to 12.8V)	1.5

8 Final Thoughts

This project is a prototype that could definitely be improved upon. As a suggestion, instead of having timer 4 activate once and then just go, have it reset every time a spike in the magnetic field happens. This would allow for better knowing of when the train ends. For each spike of magnetic field, we wouldn't know exactly where along the train we are but if the spikes stopped happening the timer would eventually count all the way down, deactivating the lights.

For another suggestion, the magnetometer sensor could be entirely traded out for a different kind of sensor, such as the radar or a vibration sensor or even and IR break beam sensor. Of course changing the sensor changes the challenges that will be encountered on determining when a train is or isn't there.

For some finishing thoughts, this project could definitely benefit from more robust code. One thing that should be implemented is a heartbeat timer that is reset upon receiving a LoRa message. This would help with making sure that the detection module is still active and functioning properly.

In addition, more error checking for the peripherals should be implemented, i.e. what happens if the LoRa module stops working, or the radar, or the power MOSFET. There are several things that could just stop working and the PIC24 itself may never know. There are lots of issues within the code like this that need to be addressed.

Overall, this project does what it set out to do, detect a train, turn on lights, and then eventually turn them off.

Appendix A: Bill of Materials

The final cost of what was made in this project (accounting for only the components that were on the final prototype).

	Individual Cost w/o tax	**Amount**	**Amount x Cost**
ABS Electrical Junction Box 15.7"x11.8"x7.1"	$79.99	2	$159.98
Renogy 10 Amp 12V/24V PWM Solar Charge Controller	$24.99	2	$49.98
OSH Park PCB Printing	N/A	3	$130.95
PIC24FJ128GL306-I/PT	$2.75	3	$8.25
PICkit 5 In Circuit Debugger	$94.99	1	$94.99
WIO-E5 MINI DEV BRD STM32WLE5JC	$21.90	3	$65.70
SLA12-8F Duracell Ultra 12V 8AH AGM Sealed Lead Acid	$42.99	2	$85.98
LIS3MDL Magnetometer	$9.95	2	$19.90
K-MD7-RFB-00H-02 Radar	$168.83	2	$337.66
595-TPSM84203EAB	$5.50	3	$16.50
SIB912DK-T1-GE3	$0.76	1	$0.76
EveRRay, Grade Crossing Lights (Current Limiting Resistor)	$150.00	2	$300.00
Stand	$54.95	2	$109.90
Fixed Terminal Blocks 2P 5.08mm 90DEG	$1.07	12	$12.84
Multilayer Ceramic Capacitors MLCC (various types)	N/A	27	$13.41
White LEDs	$0.27	12	$3.28
Headers & Wire Housings (various types)	N/A	9	$11.10
DIP Switches / SIP Switches	$1.11	3	$3.33
Thin Film Resistors (various types)	N/A	16	$9.10
Total			**$1,433.61**

Table 8: Component Costs for the Project

Appendix A: Code and PCB Schematics

For the code and detailed schematics, please go to the following github page.